Minnesisolering för virtuella maskiner

en hypervisorstudie

Mathias Pedersen-Sandbakken matped@kth.se
Mattias Uskali uskali@kth.se
Examensarbete i datalogi 15 HP
Programmet för datateknik
Kungliga Tekniska Högskolan
Handledare Mads Dam

Sammanfattning

Virtualisering har återigen fått en större uppmärksamhet tack vare molntjänster. Molntjänster ger företag möjlighet att sänka sina IT kostnader. Säkerheten i molnet är en viktig fråga för företag som planerar att införskaffa molntjänster. Då molntjänster bygger på virtualisering är det således viktigt att den hypervisor som hanterar virtualiseringen säkert virtualiserar minne. Denna uppsats studerar minnesvirtualisering i två kända hypervisors, Xen och VMWare. Hypervisorns design är beroende av hårdvarans stöd för virtualisering. Därför studeras även virtualiseringsstöd för minne från två stora processortillverkare AMD och Intel i uppsatsen. I uppsatsen visas hur Xen och VMWare isolerar det virtuella minnet mellan virtuella maskiner.

Abstract

Virtualization has yet again received more attention due to cloud computing. Companies are presented the opportunity to lower the cost when using cloud computing, security is an important question for the companies planning to introduce cloud computing. It is therefore important that the information stored in the cloud is kept isolated. The isolation is obtained through the virtualization that the cloud is based on. This report studies two well known hypervisors Xen and VMWare. The report examines the techniques used in those two hypervisors to isolate the memory. The report also studies the virtualization support given by hardware manufacturers AMD and Intel.

Förord

Vi vill tacka Mads Dam för det stöd vi erhållit under arbetets gång. Arbetet har mestadels utförts sida vid sida. De fåtal timmar då arbetet skett åtskiljt har kommunikationsmedel använts för diskussion.

Innehållsförteckning

Ordlista

AMD-V – AMD Virtualization, processorarkitektur från AMD med stöd för virtualisering.

AMD – Advanced Micro Devices, företag som utvecklar processorer och liknande teknik.

ASID – Adress Space Identifier, förklaras på sidan 23.

Bare Metal Hypervisor – förklaras på sidan 13.

BIOS – Basic Input/Output System, standard som starta datorns I/O-enheter samt OS.

Context Switch – förklaras på sidan 15.

CPU – Central Processing Unit, processorn i en dator.

CR3 – Basregister som visar var sidtabellen finns sparad i primärminnet

Hypercalls – Mjukvarutrap från virtuell maskin till hypervisor.

I/O – Input/Output, dataflöde till/från exempelvis en dator.

ISA – Instruction Set Architecture, förklaras på sidan 13.

Kernel – kärnan för ett OS, hanterar resurser samt kommunikation med hårdvara.

MMU – Memory Management Unit, komponent för att hantera minnesaccesser från CPU.

OS – Operativsystem.

Page Table – Se sidtabell.

PFN – Page Frame Number, sidplatsnummer.

Sidtabell – förklaras på sidan 17.

Syntetiska drivrutiner – Drivrutiner som används för att förbättra kommunikationen mellan virtualiseringsmjukvaran och det gästande operativsystemet.

TLB – Translation Lookaside Buffer, förklaras på sidan 19.

Trap – Undantag eller fel, synkront avbrott orsakat av ett exceptionellt villkor.

VM – Virtuell Maskin.

VMM – Virtual Machine Monitor, annat namn för hypervisor.

VPID – Virtual-Processor Identifier, förklaras på sidan 23.

VPN – Virtual Page Number, virtuellt sidnummer.

x86 – populär processorstandard.

1.Introduktion

1.1 Inledning

Den senaste trenden inom IT-världen är cloud computing (molntjänster). Molntjänster har uppnått stor popularitet både hos IT-chefer och ekonomichefer inom stora företag. Molntjänster anses vara en av de 10 viktigaste teknologiska strategierna för företag 2011 [1].

Med molntjänster avses processorkraft, lagringsutrymme och applikationer som finns på servrar i en eller flera datorhallar vilka kunden får åtkomst till via internet. Dessa tjänster säljs i form av applikationer, Software as a Service (SaaS), IT-infrastruktur, Infrastructure as a Service (IaaS) och processorkraft, Platform as a Service (PaaS).

Molntjänsterna debiteras efter användning, vilket medför att de (i teorin) är gratis då de ej används, vilket tilltalar de ekonomiansvariga i företagen [2]. Det kräver inget initialt investeringsbehov och ingen egen spetskompetens för att underhålla IT-miljön.

En ytterligare fördel med molnet är att servrarna används effektivare och bättre nyttjar servrarnas resurser, vanligtvis utnyttjas inte hårdvaran till fullo. Med virtualisering kan flera system köras på samma server. Då nyttjas hårdvarukapaciteten till en högre grad vilket medför att färre antal fysiska servrar behövs. Detta är bra för miljön då det minskar materialåtgång och energiförbrukning. När en server närmar sig sin maxkapacitet ger virtualisering en möjlighet att migrera gästsystem till en annan server.

Under 60 och 70 -talet var virtualisering ett hett område inom datalogin då hårdvaran ofta var dyr och krävde stor plats. Att då kunna köra flera maskiner på samma hårdvara var en stor fördel. En ytterliggare fördel med virtuellt minne var att applikationerna nu kunde presenteras med ett näst intill oändligt kontinuerligt minne. Detta medförde att programmeraren inte behövde ladda in olika programdelar i minnet, istället fick hela programmet plats i minnet [25]. I takt med att kostnaden för hårdvaran sjönk minskade intresset för virtualisering. Nu med bland annat molnet har virtualisering återigen fått ett uppsving vilket medfört att hårdvaruleverantörer har konstruerat stöd för virtualisering i sina nya produkter.

1.2 Bakgrund

Det som idag gör företag osäkra till att investera i molntjänster är säkerhetsaspekterna [4]. De allra flesta företag hanterar information som är känslig och de har krav på hur sådan information får hanteras. Denna typ av information kan exempelvis vara personal- eller kundinformation. Molntjänster gör att det finns många säkerhetsaspekter att ta hänsyn till då mycket information finns lagrad på samma plats. Det som oroar företagen är de risker som existerar då informationen skickas över internet, hur information lagras hos molnleverantören, hur molnleverantörerna hanterar den lagrade informationen samt att flera företag exekverar flera Virtuella Maskiner (VM) på samma server. En hypervisors uppgift är att isolera VM från varandra för att säkerställa att information ej hamnar i fel händer.

Hårdvarustödet för virtualisering har utvecklats på senare år med bland annat AMD-V. Detta medför att hypervisorn blir mindre komplex eftersom den kan dra nytta av hårdvaran [17]. Det finns vissa skillnader i hur man bygger en hypervisor och dessa skillnader är beroende av vilken arkitektur som ska virtualiseras.

Enligt Popek och Goldberg [32] existerar tre krav som bör uppfyllas av en hypervisor.

1. Exakthet, mjukvara som körs i en VM ska exekvera exakt som det gör när det exekverar direkt på hårdvaran.
2. Prestanda, En stor mängd av mjukvarans instruktioner ska kunna utföras utan hypervisorns inblandning.
3. Säkerhet, hypervisorn ska kontrollera all hårdvaruåtkomst.

Man kan alltså sammanfatta detta till att hypervisorn ska vara transparent så att mjukvaran i en VM inte uppfattar att den virtualiseras. Hypervisorn ska dessutom säkerställa isolering och korrekthet så att en VM endast kan komma åt sin egen information, vilket medför säkerhet.

1.3 Syfte

Molntjänster bygger på virtualisering. Virtualiseringslagret ska delge hårdvaruresurser till de molntjänster som exekverar på en server hos en molnleverantör. Dessutom ska virtualiseringen skapa isolering mellan molntjänsterna. För företag är denna isolering oerhört viktig och ett orosmoln för IT-ansvariga hos de företag som planerar att införskaffa

molntjänster. För att isolering ska fungera korrekt krävs att minnet hos de virtuella maskinerna är korrekt isolerade från varandra.

Syftet med uppsatsen är att studera hur minne virtualiseras hos några av de hypervisors som används i dagens molntjänster. Fokus ligger på minnesisolering, då denna är en central del i informationssäkerheten för molntjänsten. Ytterligare en central aspekt är hypervisorns prestanda som påverkas av hårdvarustödet hos den hårdvara som virtualiseras.
För att söka svar på syftet har vi undersökt minnesvirtualisering i två hypervisors Xen och VMWare. Vi har dessutom undersökt det hårdvarustöd för virtualisering som lanserats av hårdvarutillverkarna AMD och Intel.

1.5 State of The Art

En hypervisors uppgift är att virtualisera de resurser hårdvaran erbjuder i form av Central Processing Unit (CPU), minne och I/O samt att skapa en virtualiserad miljö. Denna miljö är mer känd som en VM. Fler VM:s kan köra på samma maskin och får åtkomst till hårdvaran via hypervisorn. Vi kommer här redogöra för några olika State-of-The-Art hypervisors som används i dagsläget.

1.5.1 Xen Hypervisor

Xen Hypervisor används bland annat i molntjänsten Amazon Web Services. Xen körs på x86 processorarkitekturen och denna arkitektur är tämligen svår att virtualisera då den inte har hårdvarustöd för att virtualisera CPU, minne och I/O [9, 10, 11]. Detta medför att hypervisorn blir betydligt mer komplex i förhållande till om hårdvarustöd för virtualisering existerat. Enligt Xen själva är deras prestanda nära inpå den prestanda som hårdvaran kan prestera om den körs utan Xen Hypervisor [11].

1.5.2 VMWare ESXi

ESXi använder sig av privilegierade ringar i fyra nivåer, från ring 0 som är den lägsta men mest privilegierade ringen till ring 3 med minst privilegier. I ring 0 körs hypervisorn, VM körs i ring 1 medan applikationer i VMs exekveras i ring 3. Varje VM innehåller ett komplett system med exempelvis CPU och BIOS. Vilket OS som helst kan användas i ESXi, till skillnad från de övriga hypervisors vi undersökt [18].

1.5.3 Microsoft Hyper-V

Hyper-V exekveras på en utveckling av den arkitektur som Xen exekveras på x86-64 processorarkitektur, vilken har stöd för virtualisering i hårdvaran [1]. Hyper-V används tillsammans med Windows Server 2008 eller en Server Core. Isolering sker genom partitioner där OS exekveras [13]. Windows Server Hyper-V inkluderar syntetiska drivrutiner, vilket resulterar i prestandaförbättring då detta reducerar antalet gånger CPU:n behöver byta mellan kernel-mode och user-mode [13].

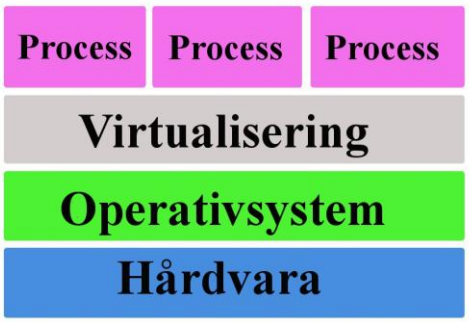
2. Virtualisering

Virtualisering är en abstraktion av ett datorsystem, där ett lager av virtualiseringslogik tillhandahåller och hanterar virtualiserade resurser till en klient [9]. Goldberg liknar en VM vid en avbild av ett eller flera kompletta system [7]. Enligt Mallach ger även Goldberg definitionen av virtualisering som:
”*A system … which … is a hardware-software duplicate of a real existing machine, in which a non-trivial subset of the virtual machine's instructions execute directly on the host machine...”* [8].
Som följd av dessa definitioner kan man säga att virtualisering handlar om att ge en eller flera VM en illusion av att de har ensam tillgång till hårdvarans resurser.

Enligt Douglas och Gehrmann är de två mest framträdande virtualiseringsformerna processvirtualisering och systemvirtualisering [9]. En likartad uppfattning har Daniels som menar att det finns 3 typer av virtualisering: mjukvaruvirtualisering, hårdvaruvirtualisering och containers [6]. Ur beskrivningen av dessa typer av virtualiseringar kan man dra slutsatsen att mjukvaruvirtualisering och processvirtualisering är samma typ av virtualisering. På samma sätt är hårdvaruvirtualisering och systemvirtualisering även dessa samma typ av virtualisering. Det som skiljer Douglas och Gehrmanns syn på virtualisering med Daniels är således containers. Vid systemvirtualisering virtualiseras hårdvaran till ett gästande operativsystem till skillnad mot processvirtualisering där hårdvaran virtualiseras till en applikation som exekveras i ett operativsystem. Systemvirtualisering kan uppnås med hjälp av paravirtualisation, containers eller ISA-översättning. I detta kapitel studeras och beskrivs olika typer av virtualisering. Detta kommer sedan att leda in på hypervisors vilket studeras i efterföljande kapitel.

2.1 Processvirtualisering

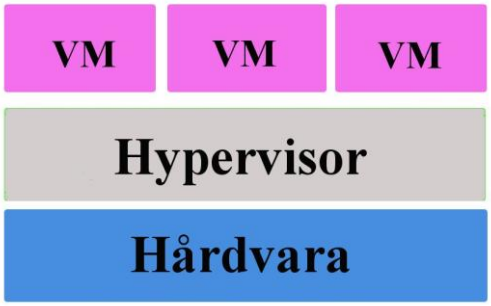
Vid processvirtualisering ligger virtualiseringslagret mellan OS och VM [6]. OS virtualiserar systemets resurser såsom minne, CPU och andra enheter. Vilket leder till att varje exekverande process erhåller intuitionen av full tillgång till hårdvarans resurser [9]. Av dessa definitioner kan vi dra slutsatsen att processvirtualisering sker mellan OS och de processer som exekveras, vilket visas i figur 1. Det krävs alltså att de processer som exekveras är designade för det OS som de exekverar i. Enligt Douglas och Gehrmann är processvirtualisering ett fundamentalt koncept i varje modernt operativ system. I dagens datorer exekverar flera processer samtidigt, processvirtualiseringen isolerar dessa processer från varandra och erbjuder processen en illusion av full åtkomst till hårdvaran.



Figur 1, Processvirtualisering

2.2 Systemvirtualisering

Vid systemvirtualisering virtualiseras hela datorsystemets miljö. Virtualiseringslagret mellan hårdvaran och VM benämns hypervisor eller Virtual Machine Monitor(VMM). Hypervisorn är en mjukvara som exekveras på hårdvaran för att virtualisera hårdvaran för VM. Hypervisorn har således till uppgift att hantera och fördela hårdvarans resurser mellan VM [9], vilket illustreras i figur 2. I uppsatsen studeras denna typ av virtualisering och hur dessa hypervisors virtualiserar minne till VM. Då det är denna virtualisering som används av molntjänstleverantörerna idag [10].



Figur 2, Systemvirtualisering

2.3 Containers

Container är mjukvara som virtualiserar omvärlden för ett OS. Dessa containers exekverar på samma underliggande kärna. Isolationen förhöjs genom att tillhandahålla så kallade väggar mellan grupper av processer. Vilket medför möjligheten att exekvera fler instanser i isolation från varandra inom ett OS. Containers emulerar ingen hårdvara, av detta skäl måste gäst-OS kommunicera med det värd-OS som i sin tur utför de korrekta anropen till hårdvaran. Eftersom kravet på ett värd-OS existerar blir inte tekniken med att använda containers lika flexibel då generellt de gäst-OS som exekveras måste vara av samma typ som värd-OS är [15].

2.4 Paravirtualisation

Med paravirtualisation menas den kommunikation som sker mellan det gästande systemet och hypervisorn [14]. Den gästande kod som exekveras modifieras så att den använder ett annat gränssnitt än den som finns för hårdvaran på systemet. Detta gränssnitt är antingen lättare eller säkrare att virtualisera och kommer att förhöja prestandan för systemet [9]. Gränssnittet byter ut icke-virtualiserbara instruktioner mot hypercalls, vilka kommunicerar direkt med det virtualiserade lagrets hypervisor [14]. Således möjliggör paravirtualisation virtualisering för ett system som exekveras på hårdvara där virtualisering inte stöds.

2.5 Ringar

Konceptet med ringar introducerades under 70-talet och det är en skyddsmekanism som tillåter olika typer av kod att exekvera i olika skyddade nivåer (ringar). Konceptet innebär att kod har tillåtelse att utföra olika hårdvaruinstruktioner beroende på vilken ring koden exekverar i. Kod som exekverar i den högst privilegierade ringen (ring 0) har således full tillgång till hårdvaran. Detta koncept utnyttjar hypervisorn som exekverar i ring 0 medan en VM exekverar i en lägre privilegierad ring. När kod i en lägre privilegierad ring vill utföra instruktioner vilka den ej har tillåtelse att utföra sker en trap till en ring som har tillåtelse att utföra denna instruktion [9].

2.6 ISA översättning

Instruction Set Architecture (ISA) är det lågnivå-gränssnitt som ligger närmast hårdvaran. Endast mjukvara som exekverar i den högst privilegierade ringen har tillåtelse att utföra ISA-instruktioner som hanterar hårdvaruresurser. När mjukvara som inte exekverar i privilegierat läge försöker använda dessa instruktioner genereras en trap till den mjukvara som exekveras i det privilegierade läget. För att kunna exekvera olika typer av OS i VM krävs en översättning mellan den ISA som gäller för hårdvaran och den ISA som OS är implementerat för. En sådan ISA översättning kan krävas för att helt virtualisera en miljö [9].

3. Hypervisor

Som tidigare nämnts är hypervisorn är det lager av virtualisering/abstraktion som ligger mellan hårdvaran och VM och dess uppgift är att hantera och fördela hårdvarans resurser. Av Goldberg beskrivs två typer av hypervisors, typ 1 och typ 2 [16]. Båda dessa ger en fullt virtualiserad miljö men uppnår det på olika sätt.

3.1.1 Typ 1

Typ 1 är vad som kallas för en bare-metal hypervisor och den ligger direkt ovanpå hårdvaran. Hypervisors av typ 1 kan uppnås på olika sätt. Dels genom ISA översättning som ej kräver någon modifikation av OS i aktuella VM. Det kan även uppnås med hjälp av paravirtualisation, detta kräver dock modifiering av OS hos VM.

3.1.2 Typ2

Denna typ av hypervisor körs i ett OS och ligger mellan OS och VM, det OS som ligger under hypervisorn benämns som värd-OS. Denna typ av hypervisor kan liknas vid processvirtualisation men det finns en skillnad eftersom denna typ virtualiserar ett komplett OS. Den här typen av hypervisor är vanlig bland persondatorer där användaren vill ha tillgång till olika OS snabbt och enkelt. Hypervisorn installeras som ett vanligt program över det OS som är installerat på hårdvaran.

3.2 Arkitektur

De hypervisors som studerats är av typ 1. Här ges en beskrivning av hur de skapar ett fullt virtualiserat system åt varje enskild VM.

3.2.1 Xen hypervisor

I Xens system som vi tidigare har behandlat är hypervisorn det lägsta och mest privilegierade lagret i hierarkin. Över detta lager ligger olika VM, vilka hypervisorn ger möjlighet att använda de resurser som servern erbjuder. Det första gästsystemet kallas med Xen-terminologi för domän 0, de övriga gäst-systemen kallas för domän U. Där U står för användare (engelskans user). Domän 0 startar automatiskt då hypervisorn startas och ges administrations privilegier, samt speciella rättigheter till den faktiska hårdvaran som standard. Administratören kan senare logga in och modifiera de gäst-system som exekveras i domän U. Som vi har nämnt tidigare använder sig Xen av något som heter paravirtualisation. För denna teknik måste gäst-OS modifieras för att kunna virtualiseras. I paravirtualisation finns det något som heter hypercalls. Ett hypercall är en mjukvaru-trap från en domän till hypervisorn. Domänerna använder sig av dessa hypercalls för att få tillgång till privilegierade operationer som de annars inte kan exekvera, exempelvis för att uppdatera sidtabeller.

3.2.2 VMWare ESXi

Till skillnad från den implementation som Xen använder sig av behöver man i ESXi från VMWare inte modifiera det gästande systemet. Dock finns det drivrutiner som man kan installera hos gästsystemet för att bättre sammanfoga systemet med hypervisorn. Hypervisorn tillgodoser här gästsystemet med ett gränssnitt som används som simulering av den riktiga hårdvaran. ESXi uppnår full virtualisering genom binära översättningar och direkt exekvering. Gästsystemet vet genom detta inte att den exekverar i ett virtualiserat system,

och behöver därmed ingen modifiering. Hypervisorn översätter alla gäst-systemets instruktioner och cachar allt om samma instruktion kommer igen.

3.3 Minnesvirtualisering

Att virtualisera minne är en välkänd teknik och används i de flesta OS [19]. Virtualisering av minne ger en process illusionen av att den har en stor sammanhängande minnesarea att tillgå. Genom att sprida ut det virtuella minnet på icke sammanhängande platser i det fysiska minnet och på disk skapas en bild av att det fysiska minnet är större än vad det i verkligheten är. Adresser som används av processen kallas för virtuella adresser, en samling av dessa kallas för en adressrymd. De adresser minnet använder sig av kallas för en minnesadress och en samling av dessa kallas för en minnesrymd. Hårdvaran ansvarar för att flytta information in i det fysiska minnet när och endast när det krävs av processen. Eftersom det inte finns någon korrespondens mellan virtuella adresser och minnesadresser ger detta en förhöjd komplexitet i adresseringsmekanismen. Därför används något som benämns sidtabell vilket kommer att beskrivas nedan [30]. En hypervisor använder sig utav virtualisering av minne i ytterligare en nivå. VM får då en illusion att den i sin tur har en stor sammanhängande minnesarea, precis som för processen i VM.
Vanligtvis virtualiseras minnet genom en virtuell Memory Management Unit (MMU) för varje VM som reflekterar varje VM:s minnesutrymme [29]. Virtualiseringen av minne kan vara både statisk och dynamisk. Att statiskt allokera minnet till varje VM medför större isolation och säkerhet än att utföra det dynamiskt [9]. Genom att statiskt allokera minne för varje VM erhålls starkt isolerade VM och det leder till att inga problem med osäker radering av information av tidigare använd disk-yta uppstår.

3.3.1 Context switch

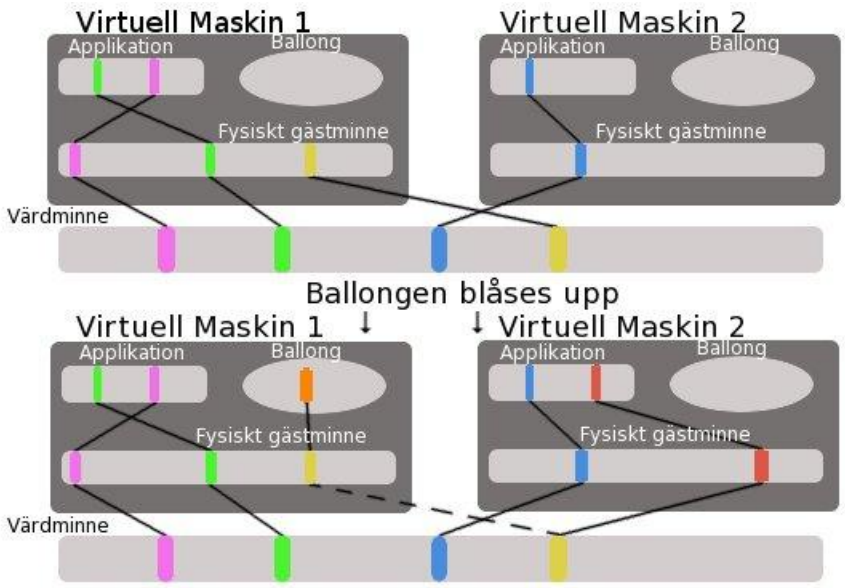
Context switch är processen av att spara och återställa tillståndet av en CPU så att den kan återuppta exekveringen från samma tillstånd vid en senare tidpunkt. I och med användningen av context switch är det möjligt att exekvera flera processer på en CPU. En vital del ur säkerhetssynpunkt för att context switch ska vara säker är att CPU och dess register rensas innan en annan VM får tillgång till CPUn då dessa register innehåller information om en annan VM.

3.3.2 Dynamisk allokering av minne

Problematiken med att statiskt allokera en minnesmängd för VM är att det är svårt att bedöma hur mycket minne en VM verkligen behöver. Allokeras för lite minne till en VM kan

det leda till att VM prestanda sjunker vid hög belastning på hårdvaran, detta trots att hårdvaran har ledigt minne som inte är allokerat av ett annat VM. Allokeras en stor mängd minne till varje VM kommer inte lika många VM att ha möjlighet att exekveras på servern. Att kunna allokera minne dynamiskt för VM är en av de pelare vi tidigare har berättat om som gjort molntjänster så framgångsrika.

Ballooning är en process som exekverar i de olika VM. Processen kan liknas vid en ballong och det finns en ballong i varje enskild VM. När det fysiska minnet hos hårdvaran är på väg att ta slut blåses ballongen upp och upptar fysiskt gästminne, detta utförs för att en VM ska minska minnesanvändning, vilket visas i figur 3. Dessa fysiska minnesadresser ges av hypervisorn till en VM med få lediga fysiska adresser. Då det är en drivrutin som körs direkt från OS:et i VM bestämmer den själv vilka minnesadresser den kan allokera och på så sätt göra tillgängliga för en annan VM. På samma sätt som ballongen utökas kan den även förminskas och ge tillbaka de virtuella minnesadresser till VM som den tidigare har allokerat [19]. Vi kommer här nedan att berätta om hur de olika hypervisors använder sig av ballooning för att ta hand om minnesallokeringen.



Figur 3, Ballooning

VMWares allokering fungerar genom att ett system kan exekvera exempelvis 3 VM med 2 GB minne var, medan hårdvaran endast har 4 GB minne. Detta kallas för overcommitment. Om någon av dessa 3 VM utsätts för hård belastning och det fysiska minnet börjar ta slut, utökas

ballongen och allokerar minnesplatser från VMs lista av lediga minnesadresser och kan ta dessa fysiska minnesplatser och ge dem till en annan VM.
VMWare använder sig utav fler tekniker än ballooning för att återta minne, vilket de benämner ”Memory reclamation”. Dessa är Transparent Page Sharing (TPS), Hypervisor swapping och Memory compression. Vilka kortfattat kan beskrivas:
TPS – tar bort redundanta sidor som innehåller exakt samma information.
Hypervisor swapping – återtar minne från VM genom att låta ESXi direkt swappa VMs minne.
Memory compression – komprimerar sidor som skall swappas.

Xen använder nyttjar ballooning på liknande sätt som VMWare. De har 4 olika gränser för minne, vilka är statiskt minimum, dynamiskt minimum, dynamiskt maximum och statiskt maximum. Dessa kan ändras i XenServer utan att behöva starta om VM. Beroende på hur långt det är mellan dessa gränser kan ballooning användas för att frigöra minne från en VM till en annan genom att hypervisorn blåser upp ballongen enligt ovannämnd process. XenServer säkerställer att summan av alla VM:s dynamiska minimum inte överstiger hårdvarans minnesstorlek [21].

3.3.3 Sidtabell

En viktig del i minnesvirtualisering är datastrukturen sidtabell. Här studeras sidtabellen och hur den används för minnesvirtualisering.

Tekniken för minnesvirtualisering med hjälp av sidtabeller beskrivs av Roos och Brorson [24, 25]. Den virtuella adressrymden delas upp i block som kallas för sidor. Primärminnet delas upp i block av samma storlek som adressrymden, denna uppdelning kallas för sidplatser eller sidramar (page frame). När en process exekveras fyller man sidramarna med de sidor som är aktuella för processen just då och de sidor som för tillfället ej är aktuella för processen ligger i sekundärminnet eller på disk. På så vis skapas en illusion av ett stort kontinuerligt minne för den enskilda processen.

Antalet sidor i sidtabellen är beroende av antalet bitar hos adressbussen samt storleken på en sida. De minst signifikanta bitarna i den virtuella adressen anger förskjutningen i sidan. Vilket medför att en större sida kräver fler bitar för att ange förskjutningen i sida och således blir antalet bitar att ange plats i sidtabellen mindre, vilket leder till en mindre sidtabell. Exempelvis gäller det att om adressbussen har 32 bitar och storleken på en sida är 512 byte erhålls 2^24 = 16777216 sidor.

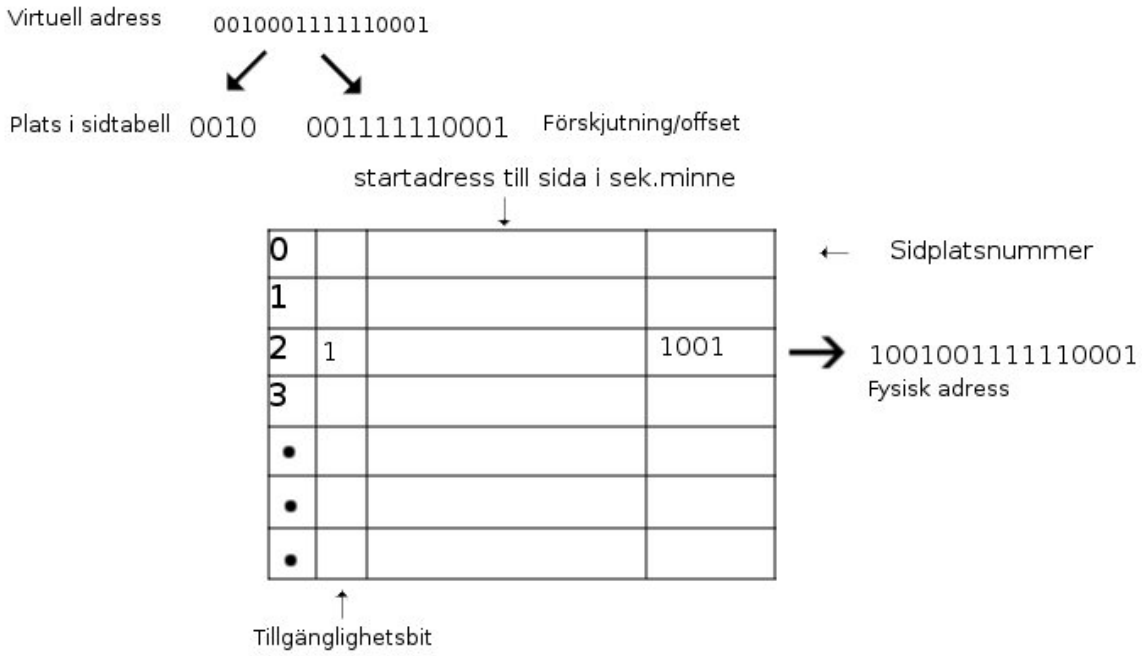
Processen att översätta en virtuell adress till en fysisk adress går till enligt följande:

1. De mest signifikanta bitarna i adressen som anger positionen i sidtabellen väljs ur den virtuella adressen.
2. Med hjälp av basadressen (som anger sidtabellens minnesadress) och positioneringen i sidtabellen erhålls sidans fysiska adress i minnet.
3. Med sidans fysiska adress och de minst signifikanta bitarna i den virtuella adressen som anger förskjutningen i den fysiska sidan erhålls adressen till den fysiska minnesplatsen.

I sidtabellen existerar även information om sidans tillgänglighet i primärminnet och sidans fysiska adress till disk om sidan ej finns i primärminnet. Informationen om sidans tillgänglighet finns i något som benämns som tillgänglighetsbit (valid bit). Är tillgänglighetsbiten ett-ställd finns sidan i primärminnet och sidans adress i primärminnet hämtas från tabellen. Ligger inte sidan i primärminnet ges sidans adress till disk ur tabellen [23, 24].

Basadressen till sidtabellen finns i CR3 registret i både AMD och Intel processorer [20, 28].

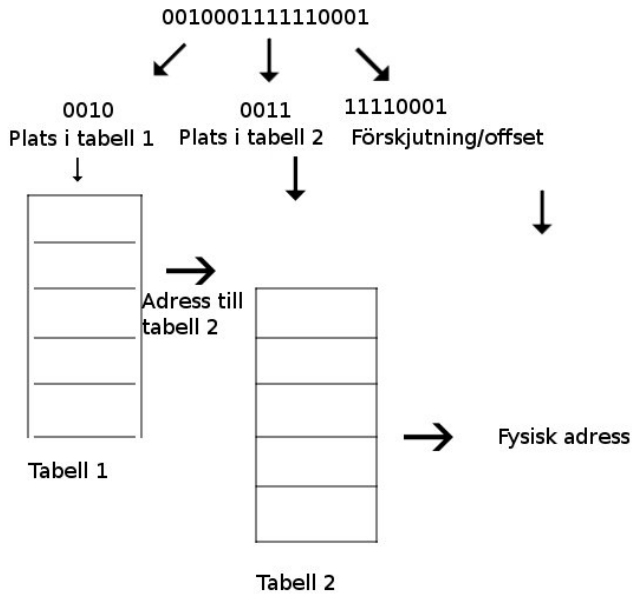
Figur 4 nedan visar adressöversättning från virtuell adress till fysisk adress med hjälp av sidtabell.



Figur 4, Adressöversättning med sidtabell

Sidtabellen som vilken annan datastruktur innehåller endast data och även den behöver lagras i minnet. För att uppslagning i sidtabellen ska effektiviseras lagras den i primärminnet. I exemplet ovan innehöll sidtabellen 16777216 sidor. Sidtabellen blir snabbt stor och kräver således mycket utrymme i primärminnet vilket inte är önskvärt. För att lösa detta problem delas ofta den virtuella adressen in i ett flertal delar där varje del anger position i en mindre sidtabell. Den stora sidtabellen delas alltså i en hierarki med fler mindre sidtabeller. Fördelen med detta är att endast de fåtal mindre tabeller som för tillfället är aktuella behöver lagras i primärminnet vilket leder till att man sparar utrymme utan minskad snabbhet [23, 24].

Processen som översätter den virtuella adressen till den fysiska adressen benämns som sidtraversering (page walk) [24, 20]. I stort sett alla moderna processorer har hårdvarustöd för sidtraversering [20]. Den nackdel som finns med denna uppdelning är att fler tabeller leder till att sidtraverseringen tar längre tid då det krävs fler tabelläsningar [20].
Figur 5 nedan visar sidtraversering där sidtabellen delats upp i två nivåer.



Figur 5, Adressöversättning med två sidtabeller

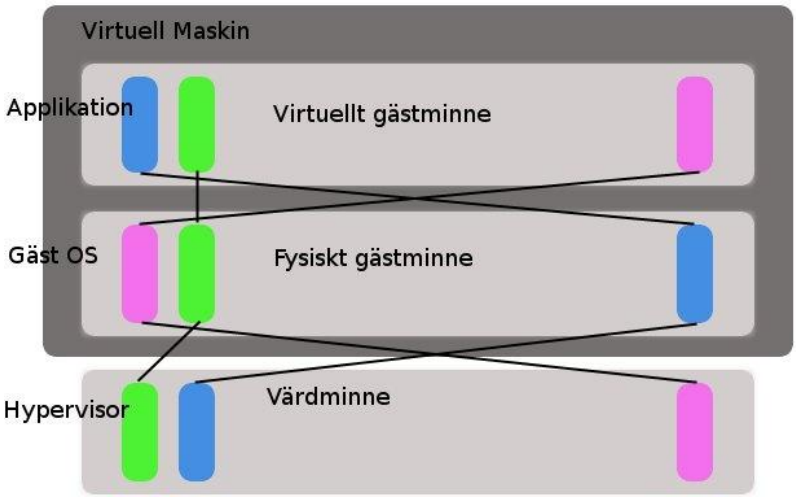
3.3.4 TLB

Translation Lookaside Buffer (TLB) är en cache som innehåller en delmängd av sidtabellerna. MMU:n använder sig utav TLB för att snabbare översätta en virtuell adress till en fysisk adress. Översättningen effektiviseras till följd av att sidtraversering ej utförs lika frekvent. Problematiken med denna teknik är att TLB måste rensas då ett OS exekverar processer parallellt med varandra på en singel processor. Innan en context switch måste TLB rensas för att den nya processen skall få tillgång att fylla TLB med sina egna översättningar. Detta måste göras eftersom de olika processerna har likadana adressområden vilka börjar från 0 och uppgår till en viss storlek. Dessa adressområden har samma virtuella adresser men olika fysiska adresser hos hårdvaran. Om en adressöversättning ligger kvar i TLB kan således en felaktig översättning göras. Detta skulle medföra en säkerhetsrisk, alltså måste TLB rensas vid context switch. Att rensa TLB är väldigt kostsamt eftersom en ny process måste fylla TLB med sina egna adressöversättningar (från virtuell adress till fysisk adress). Det är kostsamt då en sidtraversering måste utföras vid varje ny adressöversättning [20, 22].

3.3.5 Användning av sidtabell och TLB för minnesvirtualisering

Att virtualisera minne med sidtabeller enligt den teknik som beskrevs i föregående kapitel är välkänd och används i flertalet av de moderna OS [19]. Det kan ses som den del i processvirtualisering som virtualiserar minnet åt varje enskild exekverande process.

Vid systemvirtualisation krävs virtualisering av minne till varje enskild VM dels för att skapa isolering mellan VM [19] men även för att de OS som ska exekvera i VM är skapade för ett kontinuerligt minnesutrymme [27]. Dessutom ska minnesvirtualisering till de processer som exekveras i VM stödjas. Det medför att hypervisorn behöver virtualisera minnet i två lager, vilket visas i figur 6 nedan.



Figur 6, Två lagers minnesvirtualisering

De flesta processorer som idag finns i moderna datorer saknar stöd för denna minnesvirtualisering i två lager [9, 20]. Den teknik som används av både Xen och VMWare för att lösa detta benämns som skugg-sidtabell (shadow paging). Tekniken bygger på att hypervisorn exekverar i den högst privilegierade ringen och har full tillgång till hårdvaran.

I hypervisorn finns en adressöversättning mellan hårdvarans fysiska minne och det fysiska minnet hos VM. I Xen sparas denna översättning i en tabell medan det i VMWare sparas i en pmap. Pmap är en datastruktur som lagrar adressöversättning från gästens fysiska minne till hårdvarans fysiska minne [19].

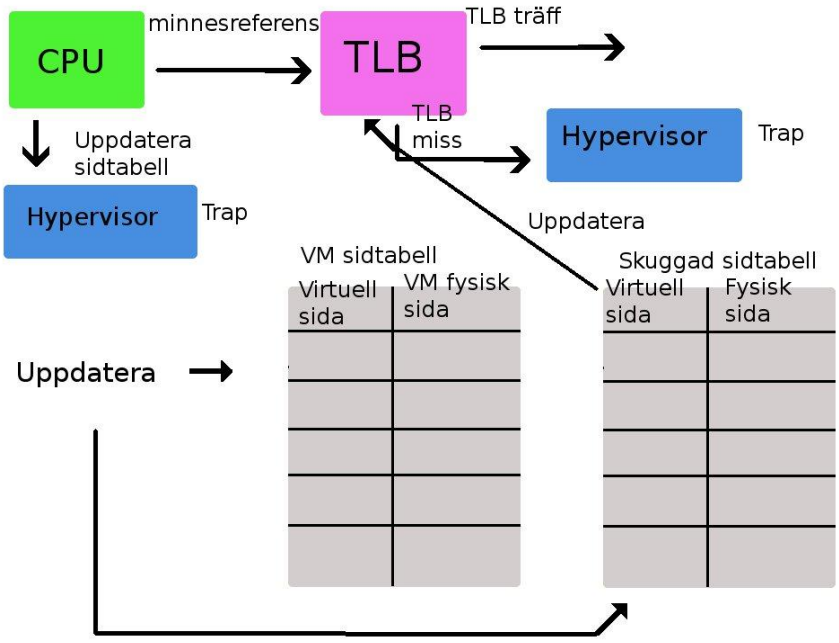
Hypervisorn lagrar dessutom en direkt översättning från processens virtuella minne till hårdvarans fysiska minne. Denna översättning lagras i en sidtabell som hålls gömd för den VM som exekverar. Denna sidtabell benämns skugg-sidtabell och har gett namn åt tekniken. En separat skugg-sidtabell lagras för varje VM och hypervisorn skiftar skugg-sidtabell vid context switch. Hypervisorn tvingar sedan processorn att använda den skuggade sidtabellen då adressöversättning ska utföras.
För att denna teknik ska fungera krävs att den skuggade sidtabellen hålls uppdaterad och konsistent mot sidtabellen hos VM. Det finns två olika tekniker för att uppnå detta.

Den första tekniken skrivskyddar VM:s sidtabell. Vilket generar sidfel vid ett uppdateringsförsök och således en trap till hypervisorn som kan kontrollera att den nya översättningen är säker för att sedan uppdatera skugg-sidtabellen.

Den andra tekniken tillåter VM att uppdatera sin egen sidtabell. När VM sedan vill ha åtkomst till denna minnesplats finns ingen översättning i skugg-sidtabellen som används av

processorn och ett sidfel uppstår vilket ger hypervisorn kontroll. När hypervisorn har getts kontrollen kan den uppdatera skugg-sidtabellen med den nya adressöversättningen. Båda dessa tekniker medför att skugg-sidtabellen hålls konsistent med VM:s sidtabell [20, 19, 9, 27, 26].

Sidtraverseringar tar tid och påverkar således prestandan för en VM. För att undvika sidtraverseringar används TLB som en form av cache för adressöversättningar. De översättningar som sparas i TLB är de översättningar som hämtas från skugg-sidtabellen vilket innebär att en snabbare och direkt översättning kan ske då VM vill ha åtkomst till en virtuell adress som tidigare översatts och nu finns lagrad i TLB [20].
Figur 7 visar hur uppdatering av skugg-sidtabell med TLB fungerar.



Figur 7, Minnesvirtualisering med skugg-sidtabell och TLB

Att virtualisera minnet i två lager på detta sätt medför att minnet mellan VM är isolerat då den enskilda VM ej har direkt tillgång till hårdvarans fysiska minne och ej kan manipulera den sidtabell som används för adressöversättning av processorn. Det finns dock nackdelar med denna teknik. Dels krävs det att man separat sparar en direktöversättning mellan processvirtualiserat minne och fysiskt minne hos hårdvaran och dels leder det även till ett stort antal sidfel vilket försämrar prestandan för VM [20]. Användningen av TLB medför en förbättrad prestanda men även en säkerhetsrisk, då detta kan leda till felaktiga adressöversättningar om det finns lagrad information i TLB. För att undvika denna säkerhetsrisk sker en rensning av TLB vid varje context switch. Både sidtraversering och

rensning av TLB medför prestandaförsämring på grund av virtualiseringen. För att undvika denna prestandaförsämring krävs hårdvarustöd för minnesvirtualisering.

3.4 AMD-V och Intel VT-x

AMD-V och Intel VT-x är två virtualiseringsstöd som är inbyggt i nyare AMD och Intel processorer. Dessa innehåller stöd för minnesvirtualisering som förbättrar prestandan och minskar virtualiseringskostnader.

3.4.1 Rapid Virtual Indexing och Extended Page Table

Stödet för minnesvirtualisering sker på samma sätt i både AMD-V och Intel VT-x. I AMD-V kallas det för Rapid Virtual Indexing och i Intel VT-x benämns det som Extended Page Table (EPT). I båda fall handlar det om nästlade sidtabeller vilket medför att en VM tillåts manipulera sidtabellen fritt utan insyn från hypervisorn [9, 31, 3].

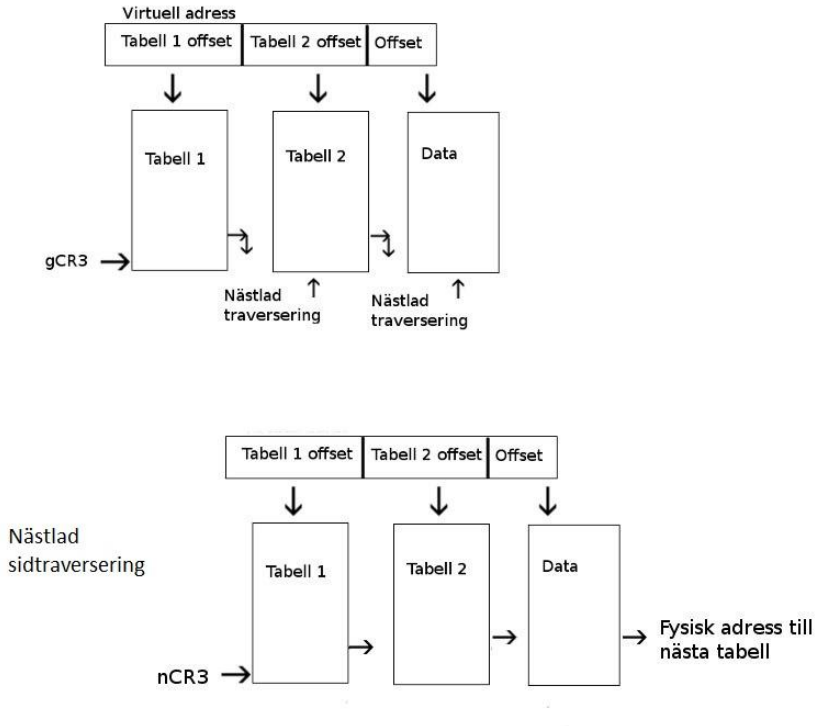
Vid användning av nästlade sidtabeller existerar som tidigare sidtabeller som översätter gästens virtuella minne till gästens fysiska minne. Skillnaden är att hypervisorn ej behöver lagra adressöversättning mellan fysisk adress i VM och fysisk adress hos hårdvaran, samt att ingen direktöversättning mellan processens virtuella minne och hårdvarans fysiska minne behöver lagras. Ytterligare en fördel är att inget avbrott till hypervisorn krävs när en VM vill utföra en minnesåtkomst.

Istället för avbrott vid minnesåtkomst sker en nästlad sidtraversering. Den nästlade sidtraverseringen kan delas in i dessa tre steg.

1. Välj ut tabellens förskjutningsbitar ur den virtuella adressen.
2. Leta upp posten i sidtabellen och erhåll gästens fysiska adress till nästa sidtabell
3. Utför en nästlad traversering för att översätta gästens fysiska adress till hårdvarans fysiska adress för att erhålla den fysiska adressen till nästa tabell i minnet.

Figur 8 visar en nästlad sidtraversering.

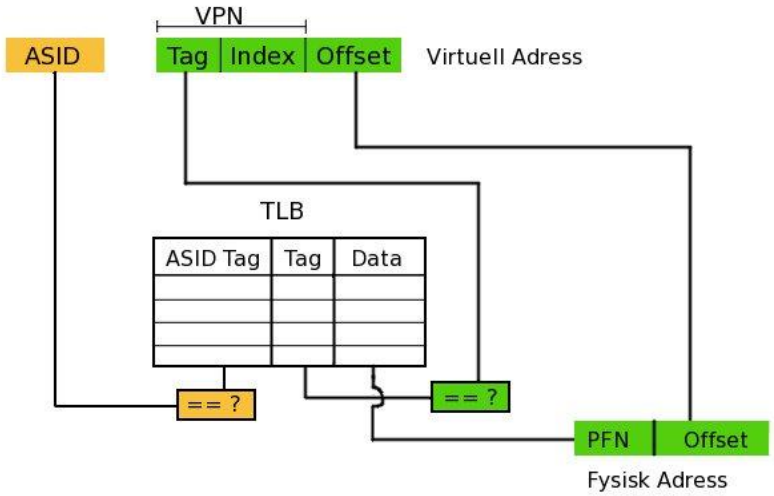
Fördelen med nästlade sidtabeller jämfört med skugg-sidtabeller är att det leder till mindre antal sidfel. Samt att adressöversättning sker i hårdvara och inte i mjukvara (hypervisor) vilket alltid är snabbare. Det sparar även minne då hypervisorn inte behöver lagra en skugg-sidtabell. Nackdelen är att sidtraverseringen blir längre [20].



Figur 8, Nästlad sidtraversering

3.4.2 ASID och VPID

Adress Space Identifier (ASID) och Virtual-processor Identifier (VPID) är AMD respektive Intels lösningar för att undvika att rensa TLB:n vid context switch.
AMD:s ASID är ett ID för VM:s som har översatt virtuella adressen till en fysisk adress. Hypervisorn tilldelar ett unikt ID till de VM:s som skall exekvera på systemet och är den enda komponent som har tillgång till detta ID [20]. I AMD-V representeras detta ID av en tag som varje post i TLB:n innehåller, vilket visas i figuren nedan. I och med användningen av ASID krävs ej rensning av TLB vid context switch. På så vis uppnås en effektivare adressöversättning för systemen. Den virtuella adressen består av Virtual Page Number (VPN) och ett offset. VPN i sin tur är uppdelad i en tag och ett index. Först jämför hypervisorn ASID:t för VM och sedan med en post i TLB:n. Stämmer även taggen från VPN överens med TLB-taggen erhålls en träff och TLB-datan ger ett sidplatsnummer. Tillsammans med den offset som existerar i den virtuella adressen erhålls den fysiska adressen [20], vilket visas i figur 9 nedan.
Intels VPID låter hypervisorn tilldela ett VPID skilt från 0 till de olika virtuella processerna som exekverar i varje enskild VM, vilket betyder att Intel skiljer sig en aning från AMD. I övrigt används VPID på samma sätt som ASID vid TLB uppslagning [9, 28].



Figur 9, TLB utökad med ASID

3.4.3 Prestandaförbättring med Intel VT-x och AMD-V

I ett forskningsexperiment utfört av VMWare testades prestandaförbättring vid användning av AMD-V och Intel VT-x. I experimentet implementerades en hypervisor som använde sig av AMD-V och en hypervisor som använde sig av VT-x. Därefter testade man dessa hypervisors på minnesintensiva applikationer exempelvis en SQL applikation. Övergripande gav användningen av AMD-V en prestandaförbättring för minnesvirtualisering på 42% och för Intel VT-x var samma siffra 48% [20, 3, 31].

4. Diskussion

Vi har kommit fram till att minne virtualiseras med hjälp av sidtabeller. För att uppnå virtualisering i flera lager så används en skugg-sidtabell där översättningen mellan processens virtuella minne och hårdvarans fysiska minne lagras. Denna skugg-sidtabell används sedan av hårdvaran för att utföra sidtraversering när en minnesadress ska översättas från virtuell adress till fysisk hårdvaruadress.

Då hypervisorn exekverar i en mer privilegierad ring än vad VM gör medför detta att en trap till hypervisorn sker då en VM uppdaterar sin sidtabell. Detta medför alltså att den enskilda VM ej har direkt tillgång till det fysiska minnet hos hårdvaran och således är minnet isolerat mellan VM. Nackdelen med denna teknik är emellertid att den leder till virtualiseringskostnader så som ett ökat antal sidfel samt att den kräver extra utrymme i minne då en skugg-sidtabell måste sparas för varje VM.

Med hårdvarustöd för minnesvirtualisering går det att eliminera hypervisorns inblandning i uppdateringen av sidtabellen. Således blir den enskilda VM tillåten att utföra uppdateringar till sidtabellen vilket leder till mindre antal sidfel och därmed en förbättrad prestanda.

Vi har i vår undersökning inte kunnat finna några specifika brister att peka på i den teknik som används för minnesvirtualisering av både Xen och VMWare. I dagsläget känner en VM inte till adressöversättningen till hårdvarans fysiska minne. Det finns dock inget som säger att en VM i framtiden inte kan utvecklas till att komma åt denna information.

Det uppsving som virtualisering erhållit under senare år med dess inblandning i molntjänster har lett till att processorutvecklare skapat virtualiseringsstöd i moderna processorer. AMD och Intel erbjuder i sina mer utvecklade processorer stöd för minnesvirtualisering som eliminerar användandet av skugg-sidtabell. Detta innebär att virtualiseringskostnader som stort antal sidfel och trap till hypervisor vid sidtabelluppdatering elimineras, vilket medför att hypervisorn blir än mer transparent och således bättre uppnår de designkrav som Popek och Goldberg satt upp för hypervisors. Då det fortfarande krävs en översättning i två nivåer finns alltjämt vissa virtualiseringskostnader kvar. Dessa virtualiseringskostnader är likväl avsevärt mindre än de virtualiseringskostnader som existerar vid användandet av skugg-sidtabell, då dessa kostnader nu har förflyttats från mjukvara till hårdvara.

Ett eventuellt framtida problem som vi ser är att en VM skulle utvecklas och till följd därav förmå att hitta andra VM:s ID och genom detta kunna hitta adressöversättningar i TLB. Vi har

dock inte utfört några egna tester vilket skulle kunna vara ett led till fortsatta studier inom ämnet.

5. Slutsats

De hypervisors som vi har undersökt har en korrekt minnesisolering mellan VM:s, vilket ytterliggare visar på att de följer de designkrav som sattes av Popek och Goldberg för hypervisors. Således har även de molntjänster som bygger på dessa hypervisors en korrekt minnesisolering vilket medför en god säkerhet för molntjänsterna.
Utvecklingen av hårdvara medför en prestandaförbättring för minnesvirtualisering vilket i framtiden kan medföra en prestandaökning hos de molntjänster som använder denna hårdvara.

Referenser

[1] Wikipedia: Hyper-V
URL: http://en.wikipedia.org/wiki/Hyper-V
besökt 2011-03-08

[2] ... och många faller efter, av Staffan Movin
URL: http://www.idg.se/2.1085/1.342399/-och-manga-foljer-efter
besökt 2011-03-04

[3] Performance Evaluation of Intel EPT Hardware Assist
Nikhil Bhatia
Revision: 20090330; Item: EN-000194-00
2008-2009 VMWare, Inc

[4] Molnet lockar allt fler trots osäkerhet, av Tommy Engfors
URL: http://www.idg.se/2.1085/1.287597/molnet-lockar-allt-fler-trots-osakerhet
besökt 2011-06-13

[5] Cloud Computing Security – making virtual machines cloud ready
Cloud Computing Security
Trend Micro White paper, 2010

[6] Server Virtualization Architecture and Implementation
Jeff Daniels
Crossroads, Volume 16 Issue 1, September 2009. Pages 8 – 12, ISSN: 1528-4972
ACM New York, NY, USA

[7] Architecture of Virtual Machines
R. P Goldberg
Proceedings of the workshop on virtual computer systems. Pages 74 – 112
ACM New York, NY, USA ©1973

[8] On the Relationship Between Virtual Machines and Emulators
Efrem G. Mallach
Proceedings of the workshop on virtual computer systems. Pages 117 – 126
ACM New York, NY, USA

[9] Secure Virtualization and Multicore Platforms State-of-the-Art report
Heradon Douglas och Christian Gehrmann
Swedish Institute of Computer Science

[10] Amazon Web Services: Overview of Security Processes
White paper: Amazon Web Services, 2008

[11] Xen Users' manual
The Xen Team
University of Cambridge, UK

[12] Produktsida för VMWare ESXi
URL: http://www.vmware.com/products/vsphere-hypervisor/index.html
besökt 2011-03-12

[13] Quick Migration with Hyper-V
White paper: Microsoft, 2008

[14] Understanding Full Virtualization, Paravirtualization, and Hardware Assist
White paper: VMWare, 2007

[15] New Approach to Virtualization is Lightweight.
Steven J. Vaughan-Nichols.
IEEE Computer, Volume 39, Issue 11, Pages 12 – 14, 2006.
IEEE Computer Society Press Los Alamitos, CA, USA

[16] Architectural Principles For Virtual Computer Systems (Sid 22-26)
Robert P. Goldberg
Harvard Univ Cambridge MA Div of Engineering and Applied Physics

[17] Xen and the Art of Virtualization
Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt och Andrew Warfield
SOSP '03 Proceedings of the Nineteenth ACM Symposium on Operating systems principles,
ISBN: 1-58113-757-5
ACM New York, NY, USA ©2003

[18] Understanding Full Virtualization, Paravirtualization, and Hardware Assist
Whitepaper VMWare

[19] Understanding Memory Resource Managment in VMWare ESX 4.1
White paper VMWare

[20] AMD-V™ Nested Paging
White paper AMD, 2008

[21] Dynamic Memory Control
URL: http://wiki.xensource.com/xenwiki/Dynamic_Memory_Control
besökt 2011-03-29

[22] Virtualization System for Computers that Use Adress Space Identifiers
Xiaoxin Chen, Alberto J. Munoz och Sahil Rihan
United State Patent, Patent No.: US 7,409,487 BI, Date of Patent: Aug. 5, 2008

[23] Virtual memory
URL: http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Memory/virtual.html
Besökt 2011-04-12

[24] Roos, Olle (1995). *Grundläggande datorteknik*. Lund: Studentlitteratur

[25] Brorsson, Mats (1999). *Datorsystem: program- och maskinvara*. Lund: Studentlitteratur

[26] Xen memory management

[27] Memory resource management in VMWare ESX server.
Carl A. Waldspurger.
In Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation, 2002

[28] Intel technology journal, Intel Virtualization technology
Volume 10, Issue 03
10 Augusti 2006
ISSN: 1535-864X

[29] A Virtual Machine Introspection Based Architecture for Intrusion Detection
Tal Garfinkel och Mendel Rosenblum
Computer Science Department, Stanford University

[30] Virtual Memory
Peter J. Denning
Journal, Volume 2, Issue 3, Sept. 1970, pages 153 – 189. ISSN: 0360-0300
ACM New York, NY, USA

[31] Performance Evaluation of AMD RVI Hardware Assist
Nikhil Bhatia
Revision: 20090311; Item: EN-000147-01
2008-2009 VMWare, Inc

[32] A comparison of software and hardware techniques for x86 virtualization
Keith Adams och Ole Agesen
ACM SIGPLAN Notices – proceedings of the 2006 ASPLOS Conference, Volume 41, Issue 11, November 2006, pages 2-13, ISSN: 0362-1340
ACM New York, NY, USA

Figurförteckning