Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Systém na testování výkonnosti virtualizační technologie superpočítače CMU

Bakalářská práce

Autor: Luboš Plch

Vedoucí práce: Ing. Jan Fesl

České Budějovice 2015

Zadání bakalářské práce:

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: Luboš Plch
.....
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika
.....

Katedra/ústav, kde bude práce vypracovávána:.UAI PRF JU
.....

Školitel: Ing. Jan Fesl
.....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF:
.....
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant: .....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: Systém na testování výkonnosti virtualizační technologie superpočítače CMU
.....

.....

Cíle práce:

Účelem zadávané práce je vytvoření výkonného jádra systému určeného pro testování virtualizační technologie univerzitního superpočítače CMU. Systém musí umožňovat distribuované spouštění virtuálních strojů v rámci gridové infrastruktury s využitím virtualizační technologie Hyper-V. Kvalitativními parametry testování budou zejména režie virtualizační technologie v závislosti na počtu spuštěných virtuálních počítačů, výkonnost různých typů distribuovaných souborových systémů pro přenost obrazů virtuálních počítačů a interní teplota hardwaru daného systému v závislosti na jeho zatížení. Implementace testovacího systému musí být realizována modulárně pomocí dynamicky načítaných knihoven s využitím volání jádra operačního systému (WIN API) popřípadě pomocí rozhraní POWER SHELL. Implementaci je nutné vytvořit v programovacím jazyce C++ nebo C#. Součástí této práce je i poskytnutí relevantních výstupů pro možnou vědeckou publikaci.

Základní doporučená literatura:

1. Miroslav Virius, Od C k C++, nakladatelství Koop, třetí vydání.
2. Přednášky předmětu UAI 630 – Distribuované a paralelní algoritmy
3. Aktuální či budoucí vědecké články a publikace dle pokynů vedoucího práce

Financování práce: .....

Vedoucí práce: .....podpis: .....

U externích vedoucích fakultní garant práce: .....podpis: .....

Garant oboru bak. studia, pokud je obor zajišťován jinou katedrou/ústavem, než ze které je školitel (nepožaduje se u oboru biologie): .....podpis: .....

Vedoucí katedry/ústavu, kde bude práce vypracovávána: .....podpis: .....

Případný souhlas vedoucího ústavu AV: .....podpis: .....

V Českých Budějovicích dne ..11.2. 2015.....Podpis studenta: .....

Bibliografické údaje
Plch Luboš, 2015: Systém na testování výkonnosti virtualizační technologie superpočítače CMU
[System for testing performance of virtualization technology supercomputer CMU. Bc. Thesis, in Czech] – 47 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Tato práce pojednává o systému, který dokáže na reálné fyzické infrastruktuře superpočítače vytvořit infrastrukturu virtuálních počítačů a tuto virtuální infrastrukturu následně vzdáleně řídit z jednoho centrálního bodu.

Abstract:

This bachelor thesis deals about a system, what is able to create the infrastructure of virtual computers, which are running on the HPC physical architecture. The created virtual infrastructure can be managed from one remote central node.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů..

Datum: 20. 4. 2015

Podpis

Poděkování
Chtěl bych poděkovat svému školiteli Ing Janu Feslovi za pomoc a rady při psaní této bakalářské práce.

Obsah

1. Úvod

Virtualizace je v posledních několika letech velmi často skloňovaným termínem a velice úzce souvisí s moderními termíny jako je Cloud, Infrastracture as a service, atd. Málo kdo ale ví, jak vlastně daná technologie funguje a jaké jsou její limity. Velkou neznámou je pak efektivita této technologie, která je označována jako malá, avšak není dostupný nástroj jak si toto tvrzení ověřit. Cílem této práce je seznámení se základy virtualizační technologie, která je dnes používána v nejmodernějších datacentrech a na základě těchto znalostí navrhnout a zprovoznit virtualizační technologii v rámci superpočítače CMU.

Na takto postaveném systému je poté nutno implementovat systém, který umožní měřit ztráty takovéto technologie oproti technologii fyzické. Jeho cílem je snadné ovládání tohoto jinak velice komplexního systému, které umožní uživateli otestovat superpočítač CMU a jeho efektivitu.

Virtualizační technologie je popsána spolu s testováním výkonnosti ve třetí kapitole této práce. Čtvrtá kapitola pak hovoří o návrhu celého systému a to od návrhu vlastního superpočítače přes jeho softwarové vybavení až po návrh implementace systému pro testování tohoto superpočítače. Vlastní implementace tohoto systému a potřebná nastavení na superpočítači jsou popsána v páté kapitole. A šestá kapitola hovoří o otestování tohoto systému a to včetně testů při reálném provozu.

2. Cíle Práce

Cílem této práce je navrhnout architekturu superpočítače CMU která bude vhodná pro běh několika desítek virtuálních počítačů a jejich následné používání přes vzdálenou správu daného systému. A to včetně výběru souborového systému pro ukládání virtuálních počítačů.

Dalším cílem je vysvětlit funkčnost vlastní virtualizační technologie, techniky které se pro virtualizaci používají a zhodnotit jejich výhody a nevýhody s ohledem na jednotlivé implementace těchto technik a následně vybrat vhodnou virtualizační techniku pro superpočítač CMU.

Hlavní cíl této práce je vytvořit software pro vytvoření virtuální infrastruktury na infrastruktuře fyzické a její následné řízení z jednoho bodu což umožní otestování takto vytvořené infrastruktury.

3. Teorie

Teoretická část Bakalářské práce se skládá z několika sekcí.

V části 3.1 se řeší obecné principy virtualizace a odlišnosti jednotlivých implementací z hlediska potřebných prostředků, nároků na systém, finančních nákladů a rozebírá možnosti automatizace u jednotlivých virtualizačních technologií, také řeší technologie Intel VT-X a AMD-V které jsou pro některé hypervizory nezbytné a pro jiné volitelné.

Část 3.2 rozebírá možnosti měření výsledků pro vybrané hypervizory

3.1. Virtualizace

3.1.1. Historie

Virtualizace byla poprvé použita už v roce 1967, kdy IBM představilo svůj hypervizor CP-40/CMS [1] což byl hypervizor prvního typu (mluvit o CP-40 jako o hypervizoru je sporné, je ovšem označován za první systém tohoto typu). Tedy bez asistence operačního systému. Později (1968) se stal základem revoluční rodiny hypervizorů od firmy IBM CP/CMS (*Control Program/Cambridge Monitor System*). Ovšem jeho počátky můžeme vysledovat až k systému CTSS (*Compatible Time-Sharing System*) Což byl jeden z prvních multitaskingů který pracoval na principu časového přepínání. Vyvinutém a přestaveném na MIT v roce 1961.

CP/CMS byl původně vyvinut jako systém pro podporu více přihlášených uživatelů, tím, že spouštěl několik instancí operačních systémů, ke kterým se uživatelé hlásili. Předpověděl tak moderní způsob plné virtualizace, protože používal 2 části pro běh

- CP (*control program*) byl vlastní „hypervizor“ tedy systém který řídil prostředky fyzického systém. Nebyl jako hypervizor označován (toto označení vzniklo až později (v roce 1972)) kdy IBM tento systém přejmenovalo na VM.
- CMS (*Cambridge monitor system*) byl vlastní operační systém, jehož jednotlivé instance běželi na CP a zajišťoval připojení více uživatelů zároveň do jednoho fyzického počítače.

Z CP/CMS byly později odvozeny další systémy a bylo vytvořeno několik nových schémat jak virtualizovat. Ty nejpodstatnější budou v práci popsány.

3.1.2. Princip

Virtualizace je označení postupů, které souží v informatice pro spouštění několika na sobě více či méně nezávislých operačních systémů na jednom fyzickém počítači. Toho může být dosaženo několika možnými postupy:

- **Plná (nativní) virtualizace**. Virtuální stroj pracuje na stejném typu hardware jako stroj fyzický. Toto umožnuje spouštět virtuální stroj bez emulace procesoru pouze se softwarovým oddělením od stroje fyzického. Výhodou tohoto systému je vysoká rychlost virtuálního procesoru, ovšem za cenu relativně velkého overheadu na řízení střídání systémů na procesoru i na většině periferií. Dnes se jedná o nejpoužívanější způsob virtualizace na x86 architektuře, kterou podporují systémy jako XEN, KVM, Hyper-V, VM-Vare a v neposlední řadě Oracle virtual box. Je použit hypervizor [2]. Přináší velkou výhodu oddělení virtuálních počítačů, kde není třeba upravovat hostovaný, ani hostitelský operační systém, tudíž by nemělo dojít k potížím s kompatibilitou operačních systémů.

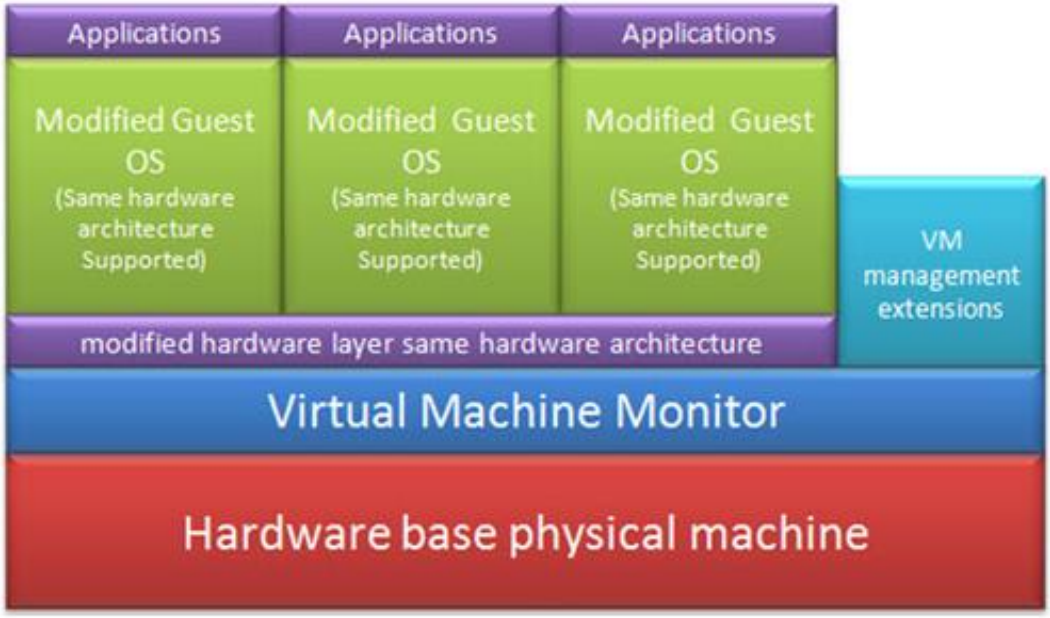


Schéma Plné Virtualizace [3]

- **Paravirtualizace** [4]. Jde o systém, kdy hostitelský operační systém musí být upravený tak, aby umožnoval sdílení svých prostředků mezi několik oddělených virtuálních počítačů. Tato metoda je výhodná, pokud požadujeme malý overhead (režii kterou spotřebuje virtualizační technologie oproti fyzickému stroji). Nevýhodou je, že hostitelský i virtualizovaný operační systém musí být upraveny pro funkci paravirtualizace, což je problém především u starších nebo méně rozšířených operačních systémů (Linux do verze 2. 6. 37 [4])

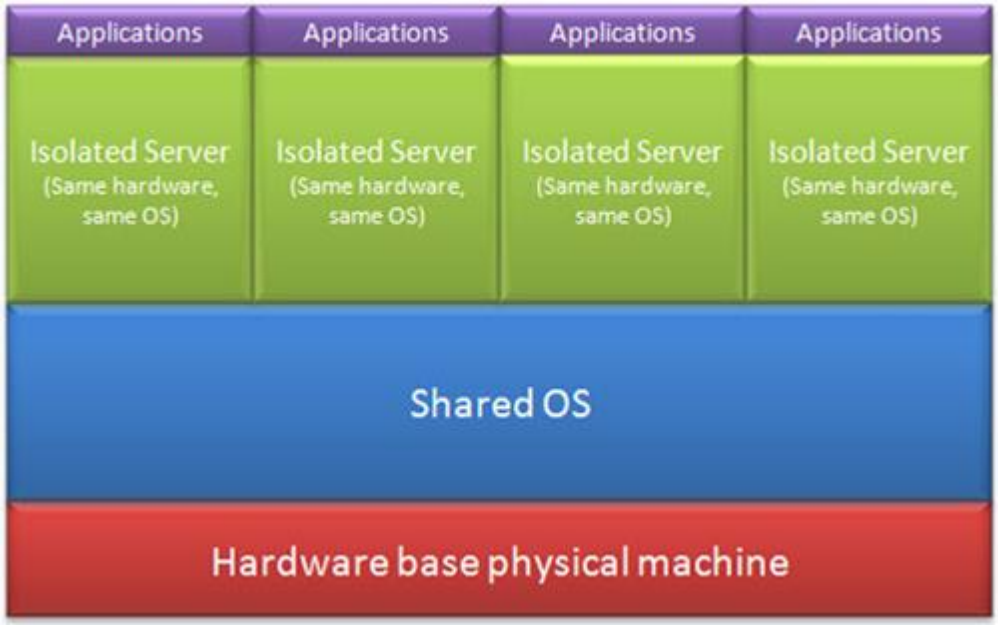


Schéma Paravirtualizace [3]

- **Aplikační virtualizace** [5]. Slouží pro běh aplikací, které nejsou závislé na operačním systému. Jde například o JVM (*Java Virtual Machine*) která slouží pro multiplatformnost Javy, kdy program napsaný a přeložený v Javě můžeme spustit na libovolném operačním systému, pro nějž je implementována JVM. Nejedná se o virtualizaci v klasickém slova smyslu, ale o jakési API, které překládá příkazy Javy do jazyku operačního systému, na kterém aktuálně běží.

- **Emulace (simulace).** Jde o simulování celého hardware pomocí software. Tento postup je velice pomalý a hodí se zejména pro testování aplikaci na jiné procesorové architektuře, než kterou disponuje použitý počítač (z pravidla Power Pc, arm, …). Příkladem je i Microsoft Virtual PC (v režimu pro Power PC) což je nástroj pro spouštění software napsaného speciálně pro starší verze Windows než

kterou disponuje aktuální počítač (tento nástroj je lépe znám jako režim kompatibility). Dalšími příklady jsou například Bochs, Qemu,…

Tyto postupy jsou dnes nejpoužívanější, ovšem ne jediné metody pro virtualizaci. V následující práci se vzhledem k rozšířenosti a kompatibilitě s jakýmkoliv hostovaným operačním systémem budu zabývat pouze plnou virtualizací.

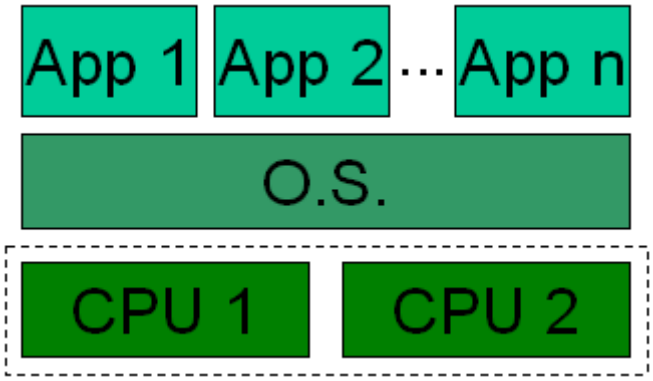
3.1.3. Plná virtualizace

Plná virtualizace je dnes nejpoužívanější metodou pro virtualizaci celého operačního systému a to především díky vysoce výkonným procesorům dnešní doby, které jsou najednou schopny obsloužit desítky až stovky klientů a díky dobré optimalizaci, kterou přinesly zejména technologie INTEL VT-x a AMD-V. Popřípadě technologiím INTEL VT-i, VT-d, VT-c které jsou dnes pod různými názvy implementovány v procesorech od obou hlavních výrobců procesorů architektury x86.

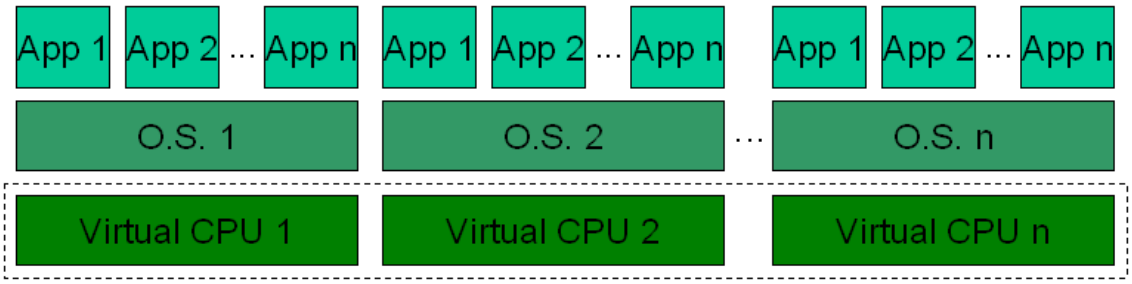
INTEL VT-x (Intel Virtualization Technology for IA-32 and Intel 64 Processors)

Tato technologie rozšiřuje instrukční sadu procesoru o 10 nových instrukcí: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, WMLAUNCH, VMRESUME,VMXOFF a VMXON. Které slouží pro zrychlení vykonávání instrukcí virtuálních počítačů oproti klasické virtualizaci na bázi aplikace.

Zrychlení je umožněno tím, že je se spouští instrukce virtuálního počítače přímo na fyzickém procesoru. Tím odpadá overhead hostitelského operačního systému, který nemusí s touto technologií instrukce virtuálního počítače překládat.

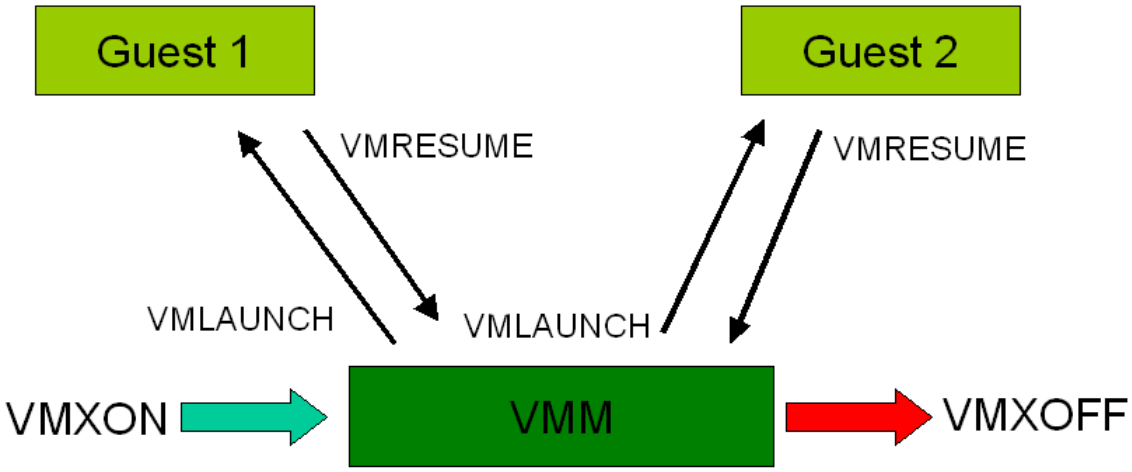


Běh klasického operačního systému běžícího na dvou jádrovém procesoru. Virtuální počítač by zde běžel jako jedna z aplikací (APP1 – APP n) [6]



Běh několika operačních systémů (jeden fyzický a n-1 virtuálních) na jednom fyzickém procesoru s technologií Intel VT-X [6]

Toto je možné díky spouštění virtualizačního módu procesoru kdy se střídají virtuální počítače jako by to byly počítače fyzické měněné na procesoru v čase. Pro vstup do režimu virtualizace slouží instrukce VMXON a pro vystoupení z něj VMXOFF. Pokud je procesor v režimu virtualizace tak se jednotlivé virtuální počítače na něm střídají spolu s hostitelským operačním systémem pomocí instrukcí VMLAUNCH která spouštějí virtuální počítač a VMRESUME které dává ostatním počítačům vědět, že právě prováděný počítač byl obsloužen.



Průběh obsluhy virtuálních počítačů s technologií Intel VT-X. Guest1 a Guest2 jsou virtuální operační systémy. [6]

Novější procesory obsahují ještě doplněk EPT (Extended Page Tables) který umožnuje každému počítači mít vlastní tabulku adres instrukcí. Díky tomu není třeba aby VMM střídal tabulky instrukcí na procesoru při každém střídaní počítačů, což značně redukovalo výkon.

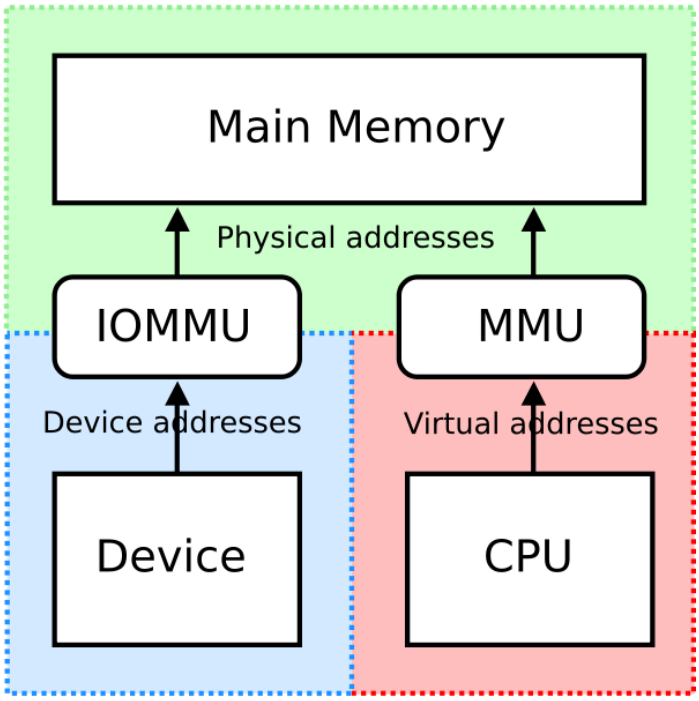
Intel VT-i (Intel Virtualization Technology for Itanium Processors)

Jedná se o obdobu technologie VT-x s tím rozdílem že běží na Převážně serverových procesorech Itanium od Intelu, ovšem dnes je, stejně jako tyto procesory, převážně na ústupu ve prospěch vysoce výkonné architektury x86 (Itanium zde bylo nahrazeno procesory Intel Xeon) a architekturou ARM na poli vysoce úsporných serverů, které se dnes začínají vyrábět a hromadně nasazovat díky spotřebě, která dosahuje cca 10% spotřeby řady x86

Intel VT-d (Intel Virtualization Technology for Directed I/O)

Tato technologie umožnuje hostovanému operačnímu systému přistupovat přímo k PCI zařízením, tím se stane jeho dedikovaným nástrojem a může ho používat stejně jako by byl fyzický operační systém (což je velký výhoda především pro grafické a síťové karty kde jde především o rychlost a každý overhead nás znatelně omezuje) Funkci zajištuje Intelem navržená jednotka IOMMU (*Input/output Memory Management Unit*) což je typ MMU (*memory management unit*) která překládá virtuální adresy viditelné procesorem na fyzické adresy uložené v paměti RAM s tím rozdílem, že nepřekládá adresy viditelné

procesorem, ale adresy viditelné daným zařízením. Tím pádem se dá říct že IOMMU je MMU která neslouží procesoru, ale jednotlivému zařízení a rozšiřuje adresovatelnou paměť počítače o jeho interní paměť



Porovnání IOMMU a MMU [7]

Ovšem jakmile je jednou zapnuto VT-d tak ztrácíme možnost live-migrace virtuálního stroje.

Předpoklady pro zpuštění VT-d jsou následující:

- Chip set musí technologii VT-d podporovat (zpravidla se jedná o chip sety od Intelu)
- Podpora Musí být implementována také výrobcem základní desky do BIOSu
- Podporován musí být také procesor (Intel Xeon 34xx,55xx a vyšší. Nebo některý z podporovaných procesorů řady CORE)

Pro použití technologie VT-d je také potřeba jí zapnout, dělá se to v BIOSu základní desky. Často toto bývá spojeno v jeden přepínač pro aktivaci celé rodiny Intel VT.

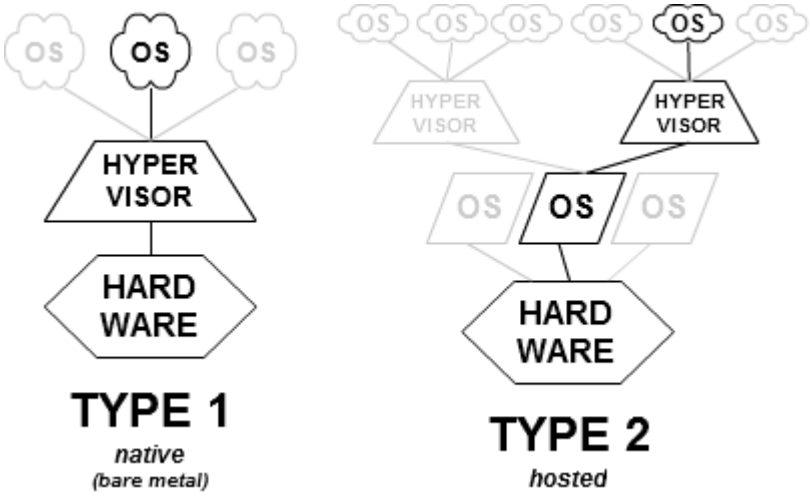
Všechny tyto techniky dnes urychlují plnou virtualizaci a rozdíly fyzických oproti virtuálním počítačům jsou podle výrobců téměř zanedbatelné, pokud jsou výrobci virtualizačního softwaru správně používány, tento software se nazývá hypervizor.

Hypervizor (VMM (Virtual Machine Monitor))

Je to hlavní součást plné virtualizace, řídí přístup virtuálních strojů k hardwaru a odděluje je od sebe navzájem i od fyzického hardwaru. Jeho implementace může být dvojí [8]:

Typ 1 (nativní): jedná se o hypervizor který běží přímo na hardwaru hostitele. Virtuální operační systémy běží přímo na hypervizoru, který měří jejich používání a řídí jejich přístup k hardwaru. První hypervizory (CP/CMS) byly právě tohoto typu a moderní hypervizory se k tomuto přístupu opět kloní (například Hyper-V, VMware, …)

Typ 2 (hostovaný): tento typ hypervizoru běží na klasickém operační systému, jako proces. Tudíž virtuální operační systém běží jako proces na hypervizoru, který běží jako proces na klasickém operačním systému. Tím je dána velká výkonová ztráta této konstrukce, od které se dnes již upouští.



Porovnání dvou základních typů hypervizorů [7]

Z hlediska rychlosti je lepší hypervizor prvního typu, ovšem má výraznou bezpečnostní slabinu: pokud běží hypervizor přímo na hardwaru tak teoretický vir který by se do něj dostal, nemůže být postihnut antivirem virtualizovaného systému, jelikož tento systém při správně funkčnosti neví, že je virtualizován. A hypervizor jelikož neobsahuje možnost instalace aplikací nemůže antivir obsahovat. Tudíž tento vir by měl plný přístup k virtuálním počítačům přes jejich virtuální disky. Naštěstí se tento vir reálně ještě nevyskytl, obrana proti němu by však byla náročná ne-li nemožná.

Na Trhu je dnes několik hypervizorů obou typů. Ty nejpoužívanější jsou:

XEN

Xen je hypervizor založený na Linuxu a je to dnes nejpoužívanější Open-Source hypervizor. Byl vyvinut na univerzitě v Cambridge Ianem Prattem Který v roce 2003 založil společnost XenSource, Inc, Která převzala vývoj a systém nadále udržuje open-Source pod licencí GPL. Vyvíjena byla ale i placená verze Xenu která byla obohacena o některé možnosti, rozšířenou správu a podporu. V roce 2007 Firmu koupila společnost Cirtix a vznikla placená verze Xenu Citrix-Xen. Systém je nadále vyvíjen jako open-Source a je „zcela zdarma“ placené jsou přidávané produkty které rozšiřují funkčnost samotného hypervizoru a taktéž podpora.

Xen funguje tak, že se spustí jako první přímo ze zavaděče a na něm se spustí vlastní operační systémy (takže na něm neběží fyzický operační systém, ale pouze systémy virtuální, kde se První jmenuje DOM0 a je považován za fyzický, protože má od Xenu přímý přístup k hardwaru a spravuje ho. Tento systém musí být vždy založený na Linuxu a jeho jádro musí být pro tento účel značně upraveno. Další virtuální počítače, které se spouštějí (uživatelské domény) se nazývají DOMu. Již mohou být i ne-Unixové systémy ale pro ty musí procesor podporovat technologii VT-x nebo AMD-v.

Do linuxového jádra nebyl Xen dlouho přidán z důvodu velkých zásahů a bylo vyžadováno přidat ručně podporu a upravit tak jádro. To bylo částečně vyřešeno v roce 2011, kdy byla část Xenu přidána do linuxového jádra (část starající se o implementaci Dom0) a podpora v rámci distribucí se tak Linuxu značně rozšířila.

Z hlediska automatizace je Xen vybaven Api které obsahuje vše potřebné pro jeho automatické ovládání lokálně i vzdáleně a to především díky použitému linuxovému jádru.

VMware ESX server

Jedná se dnes o jeden z nejpoužívanějších profesionálních (*enterprise*) virtualizačních nástrojů, ale kvůli několika stěžejním problémům jeho konstrukce, je pomalu nahrazován především systémem Hyper-V od firmy Microsoft.

ESX server je jedním ze dvou produktů firmy VMware. Jedná se o placený hypervizor, který funguje pouze na serverech, které jsou pro něj certifikované a mají příslušné

ovladače (většinou jsou řádově dražší než běžné, stejně výkonné servery) což je jedna z jeho hlavních nevýhod.

Jedná se o hypervizor prvního typu, který běží přímo na severu bez mezilehlého operačního systému, ovšem pro svůj běh prý využívá jádro Linuxu. Kvůli tomuto je VMware cílem několika žalob za zneužití tohoto software bez splnění podmínek licence pod kterým je zveřejněn a není jasné jak se tento spor bude vyvíjet nadále.

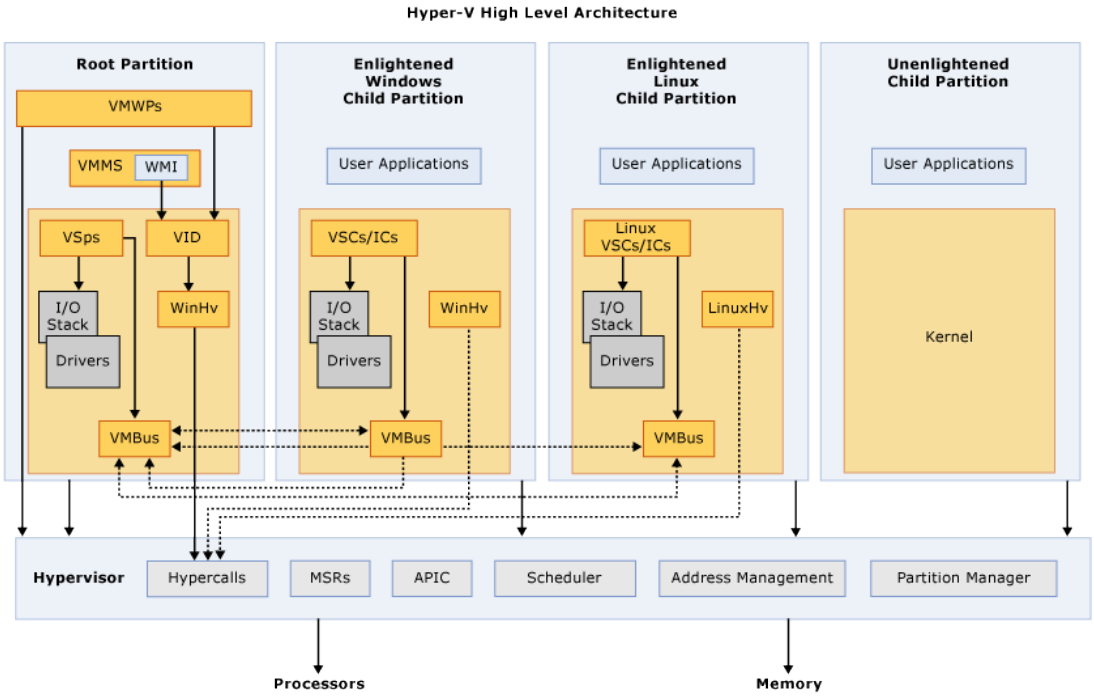
Jeho vnitřní struktura není známá.

Microsoft Hyper-V

Nejrychleji rostoucí virtualizační platformu má na svědomí firma Microsoft. Hyper-v se poprvé objevil ve Windows serveru 2008, jako jeho součást za kterou již nejsou vybírány další poplatky. Značně vylepšen byl s příchodem Windows serveru 2012.

Oproti VMware má tu výhodu, že podporuje libovolný hardware který je podporován i samotným Windows. Tudíž neexistuje vazba na předražený licencovaný hardware. Navíc od Windows 8 již není jen doménou serverů, ale je zdarma přibalen ve verzích pro a enterprise desktopového operačního systému Windows.

Z hlediska funkčnosti se jedná o hypervizor prvního typu. Tedy leží pod operačním systémem a tím fyzický (*root*) systém staví na úroveň virtualizovaných operačních systémů. K oddělení a „zaručení funkčnosti“ fyzického operačního systému používá nastavení root parametru architektury VT-x tomuto systému. Což zaručuje že fyzický systém se v případě přetížení virtuálními systémy nezhroutí, ovšem za cenu že hyper-v nelze provozovat na počítačích které tuto technologii (nebo její obdobu AMD-v) nepodporují. Hypervizor plně využívá VT-x a ostatní technologie VT, jejichž funkce implementuje jako takzvané Hypercalls které tento přístup k hardwaru zprostředkovávají



Architektura Platformy Microsoft hyper-V [3]

Z hlediska automatizace má hyper-v výhodu, že je implementované přímo v operačním systému Windows. Toto nabízí hned několik možností automatizace a správy:

- **Win32 API**. Toto api je velice komplexní a slouží k programovatelnému řízení celého operačního systému Windows. Jedním z jeho modulů je i modul pro řízení Hyper-v
- **PowerShell** obsahuje veškerá možnosti pro řízení hyper-v a je vnitřně jen implementací win32 api. Tudíž je dostatečně rychlý.
- **SystemCenter** jedná se o placenou nadstavbu pro Windows server která slouží k jeho automatizaci a vzdálenému řízení. Ve své podstatě nepřináší nic, co by jiné metody ovládání neuměli, jen toto usnadňuje.

Oracle Virtual Box

Jedná se o software který má za sebou bouřlivou historii. V roce 2007 ho představila společnost Innotek pod licencí GNU, v roce 2008 tuto společnost koupila společnost Sun a s vývojem pokračovala pod licencí GNU ovšem některé časti (doplňky) již licencovala jako closed-source a vydávala je jen jako předkompilované instalační soubory. V roce

2010 proběhl přesun pod společnost Oracle a VirtualBox se definitivně rozdělil na 2 části. Jednu Open source která obsahuje vlastní jádro systému, ovšem bez jakýchkoliv rozšíření (například podporuje pouze USB 1.1) a na část která se poskytuje jako vlastní instalace bez zdrojových kódů a má podporu i v novějších technologiích. Přičemž veškeré předkompilované instalační soubory již obsahují rozšířenou verzi a verzi základní je třeba si zkompilovat.

Velkou výhodou je podpora většiny moderních operačních systémů (Microsoft Windows, Linux, OSX) a podpora obou typů hypervizorů (i typ 1 i typ 2) a jejich přepínání pouze pomocí nastavení vlastního virtualizačního softwaru. Pod platformou Windows je však velká nevýhoda nemožnost běhu společně s Hyper-V jelikož oba se snaží zaregistrovat u hostitelského systému jako hlavní hypervizor a to není principiálně možné. Řešením tohoto je pouze reinstalace hypervizoru a restart hostitelského počítače při každém přepnutí hypervizoru. Další nevýhodou je neexistence serverové varianty.

Jedinou možností automatizace je příkazová řádka kam má VirtualBox přes interní aplikaci vyvedeno veškeré ovládání.

3.2. Měření (Benchmarking)

Měření výkonosti počítače se skládá z měření jednotlivých dílčích úloh, které vytíží vždy jednotlivé části počítače. Takto Otestovaný počítač nám poskytne řadu čísel, podle nichž jsme schopni říci jak je výkonný v jednotlivých částech jeho práce. Testovat můžeme mimo jiné následující části počítače

- Procesor.
- Grafická karta
- Paměť RAM
- Síť
- I/O operace kam spadá hlavně harddisk

Většina z těchto testů se skládá ještě z několika dílčích testů, které jsou rozdílné dle implementace.

3.2.1. Měření výkonnosti Virtuálního Počítače

U virtuálního počítače je situace částečně rozdílná, nejzajímavější pro nás není výkon virtuálního počítače jako takový, protože si ho můžeme snadno nastavit v hypervizoru, ale především poměr výkonů virtuálního a fyzického stroje. Kde jde o výkon který je režií vlastní virtualizace a pro nás je to výkon ztrátový.

Hlavním cílem tedy není říci jedno číslo, která zastupuje výkon virtualizovaného počítače, ale především poměr výkonu vynaloženého hostitelským systémem a výkonu který se skutečně dostal k uživatelské aplikaci běžící ve virtuálním počítači, popřípadě vývoj této ztráty v závislosti na počtu virtuálních počítačů a jejich využití.

4. Návrh systému

Návrh vlastního systému pro testování byl velice závislí na použitém hardwaru a to jednak z hlediska rozsahu testování, tak z hlediska možné podpory/nepodpory software od firmy WMware.

4.1. Popis použitého hardware

Použitá infrastruktura obsahuje 3 typy počítačů.

4.1.1. NAS

Jedná se o počítač, navržený pro hromadné ukládání virtuálních počítačů a jejich rychlou distribuci po univerzitní síti na potřebná místa. Z tohoto důvodu je opatřen 10GB sítovou kartou určenou pro přenos dat, která je připojena na stejně rychlou univerzitní páteř přes centrální switch určený přímo pro CMU (disponuje čtyřmi 10GB porty z nichž jeden je univerzitní páteř a 24 1GB porty). Dále jednu 1GB sítovou kartu určenou pro správu a řízení NASu zapojenou do téhož switche. Diskový prostor obstarává 6 pevných disků každý o kapacitě 3TB od firmy Seagate zapojené pro ochranu dat do RAID 5. Pro obhospodařování tohoto diskového prostoru je počítač vybaven čtyř jádrovým procesorem i5 od Intelu a 32GB RAM. V systému je označován jako CMU-NAS a je k dispozici jeden jeho exemplář.

4.1.2. CORE

Core je cca 8 let starý univerzitní server, který před tím, než byl přidělen CMU sloužil jako jeden z provozních serverů pro přírodovědeckou fakultu. Po jeho vyřazení byl shledán vhodným pro úkol řízení CMU. Je vybaven čtyř jádrovým procesorem Intel exon e5405 z roku 2007. a 6GB RAM. Jeho výhodou jsou dva 300GB pevné disky v zapojení RAID 1(zrcadlení) tudíž je bezpečný pro ukládání dat. V systému je označován jako CMU-CORE a je k dispozici jeden jeho exemplář.

4.1.3. NODE

Dvojice uzlů NODE (CMU-NODE1 a CMU-NODE2) se stará o vlastní výkon superpočítače CMU. Každý má dva dvaceti jádrové procesory E5-2660 v2 od Intelu. Doplněné o 128GB RAM. K univerzitní síti jsou připojeny jednou 10GB sítovou kartou pro přenos dat a jednou 1GB sítovou kartou pro ovládání uzlů.

4.2. Síťová topologie CMU

Celý superpočítač má topologii dvojité hvězdy, přičemž jedna hvězda se skládá z 10GB technologie a jsou do ní zapojeny počítače NODE a NAS, druhá hvězda je postavená na technologii 1GB a jsou na ní počítače NODE, NAS a CORE. Obě tyto hvězdy mají společný výstup do univerzitní sítě a tím i do internetu přes technologii 10GB.

Z této topologie vyplívá, že každý z počítačů NODE a NAS má 2 IP adresy, kde jedna slouží pro správu a řízení počítače (1GB) a druhá pro přenos virtuálních počítačů (10GB).

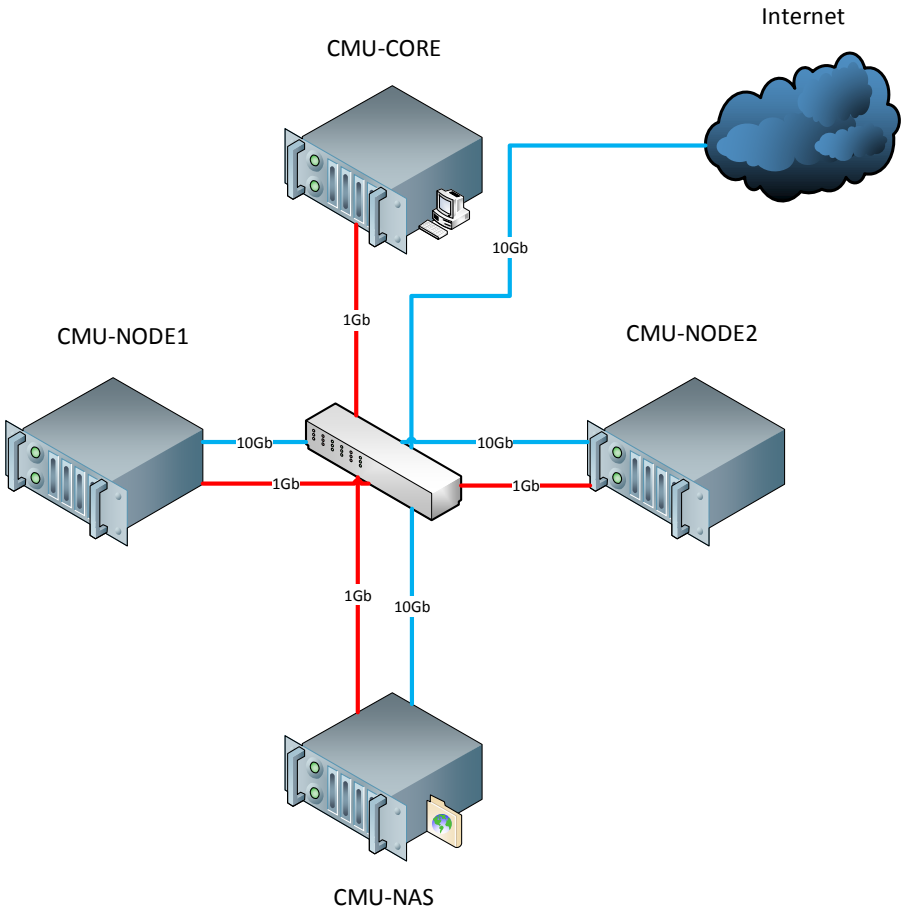


Schéma zapojení superpočítače CMU

4.3. Popis použitého Software

Z důvodu rozdílnosti výše jmenovaných hypervizorů a jejich rozdílných nároků na operační systém běžící na fyzickém počítači bylo nutné vybrat softwarovou výbavu. Vše se odvíjí od výběru hypervizoru který je pro systém klíčový.

4.3.1. Výběr hypervizoru

Jako hypervizor bylo vybráno Microsoft HYPER-V a to z několika důvodů. Hlavním důvodem bylo, že daný hypervizor je na špičkové úrovni a je masivně používaný v nejnovějších data-centrech. Dalším z důvodů byla cena daného hypervizoru, který je zdarma k Windows serveru 2012 R2 na která je akademická licence. Tudíž jediné náklady byly na vlastní Hardware, Na rozdíl od například VMware který žádné akademické licence neposkytuje a navíce hardware pro něj by byl řádově dražší než současné řešení od firmy SuperMicro.

4.3.2. Operační systém

Vzhledem k použitému hypervizoru je volba operačního systému jediná možná. Tudíž byl na všechny servery nainstalován Windows Server 2012 R2 Data-center Edition. Který byl v rámci akademické licence poskytnut Zdarma. Tento systém byl nainstalován na všechny počítače, ze kterých se superpočítač CMU skládá.

4.4. Výběr souborového systému

Výběr souborového systému se vzhledem k síťově oddělenému úložišti CMU-NAS skládala ze dvou částí. Konkrétně z výběru vlastního souborového systému a z výběru protokolu pro jeho dostupnost po síti. Vzhledem k použití operačního systému Windows v celé topologii superpočítače byl na obě tyto části požadavek, aby byly taktéž implementovány firmou Microsoft, především z důvodu jednotné autentizace vůči všem částem superpočítače a možnosti centrálního řízení.

4.4.1. Souborový systém

Firma Microsoft podporuje 2 základní souborové systémy.

- Prvním je NTFS (*New Technology File System*) jehož první verze byla vydána v roce 1993 a jedná se o žurnálový souborový systém, který je především určen pro osobní počítače, protože nemá podporu pokročilých serverových funkcí, jako RAID které musí být přidávány zvlášť a taktéž je limitován maximální velikostí svazku i souboru.
- Druhým souborovým systémem je REFS (*Resilient File System*) který byl představen v roce 2012. Jedná se o serverový souborový systém, který byl přidán pouze do Windows serveru 2012 a novějších. Tento souborový systém podporuje pokročilé serverové funkce jako RAID a bezpečnost souborového systému. Maximální velikost svazku ani souboru zde nejsou limitem. Hlavní nevýhodou je nemožnost obnovit smazané soubory jako to je u NTFS protože tento mladý souborový systém ještě nebyl dostatečně prozkoumán.

Z tohoto srovnání vyšel vítězně souborový systém REFS který by podle oficiálních srovnání měl mít i výhodu rychlejšího přístupu k datům z důvodu lepšího systému uložení dat.

4.4.2. Protokol pro síťový přístup

Jako protokol pro přístup k souborovému systému byl vybrán protokol SMB (*Server Message Block*). Což je dnes nejpoužívanější protokol pro sdílený přístup k souborům, tiskárnám, … je znám také jako CIFS (*Common internet File System*) ovšem jeho nejznámější Implementace se jmenuje SAMBA což je implementace tohoto systému pro operační systém Linux.

Výhodou SMB je používání autentizace pro přístup ke sdílenému médiu. V rámci systému Microsoft Windows jsou k tomuto používány autentizační údaje uživatelského účtu který má k tomuto úložišti právo přistupovat popřípadě další práva. Výhodou tohoto je že pokud má uživatel provádějící akci na souborovém systému práva na provedení dané akce tak již se při provádění této akce nemusí znovu autentizovat. Pokud použijeme Doménový kontrolér tak se nemusí autentizovat dokonce ani při vzdáleném přístupu, kdy je autentizace vyřešena přes doménový kontrolor automaticky

Přístup protokol SMB umožnuje buďto přes sítovou cestu (IP nebo DNS záznam) (např: \\cmu-core.prf.jcu.cz.cz\data) nebo přes symbolické jméno serveru (např: \\CMU-CORE\Data).

4.5. Návrh Software pro testování

Pro testovací software bylo úkolem vytvořit na uzlech virtuální infrastrukturu, jakou si definuje uživatel a posléze na této infrastruktuře umožnit spuštění vlastních testů a jejich výsledky vrátit zpátky na řídící server kde budou uloženy. Tato Infrastruktura se může vyznačovat rozdíly mezi jednotlivými operačními systémy, kde se může vyskytovat 32 a 64 bitová verze systému Windows 8.1 a Linux Debian 7 také v obou verzích. Těchto počítačů může být vytvořeno různé množství s různými konfiguracemi a to především s různou pamětí RAM a různým počtem virtuálních procesorů. Takováto infrastruktura musí být vytvořena vždy do stejného bodu kvůli platnosti testů. Tudíž po skončení jednotlivého testu musí být zase celá odstraněna a pro další test vygenerována znovu. Z tohoto plyne požadavek na vysokou rychlost tvorby a mazání virtuálních počítačů.

Dalším úkolem testovacího software je připojit se k takto vytvořeným virtuálním počítačům a spustit na nich požadované testy. Zde je nejnáročnějším úkolem vytvořit vlastní infrastrukturu, jelikož z podstaty technologie nejsme schopni z hostujícího

operačního systému IP adresy jednotlivých virtuálních počítačů. Na vytvořené topologii se poté bude komunikovat se serverem, který bude vlastní testy spouštět a shromažďovat jejich výsledky, které budou dále předány ke statistickému zpracování.

Z důvodu tohoto Dvojího úkolu testovacího programu bylo rozhodnuto vytvořit ne jeden, ale dva programy kde jeden z nich bude vytvářet a mazat vlastní virtuální počítače a druhý bude virtuálním počítačů na dálku spouštět testy.

4.6. Program pro vytváření virtuální infrastruktury

Základním požadavkem na tento program byla rychlost vytvoření infrastruktury, kde každý virtuální počítač se musí po každém vytvoření nacházet vždy v přesně stejném stavu. Proto bylo jako hlavní programovací jazyk vybrán C++ které díky své přenositelnosti přináší dostatečnou rychlost aplikace a to při zachování výhod objektového přístupu který je pro programátora pohodlnější při tvorbě vlastní aplikace. Jako další výhodu tohoto jazyka lze uvést výbornou správu více vláknových aplikací a jejich jednoduchou tvorbu což dodává takto napsané aplikaci další stupeň rychlosti. C++ ale neobsahuje na rozdíl od jazyků rodiny DOT.NET vlastní zprávu technologie HYPER-V a je tudíž potřeba přistoupit k automatizaci pomocí prostředků které nabízí vlastní Hyper-V. To poskytuje 2 základní možnosti automatizace.

- **Win32Api**

Jde o dlouho vyvíjený nástroj pro programovou správu systémů od firmy Microsoft a jiných produktů této firmy, které se do systému začleňují. Tudíž je možnost toto api použít i k řízení Hyper-v. jeho výhodou je snadné vyvolání funkcí api v jazyce c++ ovšem jeho hlavní nevýhodou je obrovské rozsah tohoto api a náročné spojení funkcí pro vzdálené řízení počítačů. Kdy ve zvolené topologii je potřeba spravovat všechny počítače vzdáleně ze serveru CORE . Navíc je nutný přístup serverů NODE na server NAS kde mají uložené virtuální stroje. Další nevýhodou je nutnost mýt program napsaný za pomocí Win32Api na všech počítačích systému a tím obtížnější vývoj tohoto programu, který by musel být měněn na několika místech a s tím související obtížnější rozšiřování platformy CMU kam by musel být tento program dohrán.

- **PowerShell**

Další možností automatizace je PowerShell, který je dodáván rovněž jako součást Platformy Microsoft Windows. Jedná se o interpretovaný jazyk určený primárně jako skriptovací pro ovládání celé platformy Microsoft Windows. Sám Microsoft se chlubí, že se jedná o nástroj, se kterým lze nastavit a vykonat cokoliv, co jde nastavit a vykonat graficky v samotném systému. Jakožto skriptovací jazyk má, ale i svou konzoli, která připomíná terminál a může být použit k ovládání Windows Pomocí příkazové řádky. Disponuje mimo těchto dvou hlavních rozhraní také programovatelné rozhraní pro programovací jazyky rodiny DOT.NET které slouží pro jednoduché spuštění příkazu PowerShellu a vrácení jeho výsledku. Bohužel jazyk C++ tímto rozhraním nedisponuje a při automatizaci PowerShellu je programátor odsouzen na svépomocí vytvořené funkce které tento jazyk zpracují, nebo na Win32Api.

I přes tento problém je PowerShell velice zajímavá volba a to hlavně z důvodů jeho komplexnosti, která zajišťuje jednoduchou správu virtuálních počítačů.

V případě použití domény na Windows serverech je dokonce možné spravovat vzdálený počítač bez přítomnosti klienta na tomto počítači pouze za pomocí správně upraveného PowerShell příkazu a speciálně pro tento účel upravenou politikou přihlašování, protože takto na dálku provedený příkaz musí mít přístup ještě ke třetímu počítači kde je uložen vlastní virtuální počítač. Toto ovšem v ideálním případě zajišťuje velice jednoduchou škálovatelnost, kde stačí počítač připojit do domény, kde automaticky proběhne nastavení, které doména definuje a počítač již je automaticky přidán do správy. Tak se celý proces přidání uzlu zkracuje na instalaci operačního systému a připojení do domény.

4.6.1. Topologie řízení

Vzhledem k předchozímu bylo rozhodnuto, že pro řízení virtuální infrastruktury bude použita kombinace programovacího jazyka C++ a PowerShell a že bude využito vzdáleného PowerShellu který umožní přítomnost aplikace na jednom počítači z celé Infrastruktury a na ostatních nebude tato aplikace Potřeba. To sice přináší komplikaci v podobě předávání autentizačních údajů pro vykonávání jednotlivých příkazů na vzdálených počítačích, ale za to velice ulehčuje vývoj aplikace a následnou škálovatelnost infrastruktury kdy poslední verzi není nutné nahrávat vždy na všechny servery, ale stačí,

když jí bude mít jeden z počítačů v superpočítači.

Jako vedlejší nevýhoda se projevilo, že pro přenášení autentizačních údajů je potřeba doména na platformě Windows kdy jeden server bude muset být doménovým řadičem a bude muset nést jednotné autentizační údaje pro všechny servery a těmto serverům bude muset umožnovat ověřovat platnost autentizačních údajů které byly odeslány spolu s příkazem v PowerShellu. Jelikož takto vypuštěné příkazy budou dělat i několik skoků mezi servery (typicky kdy CORE řekne NODE aby na NASu vytvořil virtuální počítač) je nutno aby autentizační údaje pro tyto příkazy platily v celém superpočítači CMU. K čemuž slouží výše zmíněná doména.

Výsledný Program tedy bude fungovat tak, že na jednom počítači bude běžet program, kam uživatel zadá kolik počítačů chce vytvořit, vybere si operační systém a konfiguraci počítače a to se pomocí vygenerovaných příkazů PowerShellu provede na vzdálených serverech kde bude vybraná topologie následně spuštěna. O čemž bude uživatel informován. Mazání takto vytvořené infrastruktury bude fungovat obdobě.

4.7. Program pro spouštění testů

Tento program byl navržen pro vlastní spouštění testů na virtuální infrastruktuře vytvořené prvním programem. Vlastní testy jsou reprezentovány několika programy s terminálovým ovládáním, které se spustí s určitými parametry a jejich výstup je poté potřeba uložit. Celé toto musí proběhnout vzdáleně na vlastním virtuálním počítači, který je před testem vytvořen a po testu smazán, proto je potřeba výsledky ukládat na fyzický server který bude celé testování řídit. Tímto uzlem bude řídící uzel celého superpočítače CMU-CORE.

Vlastní spouštění musí probíhat tak, že centrální uzel řekne všem virtuálním počítačům, kde se nachází a jaké testy mají provést. Po vykonání testů musí být výsledky odeslány zpět na řídící uzel, kde jsou uloženy k dalšímu zpracování.

Jako použitý programovací jazyk bylo vybráno opět C++ kde server je napsán pouze pro Windows a klienti jsou pro Windows i Linux. V systému Windows je klient i server jedna aplikace, která obsahuje obě funkčnosti.

5. Implementace

5.1. Nastavení a funkce Domény

5.1.1. Funkce

Doména v podání Firmy Microsoft je implementací LDAP serveru se specifickými funkcemi, která umožňuje centrální řízení celé skupiny počítačů jednou databázovou strukturou, která se jmenuje doména Active Directory. Tato doména je uložena na centrálním serveru nebo skupině serverů, který se jmenuje řadič domény. Počítače přidané do této domény získají k takovému řadiči přístup a nastavují se podle něj. Také toto umožnuje databázi doménových uživatelů, kteří se mohou přihlásit na jakémkoliv počítači v doméně jedněmi přihlašovacími údaji a používat zde svůj účet s právy danými doménou.

Takováto doména se stává strukturou několika typů logických a fyzických struktur, které popisují fyzické a logické rozložení uživatelů, počítačů a sítě.

Je možné vytvářet také skupiny domén neboli lesy. Jedná se o stromovou strukturu Domén, které sdílejí jeden výchozí doménoví řadič, který obsahuje centrální řízení přístupů a práv jednotlivých domén. Mezi těmito doménami a mezi jednotlivými počítači v doméně lze vytvořit vztahy důvěry, kde jednotlivé počítače pro komunikaci mezi sebou používají sítový protokol Kerberos. Toto umožnuje vzdálené řízení počítačů v doméně z jednoho centrálního bodu a vzdálené provádění příkazů PowerShellu kterého bylo v této práci hojně využito.

Pro správnou funkci potřebuje Doména DNS server na který mapuje přidávané objekty a používá ho pro komunikaci v rámci domény a autentizaci protokolu Kerberos doménovým řadičem.

5.1.2. Nastavení

- **DNS**

Pro Správnou funkci Domény byl nejprve nutný DNS server, pro který byla použita implementace DNS serveru od firmy Microsoft v rámci Windows 2012 R2. Tímto

serverem byla vytvořena top level Doména.cmu která je ovšem funkční pouze v rámci DNS serveru. Tato doména je použita jako standardní doména Pro celé CMU kdy každý z počítačů zde má svůj záznam. Tento DNS server je nutné používat v rámci celé domény a to už před přidáním počítače do domény a je nutné ho používat po celou dobu figurování počítače v doméně.

Všechny dotazy které nejsou v doméně. CMU tento server překládá za pomocí univerzitního DNS serveru, ovšem se zapnutou pamětí, kde často kladené dotazy si server zapamatuje.

- **Active Directory**

Vlastní vytvoření domény je velice pohodlné díky průvodci, který již Windows server 2012 R2 obsahuje. V tomto průvodci se vybere existující DNS server který bude doméně přiřazen, poté se zadají její jména a na daném serveru je doména hotova. Pro CMU byl jako doménový kontrolér zvolen CMU-CORE.

V takto vytvořené doméně byly zřízeny uživatelské účty pro všechny uživatele kteří měli na servery přístup a nastavena jim požadované práva. Poté byly všechny server přidány do domény, která od té chvíle sloužila jako jeden z možných přihlašovacích postupů do serverů. Výhodou bylo možnost spouštět příkazy powershellu na dálku, protože účty znaly všechny servery. A autentizace tak nebyla problém.

5.2. Systém pro tvorbu virtuální topologie

Jako první bylo třeba vyřešit jak spouštět vlastní spouštění Powershellu.

5.2.1. Automatizace PowerShellu

Pro C++ neexistuje žádný způsob jak přímo volat PowerShell a to již z důvodu že se jedná o jazyky interpretovaný a kompilovaný. První možností bylo spustit PowerShell jako příkaz terminálu jehož výstup se přesměruje do souboru který je následně zpracován. Toto řešení je funkční ovšem zbytečně pomalé.

Jako rychlejší řešení se jevilo vytvoření procesu PowerShellu a následné přesměrování jeho vstupů a výstupů pomoci Handle do jiného procesu který by již byl námi čitelný. Kde jsme ovšem narazili na několik problémů. Prvním byly dynamické posuvníky, které

PowerShell ukazuje jako průběh probíhajících úloh. Ty jsou uživatelsky sice přívětivé, ovšem jejich reálná funkce je implementována pomocí prostého překreslování obrazovky což výstup do druhého procesu zatížilo o velké množství zbytečného textu. Dalším problémem bylo, že vykonávání procesů PowerShellu trvá různě dlouho a výstup je vypisován i v průběhu tohoto procesu proto je těžké poznat, kdy proces vlastně skončil a čeká na další příkaz. Jako řešení se ukázalo vytvoření procesu terminálu, kde jsou prováděny příkazy PowerShellu pomocí klíčového slova PowerShell a jejich vstup a výstup je přesměrován pomocí handle. Takto vyvolaný proces jako výsledek vrátí najednou vypsaný výsledek PowerShellu a posléze nastaví příznak, že již dokončil provádění a je připraven přijmout další příkaz. Posléze je pomocí Funkcí pro čtení výsledků jiných procesů a výše zmíněných Handle přečíst po řádcích výsledek, který by se standardně vypsal do okna terminálu a ten vrátit.

Posledním problémem takto vytvořené funkce je problikávání okna terminálu, když je zpracováván. Jako řešení se ukázalo použít pro spouštění procesu funkci z knihovny Windows.h CreateProcces Která dovoluje ne jen nastavit handle ale jako parametr dovoluje i vytvořit proces bez grafického výstupu. Tudíž vše proběhne na pozadí a uživatel ani nezjistí, že byl terminál použit. Taktéž rychlost je ve srovnání se zpracováváním výstupu uloženého do souboru podstatně rychlejší. Celá funkce je Přiložena jako Příloha 1.

5.2.2. Příkazy PowerShellu

Pro správnou funkcionalitu aplikace bylo potřeba nebrat příkazy jako statické nýbrž je generovat pomocí uživatelem zadaných parametrů. Tyto parametry mohou obsahovat operační systém, který bude na vlastním virtuálním počítači spuštěn. Dále velikost operační paměti RAM, počet procesorových jader, které bude mít virtuální počítač k dispozici a počet takto vytvořených počítačů. Dále bude nutné umožnit spuštění a zastavení takto vytvořené topologie a její odstranění po dokončení vlastního testování.

- **Vytvoření Virtuálního pevného disku**

Pro vytvoření virtuálního počítače je nutné nejprve vytvořit Pevný disk, na kterém bude připravený operační systém podle volby uživatele. V rámci Této práce byly používány 4 základní operační systémy, ovšem není problém do systému přidat libovolný další

operační systém. Tyto systémy jsou Windows 8.1 x64, Windows 8.1 x32, Linux Debian x32 a Linux Debian x64. Tyto operační systémy byly nainstalovány na vzorové virtuální počítače ve své základní verzi a posléze na nich byly provedeny všechny dostupné aktualizace. Automatické aktualizace byly následně zakázány a tyto systémové disky byly vyexportovány na NAS, kde byly uloženy jako vzorové pevné disky od nichž se budou odvozovat generované pevné disky. S takto připravenými pevnými disky již pro vytvoření nového pevného disku jediný příkaz powershellu a to

new-vhd -ParentPath \\CMU-NAS\\VHD-examples\example.vhdx -Differencing -Path \\CMU-NAS\VHD\target.vhdx -ComputerName "CMU-NODE1"

Kde nový virtuální pevný disk se jmenuje target.vhdx a je odvozen od disku example.vhdx oba ty to disky jsou na NASu a jsou vytvářeny uzlemCMU- NODE1 - Differencing znamená, že disk je rozdílový a tudíž se ukládají pouze změny oproti výchozímu disku (což ušetří cca 8GB místa na disk a značně zrychlí jeho vytváření).

Parametry jsou doplňovány dynamicky pomocí uživatelem zadaných hodnot. Konkrétně vzor (example.vhdx) je dán podle operačního systému, který uživatel vybral a target je nastaven podle vnitřních regulí programu jako pořadové číslo virtuálního počítače a jeho operační systém. NODE je opět vybírán podle uživatelovy volby.

- **Vytvoření virtuálního počítače**

Pro vytvoření virtuálního počítače s dostatečným množstvím parametrů které jsou v rámci programu třeba bohužel neexistuje v powershellu jeden příkaz. Proto se tento proces rozpadá na dva kroky, kde je nejdříve virtuální počítač nastaven se základní konfigurací a následně je nastavena detailnější konfigurace tohoto počítače. Tyto kroky jsou v rámci programu neoddělitelné a vždy proběhnou oba.

Prvním krokem je:
new-vm target -MemoryStartupBytes 512MB -Generation gen -VHDPath \\CMU-NAS\VHD\target.vhdx -Switchname External -computername "CMU-NODE1"

Parametr target je stejný, jako v minulém případě nyní je ovšem použit jako identifikátor nově vznikajícího virtuálního počítače i jako identifikátor již dříve vytvořeného disku. -Switchname External určuje do jakého virtuálního switche je virtuální počítač připojen. V tomto případě se jedná o předem připravený virtuální switch který má přístup do reálné

sítě a zajištuje tak virtuálnímu počítači komunikaci přes síť. Parametr –VHDPath určuje již dříve připravený virtuální disk který bude k počítači připojen. A –computername je opět uzel na kterém je virtuální počítač vytvářen. Posledním parametrem je -Generation který určuje generaci virtuálního počítače, přičemž generace 1 je standardní virtualizovaný hardware a je použit u většiny systémů. Generace 2 je pokročilejší metoda virtualizace hardware, ale vyžaduje podporovaný operační systém (pouze Windows 8.1 x64 a novější) a pro tento systém byla použita. Parametr –MemoryStartupBytes určuje jak velikou bude mít virtuální počítač operační paměť při startu a je fixně nastaven na minimální hodnotu a to je 512MB.

Druhým krokem je nastavení virtuálního počítače:
set-vm -Name target -computername "CMU-NODE1" -Dynamicmemory -MemoryMaximumBytes memMB -MemoryMinimumBytes 512MB -ProcessorCount cpu, kde parametr target a –computername jsou použity jako v předchozích příkladech. –Dynamicmemory určuje, že paměť se bude dynamická tj že se bude měnit v rozsaku který je definován jako MemoryMinimumBytes což je spodní velikost paměti a je fixně daná na 512MB což je minimální možná hodnota. A hodnotou MemoryMaximumBytes kterou si uživatel zadává jako velikost operační paměti. V tomto rozsahu se paměť pohybuje podle potřeby daného virtuálního počítače a podle zdrojů které má fyzický počítač k dispozici. –ProcessorCount nastavuje počet virtuálních procesorů které budou přiděleny virtuálnímu počítači.

- **Spuštění a zastavení virtuálního počítače**

Vlastní spuštění a zastavení virtuálního počítače je implementováno dvojicí příkazů se stejnou strukturou.

Spuštění
start-vm target -ComputerName "CMU-NODE1"

Zastavení
stop-vm target -ComputerName "CMU-NODE1"

Kde target je opět identifikátor virtuálního počítače a computername Jméno uzlu na kterém počítač běží.

- **Načtení Virtuálních počítačů**

Jedním z posledních použitých příkazů je načtení informací o virtuálních počítačích. Toto je důležité pokud chceme vědět jaké jsou již na uzlu počítače a jaký je jejich stav popřípadě kolik prostředků je z fyzického počítače využito. K tomu slouží jedna funkce která zpracovává výsledek příkazu

get-vm –Name -ComputerName "CMU-NODE1"

Kde target je opět identifikátor virtuálního počítače a computername Jméno uzlu na kterém počítač běží.

Tento výstup je zpracován několika funkcemi podle toho jaký výstup chceme jedna z nich nám dává jen soupis existujících virtuálních počítačů, další poskytuje kompletní informace o nich a zaznamenává je.

- **Mazání Virtuálního počítače**

Pro mazání virtuálního počítače je třeba nejprve ověřit v jakém je tento počítač stavu poté ho případně zastavit a po zastavení smazat virtuální disk. Po smazání virtuálního disku je možné odstranit virtuální počítač. Toto je jediný způsob jak odstranit virtuální počítač tak, aby po něm nezůstávaly zbytečné soubory.

Vzhledem rozsahu tohoto postupu bylo zjištěno, že pokud jsou příkazy jednotlivě prováděny PowerShellem je tento postup příliš zdlouhavý a rychlejší je, když je prováděn pouze jeden blok powershellovských příkazů který je odeslán jako jeden příkaz a proveden.

Tento příkazový blok je následující:
GET-VM -computername "CMU-NODE1" -vmname target | GET-VMHardDiskDrive | Foreach { STOP-VM -Force -computername "CMU-NODE1" -vmname $_.VMname; Remove-item -path $_.Path; REMOVE-VM -force -computername "CMU-NODE1" -vmname $_.Vmname }"

Tento blok je odeslán najednou a provede následující kroky:

1. Najde cílový počítač na daném uzlu
2. Načte jeho disky

3. Pokud běží zastaví tento počítač (Force znamená neprodleně)
4. Odstraní její připojené virtuální disky
5. Odstraní virtuální počítač

Takto provedený script je rychlejší než jednotlivě vykonávané příkazy a navíc odpadá problém se zpracováváním jednotlivých příkazů. Navíc je velice snadno paralelizovatelný

5.2.3. Spouštění celých infrastruktur

Vzhledem k nutnosti spouštět několik desítek virtuálních počítačů zároveň je možné zadat počet opakování a systém příkazy provede několikrát podle definovaného počtu s různým identifikátorem a tak vytvoří několik virtuálních počítačů po sobě. Problémem je že rychlost takto vytvořeného systému je velice nízká vzhledem k využití jednoho PowerShell rozhraní, které vždy čeká na vytvoření předchozího virtuálního počítače což trvá dlouho, ale nevyužívá dostatečně prostředky daného systému. Proto jsem přistoupil k vytvoření několika paralelních procesů, které vytvářejí vždy po jednom virtuálním počítači. Ukázalo se, že u takto z paralelizované funkčnosti trvá vytvoření 20 virtuálních počítačů stejnou dobu jako, vytvoření jednoho virtuálního počítače.

Takto byly z paralelizovány funkce pro hromadné vytvoření (obsahuje spuštění) a hromadné smazání virtuálních počítačů. Části, které paralelizovat nejde a to je určování jednoznačných identifikátorů jsou z této funkce vyňaty a vloženy před vlastní paralelismus. Tento paralelismus je přiložen ve zdrojovém kódu aplikace v přílohách.

5.2.4. Výsledný program

Výsledný program je poskytován ve dvou rozdílných verzích. Kde jedna je dodána jako DLL knihovna pro použití v následujících programech. Druhá verze programu je vlastní ovládací program s grafickým rozhraním, které je vytvořeno také v C++.

- **DLL**

DLL je poskytován jako Standardní DLL, které obsahuje funkce pro: Zastavení a spuštění virtuálního počítače, načtení virtuálních počítačů a informací o nich, dále funkce pro vytvoření a smazání virtuálních počítačů a to buď v jednom nebo více jádrech a po jednom nebo více kusech.

- **Grafický Program**

K práci s tímto programem bylo vytvořeno grafické rozhraní pomocí VCL Forms v C++, která používá stejné funkce, jako výše zmíněné DLL pouze pro ně vystavuje grafické ovládání a parametry si bere z textových polí, které uživatel vyplní.

5.3. Systém pro spouštění testů

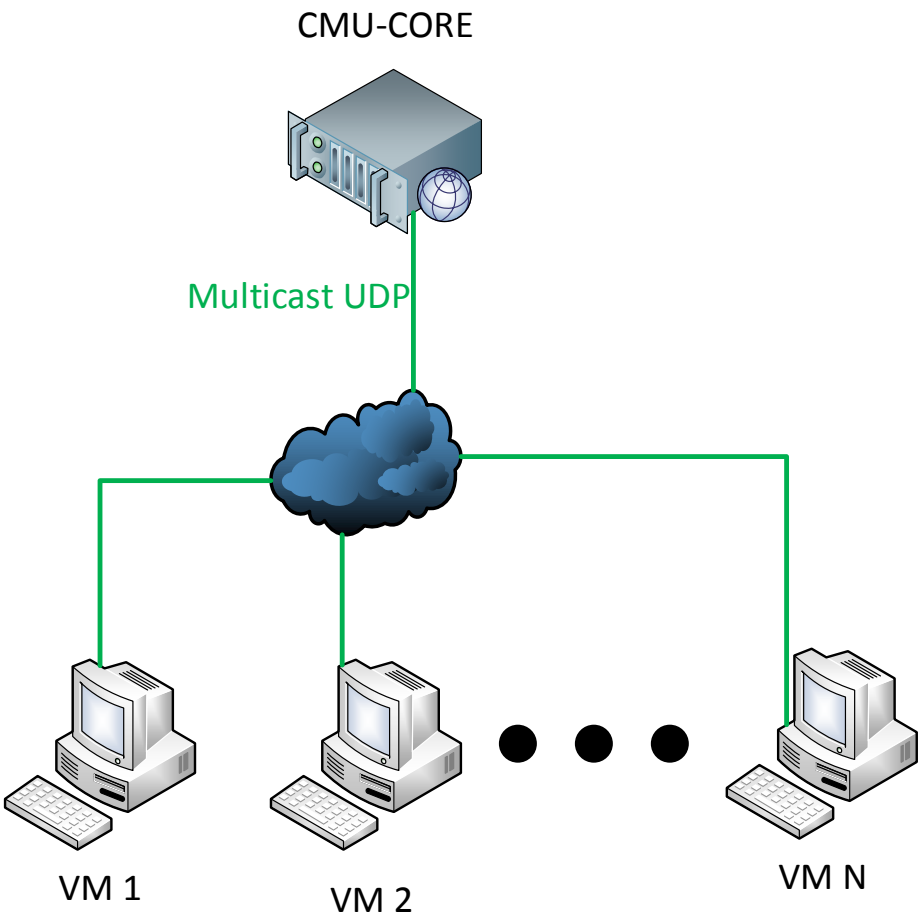
Tento program vychází z předpokladu, že máme prvním programem vytvořenou virtuální infrastrukturu, o které nevíme nic kromě toho, že běží v lokálním segmentu sítě (tj. připojená pouze přes Switche a Huby nikoliv Routry) po vytvoření o těchto počítačích nevíme nic kromě tohoto faktu. Vzhledem k tomu byl Tento program navrhnut, jako 2 celky kdy prvním je klient který je naprogramován pro Linux i Windows a druhým je server který byl naprogramován pro Windows. Klientská část je vždy připravena v každém virtuálním počítači kde je nastavena aby se spustila automaticky po stratu systému a serverová část je na libovolném počítači v rámci lokálního segmentu, popřípadě je připojen k tomuto segmentu přes VPN.

5.3.1. Spuštění

Vlastní spouštění je rozdílné pro klientský počítač (v tomto případě počítač virtuální) o kterém nevíme nic kromě toho, že je ve stejném lokálním segmentu a spouštění serveru (v tomto případě CMU-CORE) který je v lokálním segmentu stejně jako klientské počítače a jako jediný ze všech počítačů je obsluhován člověkem.

- **Spuštění Serveru.**

  Program serveru je spouštěn na CMU-CORE. Tento program je spouštěn, pouze pokud jsou testy potřeba a neběží neustále. Ve chvíli kdy je server spuštěn si nejdříve zjistí, jaké jsou v počítači síťové karty. Pro každou nalezenou síťovou kartu vytvoří vlastní thread na kterém začne vysílat multicast což je upravená UDP komunikace. V tomto multicastu je obsažena IP adresa karty ze které daný thread vysílá. Zároveň se na všech kartách poslouchá a čeká na TCP spojení které budou klienti, kteří obdrží multicast navazovat.



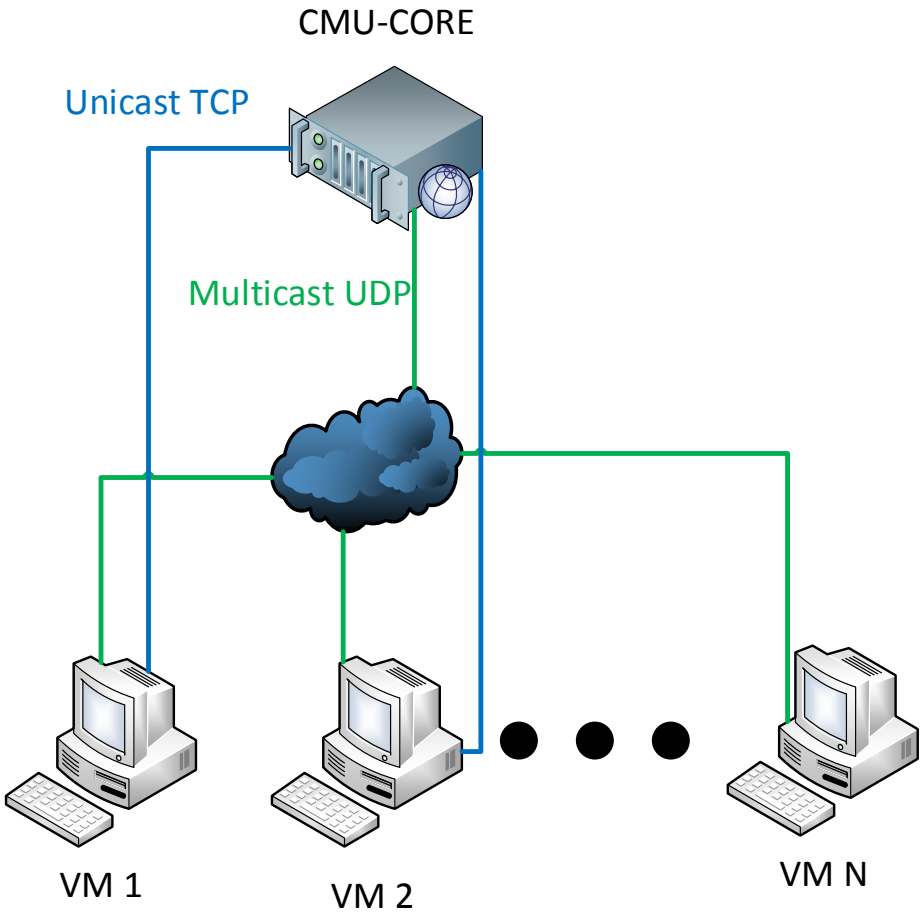
Vysílaný multicast na všech rozhraních serveru

- **Spuštění Klientských počítačů.**

  Klientské počítače jsou prvním programem vygenerované virtuální počítače, které jsou ihned po vygenerování spuštěny a nemají žádnou lidskou obsluhu. Z toho důvodu se klientský program v těchto počítačích spouští automaticky po startu systému a ihned spouští režim klienta hledání serveru. Toto hledání znamená, že klient začne na všech Sítových kartách, které jsou v počítači poslouchat, zda na ně nejde muticast od serveru. Pokud tento multicast zachytí, vypíná režim hledání (ukončí vyhledávání multicastu) a přechází do režimu ovládání. Kdy s adresou která je vložena do multicastu naváže TCP spojení na kterém očekává příkazy od serveru.

Ve chvíli kdy server zachytí příchozí TCP spojení od klienta tak toto spojení naváže a

začne s ním komunikovat. Nadále však vysílá multicast pro připojení dalších klientů. Tento multicast je vysílán po celou dobu funkčnosti serveru i když komunikuje s jednotlivými klienty pro případ, že se bude v průběhu chtít připojit další klient (například při postupném spouštění virtuálních počítačů).



Navázaná TCP spojení s neustálím vysíláním multicastu klientům

5.3.2. Příkazy a jejich funkce

Po vytvořených TCP spojeních s jednotlivými klienty server komunikuje pomocí několika základních příkazů které byly pro tento systém definovány a na obou stranách spojení implementovány.

- **getConfiguration#**

  Ihned po navázání spojení je vždy odeslán klientovy tento příkaz. Po jeho obdržení klient zjistí, co obsahuje za konfiguraci, operační systém a zda jeden o virtuální či fyzický počítač. Tyto informace odešle serveru. Server na základě těchto informací

ví, s jakým počítačem komunikuje a může podle toho přizpůsobit testování. Jelikož fyzický stav systému se nemění je nové volání v rámci jednoho spojení zbytečné volat.

- **Execute#Příkaz#**

  Jedná se o jeden z příkazů pro vlastní ovládání vzdáleného počítače, kdy jako parametr tohoto příkazu je přiložen příkaz pro operační systém. Po přijetí tohoto příkazu klient spustí přiložený příkaz a počká na jeho vykonání. Ve chvíli kdy je příkaz vykonán a klient má jeho výstup k dispozici odešle tento výstup zpět na server. Po přijetí výsledku server tento výstup uloží do souboru a oznámí, že byl příkaz vykonán. Jeho výsledek také zobrazí. Takto uložené výsledky slouží pro pozdější statistické zhodnocení vlastních testů.

- **ExecuteNB#Příkaz#**

  Jde o příkaz, který server odesílá klientovy v případě, že jde o provedení příkazu a ne o jeho výsledek. Tento příkaz je odeslán s parametrem ve kterém je stejně jako v případě Execute příkaz operačního systému. Klient tento příkaz spustí a v tu chvíli odpovídá serveru zprávou že, příkaz byl spuštěn. Na výsledek příkazu se nečeká ani není nikde ukládán. Toto je vhodné zejména pro spouštění „grafických“ aplikací jako je například Midnight Commander. Kde nám nejde o vlastní výsledek ale o spuštění aplikace jako takové se kterou chceme později pracovat.

- **Quit#**

  Poslední z požívaných příkazů slouží k ukončení spojení ze strany serveru. Pokud server pošle tento příkaz klientovy ten ukončí TCP spojení se serverem a přejde opět do fáze hledání serveru.

5.3.3. Ukončení systému

Ukončení takovéhoto systému probíhá vždy ze strany serveru. Ukončení ze strany klienta nepřipadá v úvahu, protože vlastní klienti nemají obsluhu. Pokud by ale došlo k chybě a klienti se odpojili server toto ohlásí a považuje klienta za ukončeného.

Správné ukončení ze strany serveru probíhá nejdříve ukončení vysílání multicastu, poté odesláním zprávy Quit všem klientům, kteří přejdou do fáze hledání serveru. Až poté co

jsou všichni klienti obeznámeni s ukončením serveru je ukončen vlastní server jako aplikace.

5.3.4. Výsledná aplikace

Výsledná aplikace je k dispozici ve dvou verzích podle operačního systému, na kterém běží. Obě tyto aplikace jsou napsány v C++.

- **Windows**

  Aplikace je Grafická a obsahuje jak Klienta, tak server přičemž Klient se spouští s parametrem –c. Nebo se do něj dá přepnout po spuštění grafické aplikace. Server se spouští v rámci té samé grafické aplikace kde se zadávají příkazy pro klienty a také se zde objevují výstupy programu. Grafické část byla vytvořena pomocí technologie VCL.

  parametry pro spouštění z příkazové řádky jsou:

  -s pro běh jako server

  -c pro běh jako klient

  -a adresa, ke které se připojí klient, nebo na které poslouchá server

  -n nepoužívá automatické nastavení pres multicast

- **Linux**

  Aplikace pro Linux má terminálové ovládání a je v ní dostupný pouze klient.

  parametry pro spouštění z příkazové řádky jsou:

  -a adresa, ke které se připojí klient, nebo na které poslouchá server

  -n nepoužívá automatické nastavení pres multicast

6. Testování

6.1. Testování vytváření Virtuálních počítačů

Jelikož oba výstupy tohoto programu (Grafická aplikace a DLL knihovna) obsahují stejnou implementaci bylo testování provedeno pouze na grafické aplikaci a u DLL knihovny byla tato funkčnost pouze ověřena. Vlastní testy probíhaly ve specifických podmínkách superpočítače CMU s připravenou doménou a s funkčním vzdáleným voláním PowerShellu. Bez zajištění těchto podmínek aplikace nemůže fungovat.

Při správně nastavené doméně byla aplikace testována na všechny možné scénáře použití a také na nesprávné použití například zadáním špatných hodnot nebo vypnutím aplikace v kritických okamžicích, v době vytváření a mazání virtuální infrastruktury.

Po provedení těchto testů byla aplikace shledána plně funkční, protože se nevyskytl problém s její funkčností.

6.1.1. Testování za provozu

Při testování za provozu byla aplikace poskytnuta pro vlastní testování, kdy plně sloužila jako jediný ovladač technologie Hyper-v. Celím Testováním prošla aplikace bez jediné stížnosti a nebylo nutno opravovat žádnou chybu nebo nefunkčnost aplikace.

Po tomto testování byla aplikace shledána za Funkční a další testování, jež neprobíhalo.

6.2. Testování spouštění testů

Vlastní spouštění testů bylo po vytvoření otestováno na různých sítích, kde bylo potřeba navázat komunikaci. Potvrdilo se, že pokud jsou klient a server v rámci jednoho routru navázat komunikaci není problém. Pokud by v jednom segmentu nebyly je nutno použít VPN pro virtuální připojení do jednoho segmentu, nebo nastavit na routrech přes které má daný tok probíhat PIM protokol. Po tomto testování bylo opět přistoupeno k Testování za provozu.

6.2.1. Testování za provozu

Testování za provozu proběhlo společně s testováním Programu pro spouštění virtuálních topologií při vlastním testováním superpočítače CMU a proběhlo bez problémů.

7. Návrhy pro budoucí rozvoj

Návrhem pro budoucí rozšiřování tohoto systému je jednoznačně začlenění více různých hypervizorů, což by přineslo možnost reálného porovnání těchto hypervizorů mezi sebou a možnost jasně rozhodnout o výhodách různých implementací plné virtualizace. Tento krok ovšem přináší Velký problém s automatizací, protože Další hypervizory nefungují pod stejným operačním systémem jako Hyper-v a možnost automatizace přes powershell je zde více či méně omezena (například VMware má prostředí pro PowerShell, kterým je možno ho vzdáleně ovládat, nicméně není ani zdaleka tak propracované jako prostředí od Microsoftu)

8. Závěr

Virtualizace je dnes velice rozsáhlá problematika a rozvětvuje se na několik směrů. Všechny tyto směry mají své klady a zápory a pro každý konkrétní případ se může hodit jiná z nich. Zde byla vybrána ta nevhodnější pro superpočítač CMU a byla popsána její funkce a postupy pro její nasazení na superpočítači čítajícím více než jen jeden uzel kde je používáno centrální řízení a centrální sítové úložiště ale není zde omezena škálovatelnost tohoto systému.

Pro tento systém byly implementovány dvě aplikace, kde jedna je postavena přímo na míru tomuto superpočítači a slouží pro ovládání virtualizační technologie a rychlé vytváření a odstraňování virtuálních infrastruktur. Zde byly nalezeny zajímavé postupy pro optimalizaci jak z hlediska rychlosti, tak z hlediska místa které jednotlivé počítače ukládají na síťové úložiště. Druhou z aplikací je aplikace pro vzdálené řízení jednotlivých v tomto případě virtuálních počítačů u které se ukázalo že její použití je o hodně širší než jenom v oblasti virtualizace. A jsou v ní používány zajímavé technologie například ne často využívaný multicast.

Tato dvojice aplikací společně se superpočítačem jsou funkčním nástrojem pro otestování virtualizační technologie, které skýtají nejednu příležitost pro práci s virtuálními počítači a pro rozšiřování tohoto systému.

Z tohoto mám pocit, že cíle bakalářské práce se povedlo splnit.

9. Literatura

[1] B. DuCharme, *The operating systems handbook: UNIX, OpenVMS, OS/400, VM and MVS*. New York: McGraw-Hill, c1994, p. xvii, 390 p.

[2] G. Popek and R. Goldberg, "Formal requirements for virtualizable third generation architectures", in *Ultraviolet observations of the Be star and X-ray binary 4U 1145-61 (=HD 102567=Hen 715) obtained with the IUE: a monthly publication of the ACM Publications Office*, 1974, vol. 17, no. 7, pp. 412-421.

[3] Microsoft, "Microsoft Developer Network", *Microsoft Developer Network*, 2009. [Online]. Available: https://msdn.microsoft.com. [Accessed: 16-03-2015].

[4] "Paravirtualization (PV)", *PC mag*, 2014. [Online]. Available: http://wiki.xenproject.org/wiki/Paravirtualization_%28PV%29. [Accessed: 16-03-2015].

[5] M. Johnson, *Application Virtualization: What you Need to Know For IT Operations Management*, 2012 ed.. online: Emereo Publishing, 2012.

[6] "Everything You Need to Know About the Intel Virtualization Technology", *hardware secrets*, 2012. [Online]. Available: http://www.hardwaresecrets.com/printpage/Everything-You-Need-to-Know-About-the-Intel-Virtualization-Technology/263. [Accessed: 16-03-2015].

[7] . Wales, "Wikipedia", 2001. [Online]. Available: cs.wikipedia.org. [Accessed: 19-03-2015].

[8] R. Goldberg, "Survey of virtual machine research", in *Computer*, 1970, vol. 7, no. 6, pp. 34-45.

10.Přílohy

Seznam Příloh:

1. Třída pro vytvoření procesu a přesměrování výsledku

```
#ifndef commandProcesorH
#define commandProcesorH

#include <iostream>
#include <stdio.h>
#include <windows.h>
#endif

using namespace std;

class commandProcesor
{
public:
string commandProcesor::ProcessCommand(const string
rCommandLine);
};
#include "commandProcesor.h"

string commandProcesor::ProcessCommand(const string
rCommandLine)
{
      SECURITY_ATTRIBUTES secAttr;
      secAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
      secAttr.bInheritHandle = TRUE;
      secAttr.lpSecurityDescriptor = NULL;

      HANDLE writeInChild, readFromChild;
```

```
	if(!CreatePipe(&readFromChild, &writeInChild, &secAttr, 0))
	{
		return "ERR";
	}
	SetHandleInformation(readFromChild, HANDLE_FLAG_INHERIT,
0);

	PROCESS_INFORMATION procInfo;
	STARTUPINFO startupInfo;

	ZeroMemory(&procInfo,    sizeof(procInfo));
	ZeroMemory(&startupInfo, sizeof(startupInfo));
	startupInfo.cb         = sizeof(startupInfo);
	startupInfo.hStdError  = writeInChild;
	startupInfo.hStdOutput = writeInChild;
	startupInfo.hStdInput  = INVALID_HANDLE_VALUE;
	startupInfo.dwFlags   |= STARTF_USESTDHANDLES;

	std::wstring unicode(rCommandLine.begin(),
rCommandLine.end());
	wchar_t* commandLineCopy =
const_cast<wchar_t*>(unicode.c_str()) ;

	BOOL result = CreateProcess(NULL,
		commandLineCopy, // command line
		NULL,          // process security attributes
		NULL,          // primary thread security attributes
		TRUE,          // handles are inherited
		CREATE_NO_WINDOW,             // creation flags
		NULL,          // use parent's environment
		NULL,          // use parent's current directory
		&startupInfo,  // STARTUPINFO pointer
		&procInfo);    // receives PROCESS_INFORMATION
```

```
	if(!result)
	{
		CloseHandle(writeInChild);
		CloseHandle(readFromChild);
		return "ERR";
	}
	startupInfo.dwFlags = STARTF_USESHOWWINDOW ||
STARTF_USESTDHANDLES;
	startupInfo.wShowWindow = SW_HIDE;
WaitForSingleObject(procInfo.hProcess, -1);


	TerminateProcess(procInfo.hProcess, 0);


	CloseHandle(procInfo.hProcess);
	CloseHandle(procInfo.hThread);


	std::string s="";
	DWORD dwRead = 0;


	int BUFSIZE = 4096;
	CHAR chBuf[4096];
	BOOL bSuccess = TRUE;
	DWORD dwTotalAvailBytes;
	DWORD dwBytesLeft;


	for (;;)
	  {
		bSuccess = PeekNamedPipe(readFromChild, NULL,
BUFSIZE, &dwRead, &dwTotalAvailBytes, &dwBytesLeft );
		if (!bSuccess || dwRead == 0)
			break;
		bSuccess = ReadFile( readFromChild, chBuf, BUFSIZE,
&dwRead, NULL);
		s += chBuf;
	  }
	return s;
}
```