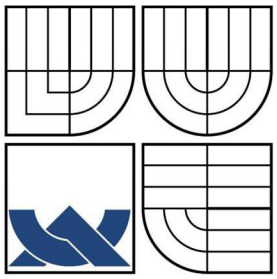
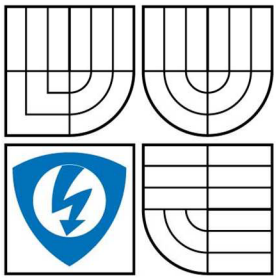


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ŘEŠENÍ PRO CLUSTEROVÁNÍ SERVERU

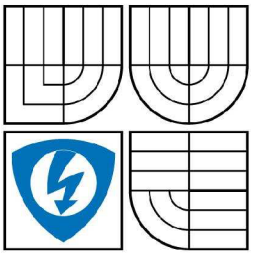
TITLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. MARTIN ČECH
AUTHOR

VEDOUCÍ PRÁCE — ING. TOMÁŠ PELKA
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Martin Čech	***ID:***	83089
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Řešení pro clusterování serverů

POKYNY PRO VYPRACOVÁNÍ:

Cílem této diplomové práce je důkladně zmapovat a zanalyzovat open source prostředky pro rozkládání záteže a rešení vysoké dostupnosti, se zaměřením na oblasti úloh ve kterých jsou open source rešení typicky nasazovány. Těmito oblastmi jsou především řešení síťové infrastruktury (routery, loadbalancery), obecné síťové a internetové služby a paralelní souborové systémy. Další částí této práce představuje výkonnostní rozbor realizovaného výpočetního clusteru.

DOPORUČENÁ LITERATURA:

[1] SLOAN, Joseph. High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI. [s.l.] : O'Reilly Media, Inc. , 2004. 368 s. ISBN 978-0596005702.
[2] LUCKE, Robert W. . Building Clustered Linux Systems. 1st edition. [s.l.] : Prentice Hall, 2004. 648 s. ISBN 978-0-13-144853-7.
[3] BOOKMAN, Charles. Linux Clustering: Building and Maintaining Linux Clusters. [s.l.] : Sams, 2002. 228 s. ISBN 978-1-57870-274-9.

Termín zadání:	9.2.2009	***Termín odevzdání:***	26.5.2009

Vedoucí práce: Ing. Tomáš Pelka

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Smlouva strana 1

ABSTRAKT

V práci je uveden rozbor Open Source Software (dále jen OSS), který umožňuje využívat a vytvářet počítačové clustery. Je zde prozkoumána problematika clusteringu a sestavování clusterů. Veškeré instalace konfigurace a správa clusteru byly prováděny na operačním systému GNU/Linux. Je zde uveden OSS umožňující sestavit úložný cluster, cluster s rozložením zátěže, cluster s vysokou dostupností a výpočetní cluster. Teoreticky byly rozebrány různé druhy benchmarků, pomocí kterých je pak měřen výkon a ten je možné dále srovnávat, např. se seznamem TOP500 nejvýkonnějších clusterů dostupným online. Praktická část práce se zabývá porovnáním výkonu výpočetních clusterů. S několika desítek výpočetních uzlů byl sestaven cluster na který byl nainstalován balík OpenMPI, který umožňuje paralelizaci výpočtů. Následně byly provedeny testy s programem High Performance Linpack, který pomocí výpočtů lineárních rovnic stanoví výkon celku. Testoval se také vliv paralelizace na algoritmus PEA. Pro praktické využití clusteru byl testován program John The Ripper na luštění přihlašovacích hesel. V práci je uvedeno množství grafů objasňujících funkci a hlavně znázorňující dosažené výsledky.

KLÍČOVÁ SLOVA

Výpočetní cluster, Benchmark, GNU / Linux, High performance computing – HPC, High Performance Linpack – HPL, John The Ripper – JTR,

ABSTRACT

The work is given an analysis of Open Source Software (further referred as OSS), which allows use and create computer clusters. It explored the issue of clustering and construction of clusters. All installations, configuration and cluster management have been done on the operating system GNU / Linux. Presented OSS makes possible to compile a storage cluster, cluster with load distribution, cluster with high availability and computing cluster. Different types of benchmarks was theoretically analyzed, and practically used for measuring cluster's performance. Results were compared with others, eg. the TOP500 list of the best clusters available online. Practical part of the work deals with comparing performance computing clusters. With several tens of computational nodes has been established cluster, where was installed package OpenMPI, which allows parallelization of calculations. Subsequently, tests

were performed with the High Performance Linpack, which by calculation of linear equations provides total performance. Influence of the parallelization to algorithm PEA was also tested. To present practical usability, cluster has been tested by program John the Ripper, which serves to cracking users passwords. The work shall include the quantity of graphs clarifying the function and mainly showing the achieved results.

KEY WORDS

Computing cluster, Benchmark, GNU / Linux, High performance computing – HPC, High Performance Linpack – HPL, John The Ripper – JTR,

ČECH, M. *Řešení pro clusterování serverů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 77 s. Vedoucí diplomové práce Ing. Tomáš Pelka.

PROHLÁŠENÍ:

Prohlašuji, že svou diplomovou práci na téma řešení pro clusterování serverů jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne ……... ……...

podpis autora

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu diplomové práce panu Ing. Tomáši Pelkovi za ochotu, vstřícné jednání a za užitečnou metodickou pomoc.

V Brně dne …… ……..

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Dnešní doba je dobou internetu. Lidé využívají rozmanité služby, které jim toto médium poskytuje. Pro mnoho z nás jsou služby na internetu zdrojem příjmů a zisků, proto je nutné u nich zachovat co možná největší dostupnost. Jestliže je určitá webová aplikace či stránka velmi žádaná, je v našem zájmu zajistit její dostupnost a pokud je to nutné, použít mechanizmy pro rozložení zátěže. K tomu je určeno mnoho komerčního, ale především Open Source softwaru (OSS). Svobodný software je dostupný se svými zdrojovými kódy a proto si ho můžeme individuálně upravovat a ladit. Můžeme tímto způsobem dosáhnout někdy i lepší výsledky než se softwarem komerčním.

Na začátku práce budou podrobně rozebrány všechny dostupné možnosti zapojení a využití počítačových clusterů. Zaměření bude především na OSS a jiné nekomerční prostředky, které umožňují rozložení zátěže na uzly clusteru (LB – load balancing), a nebo také zajištění vysoké dostupnosti používaných služeb (HA – high availability) v případě výpadku na síti. Jiný software umožňuje využití výpočetního výkonu clusteru či sdílení diskové kapacity. Využití OSS nachází především v síťové infrastruktuře – routery a loadbalancery, využívají ho také paralelní filesystémy a obecné síťové, či internetové služby.

Praktickou ukázkou bude sestavení clusteru z několik uzlů s použitý OSS operačním systém na bázi GNU/Linux, který bude podle zvoleného účelu použití dále testován. Jedním z možných testovacích nástrojů je program Linapack, který nalezneme ke stažení na stránkách [13]. Tento program slouží k měření počtu operací při výpočtu čísel s plovoucí desetinnou čárkou za 1 sekundu (Mflops/s). Výsledky programu můžeme srovnávat s ostatními počítačovými výpočetními systémy. Takto se pak hodnotí i ty největší počítačové clustery nebo superpočítače, které nalezneme na webových stránkách [3].

1 GNU/Linux

Linux je jádro (kernel) počítačových mnoha operačních systémů, původně vyvinutý Linusem Torvaldsem v roce 1991. Jeho zdrojový kód je volně k dispozici tzn. kdokoliv jej může používat a upravovat. Spojení GNU/Linux označuje kernel ve spojení s GNU knihovnami, nástroji a ostatním OSS. Dá se tedy říci, že pokud se určitá skupina spravuje operační systém GNU/Linux vzniká linuxová distribuce s velkým množstvím softwarových balíků, aktualizacemi a většinou také jednoduchou instalací. Dále v práci bude slovo Linux vyjadřovat spojení GNU/Linux tedy volný operační systém.

Zpočátku Linux využívali a spravovali hlavně počítačoví nadšenci v malých komunitách. V poslední době však vznikají nové distribuce s podporou velkých firem jako IBM, Hewlett-Packard, Red Hat, Novel, které tento systém používají na svých serverech. GNU/Linux však získává stále větší oblibu mezi širokou veřejností. Hlavními výhodami jsou možnost volby z velkého množství distribucí, nízké nebo žádné náklady, flexibilita, bezpečnost a spolehlivost. GNU/Linux tedy nachází využití v osobních počítačích (PC), také embedded systémech a superpočítačích. U PC je podíl GNU/Linux malý, kolem 1% [1], u superpočítačů je tomu naopak, zhruba 87,8% [3] [2].

Vysoké procento použití Linuxu jako operačního systému u superpočítačů je dáno bezespornými výhodami a možnostmi konfigurace OS. Chceme-li dosahovat vysokých výkonů je nutné použitý systém “vyladit“.

[1] Průměrné hodnoty z http://toplist.cz/global.html
[2] Statistika z 11/2008

2 Cluster

Cluster je virtuální počítač složený z několika vzájemně propojených PC. Jednotlivé PC (uzly) jsou nejčastěji vzájemně propojeny pomocí rychlé lokální sítě. Cluster složený z počítačů označujeme jako NOW (network of workstations) nebo COW (cluster of workstations). Protějškem clusteru jsou superpočítače (paralelní počítače), které používají velké množství procesorů a operační paměti, aby dosáhli vysokého výpočetního výkonu. Nevýhodou clusterů je, že pracují na síti, která má vetší zpoždění než sběrnice u superpočítačů. Výhod je hned několik. Cluster můžeme rozšiřovat o další uzly, omezení tvoří propustnost sdílených složek (hlavní uzel, použitá síťová technologie), tzv. **škálovatelnost** (výkonová přizpůsobivost). **Odolnost vůči výpadku** (fault tolerance) jednotlivých uzlů je další pozitivum. Pokud vypadne určitý uzel, který není klíčovou částí clusteru (centrální uzel, sdílená část sítě), omezí se celkový výkon, ale funkce zůstává zachována. Cluster je schopen paralelního zpracování úloh na více uzlech, čím dosahuje úměrný **vysoký výkon.**

V neposlední řadě spojením průměrně výkonných PC do clusteru jsme schopni ušetřit nemalé finanční prostředky za velmi drahý superpočítač.

2.1 Rozdělení clusterů

Clustery rozdělujeme podle účelu použití na několik typů. Každému typu je přizpůsobeno hardwarové softwarové řešení. Důležitá je také správné nakonfigurování systému. Více je možno nalézt ve [2].

2.1.1 Cluster s vysokou dostupností

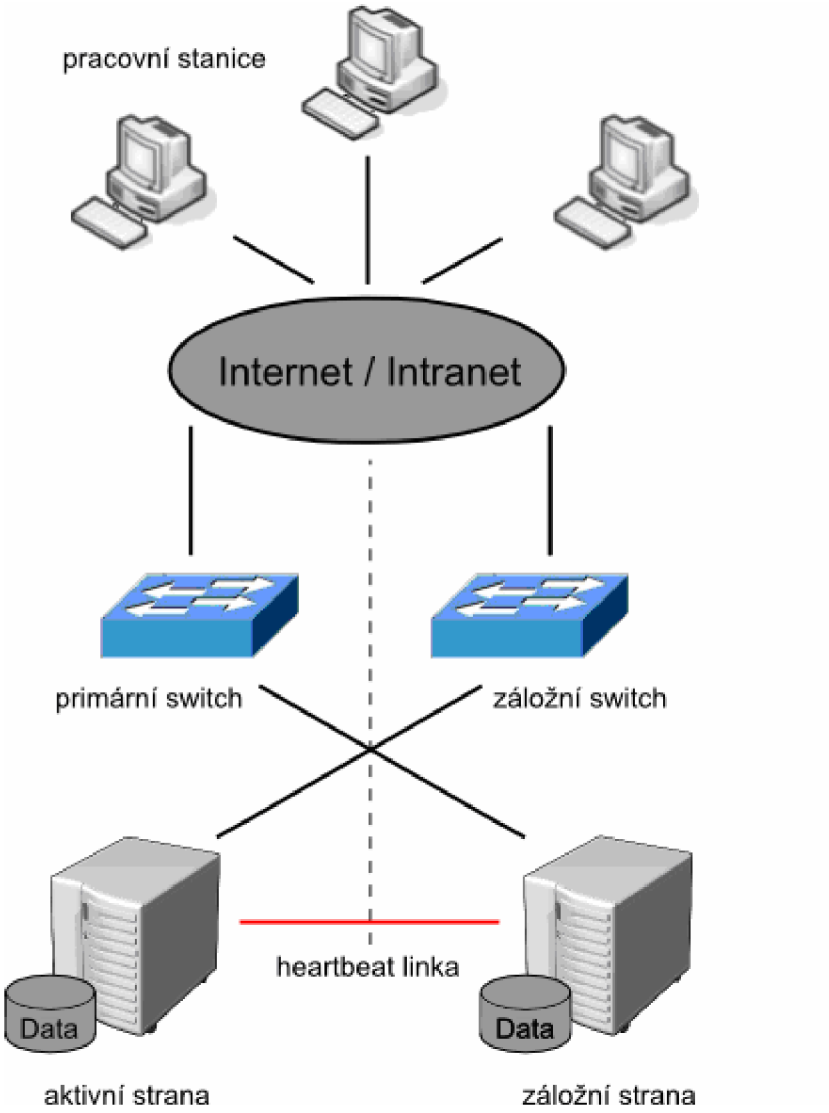
Také high availibility (failover) cluster , je seskupení počítačů s primárním účelem zajistit vysokou dostupnost služeb, poskytovaných tímto celkem. Pro funkci je nutné mít redundantní uzly, které převezmou služby prvků, co by případně selhaly. Při výpadku se služby startují automaticky a okamžitě na záložním uzlu bez zásahu administrátora. Popsaný proces se nazývá **failover.** Pracuje nezávisle, bez nutností zásahu pověřené osoby. Software pro HA-

cluster dovoluje konfiguraci serveru před spuštěním služby, jako např. volba vhodného filesystému, formát a mountování disků, konfiguraci síťového rozhraní, či spouštění podprogramů.

Tento cluster se používá pro vysokou dostupnost databází internetových obchodů, sdílení souborů a aplikací na síti. Při sestavení HA-clusteru se používá vícenásobného síťového spojení, zdvojení úložného prostoru a také připojení k různým okruhům elektrické sítě v kombinaci se záložními zdroji UPS (Uninterruptible power supply).

Pro předávání informací o statusu jednotlivých uzlů slouží tzv. Heartbeat linka. Jde o privátní síť spojující všechny uzly v clusteru. Tato ošetřuje stav split-brain, kdy dochází k mylné informaci záložního uzlu o kolizi uzlu hlavního. Pokud hlavní uzel stále běží mohlo by dojít k porušení dat na sdíleném disku při současném zápisu z více míst.

Nejčastěji se pro HA-cluster používají dva uzly, které jsou také minimem. Do clusteru je však možné připojit mnohem více uzlů. Pro řízení existuje několik možnosti konfigurace. Dva hlavní druhy modelů jsou Active/Passive a Active/Active. Při prvním zmíněném (A/P) je schopen záložní uzel převzít a spustit všechny služby hlavního uzlu. U A/A konfiguraci běží na každém uzlu primární služba. Pokud dojde k výpadku, jeden z uzlů ke své službě spustí službu, která běžela na uzlu s poruchou.



Obr. 1: Princip zapojení Linux HA

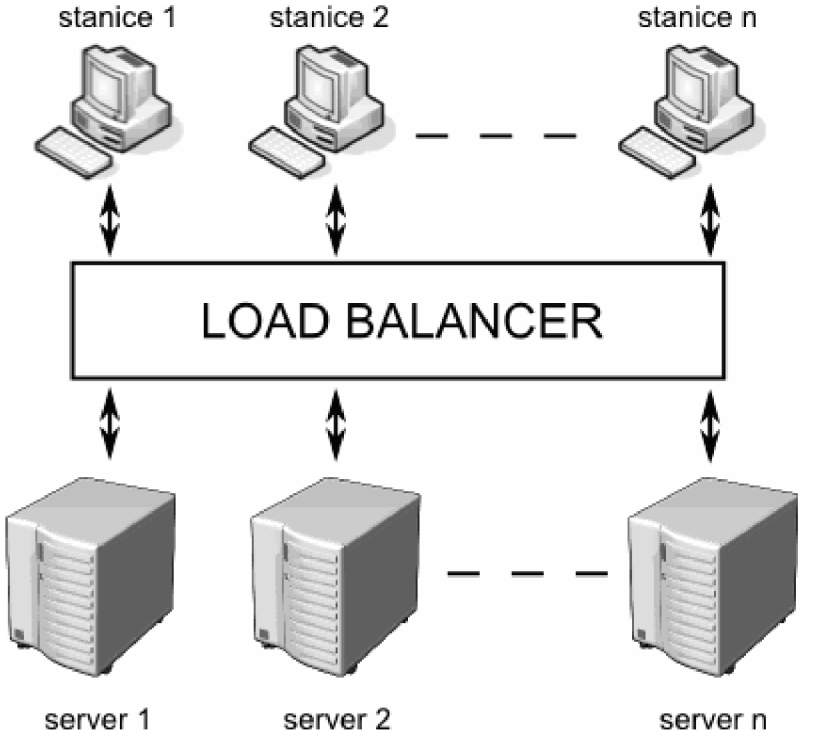
2.1.2Cluster s rozložením zátěže

Účelem clusteru s rozložením zátěže (load-balancing LB) je rozdělit výkon rovnoměrně mezi uzly, které jsou méně využívány. Příkladem použití je vytížený cluster složený z několika webových serverů. Při nasazení LB dochází k rovnoměrnému využití všech serverů. Toho se dá dosáhnout aplikací jednoduchého algoritmu **Round-Robin DNS,** kdy DNS server při dotazu klienta odpoví pokaždé jinou IP adresu cíle (uzlu). Tento jednoduchý způsob rozděluje požadavky na zpracování bez ohledu na aktuální množství spojení nebo času reakce. Je tedy vhodný, pokud jsou v clusteru stejně výkonné servery. V opačném případě to může vést k nerovnoměrnému dynamickému zatížení clusteru.

Rozšířením předchozího algoritmu vznikl **Weighted Round-Robin.** Tzv. vážený Round-Robin zohledňuje různé možnosti každého serveru v clusteru. Administrátoři ručně nastaví serverům priority pro zpracovávání požadavků podle jejich výkonu. Algoritmus už pak samostatně vhodně směřuje požadavky na servery, tak aby nedošlo k jejich přetížení.

Další algoritmus se jmenuje **Least-Connection**. Tento posílá žádosti na server v clusteru s nejmenším aktuálním počtem aktivních spojení. Na podobném principu pracuje další jednoduchý algoritmus s názvem **Load-Based.** Jak napovídá název žádosti se zasílají na server s nejmenším aktuálním vytížením.

Jiné algoritmy užívají odezvu z jednotlivých strojů k tomu, aby určily, který stroj může nejlépe obsluhovat požadavky klienta. Dodatečně také mohou LB umožňovat zajišťování poruch, či výpadků jednotlivých serverů. LB sleduje server nebo bežící aplikaci a zastaví zasílání požadavků pokud dojde k výpadku.



Obr. 2: Princip obecného LB

2.1.3 Výpočetní cluster

Výpočetní cluster (computer cluster) je skupina úzce spolupracujících počítačů, které se navenek jeví jako jeden jediný, velmi výkonný počítač. Jednotlivé uzly jsou většinou

propojeny pomocí rychlé lokální sítě (LAN). Clustery obvykle dosahují lepší dostupnosti v kombinaci s vysokým výkonem, zatímco pořizovací náklady jsou nižší než u superpočítačů.

K vytvoření výpočetního clusteru se používá OSS, který bude popisován dále. Je to např. Beowulf, OpenMosix atd.

2.1.4 Úložný cluster

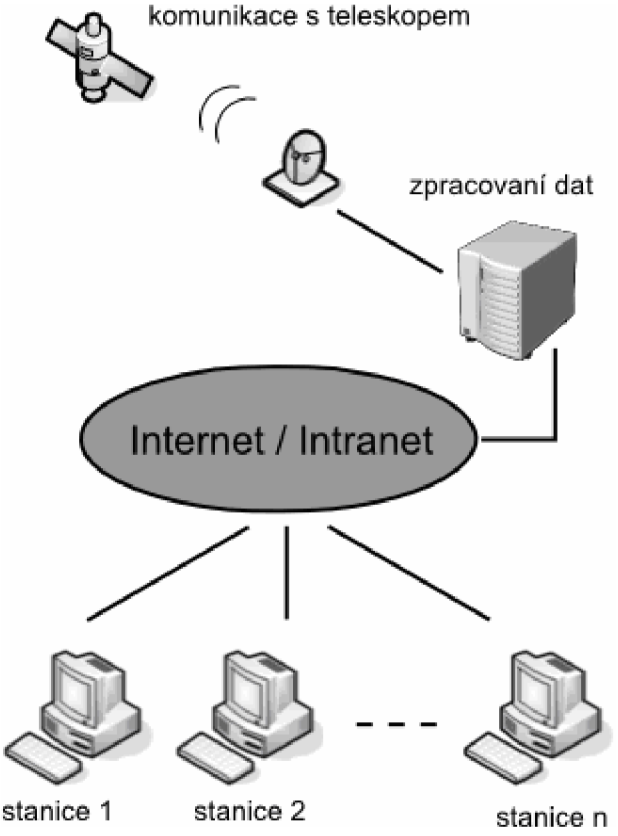
Úložný cluster (storage cluster) umožňuje přístup k diskové kapacitě, která je rozložena mezi více počítačů. Důvodem je zajištění vyšší spolehlivosti nebo dosažení vyššího výkonu. K tomu se používají speciální souborové systémy (např. GFS, OCSF2 ...), které zajišťují rozložení a duplikaci dat, pokrytí výpadků jednotlivých uzlů, mechanismus zamykání souborů a další podpůrné služby. Důležité také je, zajistit synchronizaci dat mezi uzly, aby nedošlo k jejich ztrátě či poškození. Synchronizace je však rozdílná pro data se kterými se pracuje málo (statická), a pro data které používáme téměř pořád.

2.1.5 Gridový cluster

Rozdíl mezi clusterem a gridovým clusterem je v rozsahu a počtu připojených uzlů. Zatímco v clusteru bývají spojeny pomocí LAN stovky stejnorodých počítačů, u gridu tvoří spojení síť typu WAN nebo internet s připojením sta tisíců i více počítačů. Gridovým clusterům se dostává v poslední době hodně pozornosti ze strany vědců, za kombinování výpočetního výkonu a dosahovaných výsledků na poli výzkumu. Pokud bychom si chtěli založit vlastní grid, musíme řešit problémy spojené s oprávněním, bezpečností celého projektu a v neposlední řadě také otázky finanční. Gridové clustery dosahují mnohem většího výpočetních výkonu než jednotlivé clustery s využitím nestejnorodých hardwarových a softwarových prostředků. Do gridu je možné zapojovat i různé clustery

Nejznámější a nejlépe dokumentovány gridový projekt SETI@home [4], který slouží k analýze dat z hubblova teleskopu, za účelem hledání výzkumu mimozemské civilizace. Dobrovolníci si do vlastního PC připojeného k internetu nainstalují software nezávislý na platformě OS, pomocí kterého se připojí do gridového clusteru. Software pracuje při nečinnosti PC – při aktivním spořiči obrazovky. Existují různé druhy software pro analýzu a

výzkum např. lékařských problémů jako výzkum rakoviny, AIDS, či peer-to-peer sdílení dat (Napster, Kazaa). Gridové clustery zpracovávají určité dávky dat, které v době výpočtu nejsou kontrolovány. Kdežto u homogenních clusterů se tato kontrola provádí, čímž se zvyšuje efektivita výpočtu.



Obr. 3: Naznačení funkce gridového clusteru

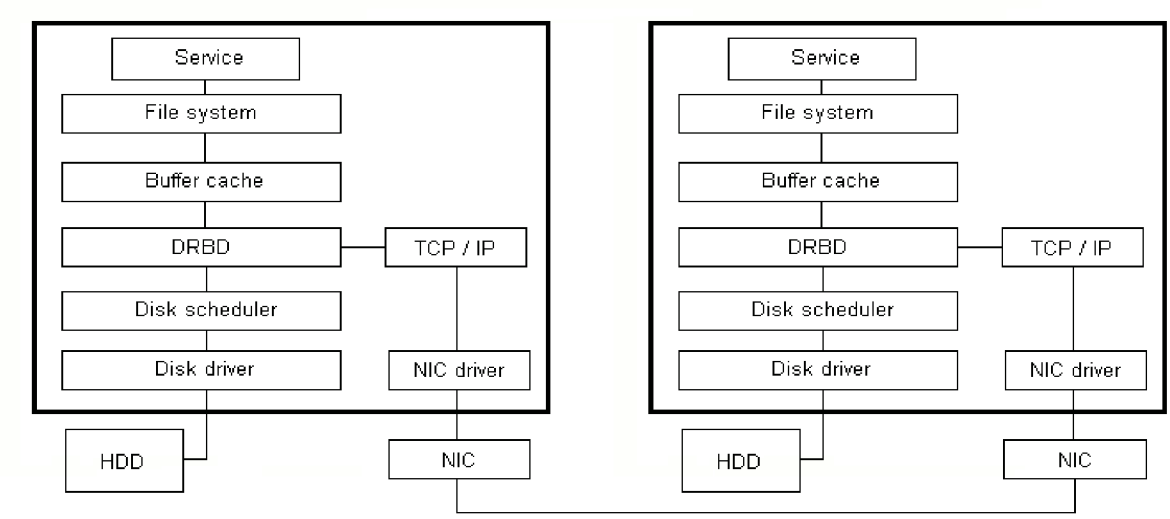
3 Popis Open Source software

OSS je takový počítačový software, u kterého je veřejně dostupný zdrojový kód. Tedy není zde nutnost vynaložit finanční prostředky na zakoupení. Tento software je možné za dodržení určitých podmínek volně využívat, šířit a dále upravovat.

3.1 Popis Open Source řešení pro úložný cluster

3.1.1 DRBD (Distributed Replicated Block Device)

Jedná se o řešení problematiky replikace dat mezi dvěma servery zapojenými do HA clusteru. Na obrázku 1223 vidíme z čeho se nástroj DRBD skládá. Především jsou zde znázorněny dva servery tvořící HA cluster. Každý z nich obsahuje části Linuxového jádra. Souborový systém, vyrovnávací paměť cache, plánovač disku, diskové ovladače, TCP / IP zásobník a řadič síťové karty (NIC). Na DRBD můžeme nahlížet jako na síťovou obdobu RAID1, pole rozloženého mezi dva servery. DRBD velice dobře spolupracuje s projektem Linux-HA ale i s dalšími distribucemi. Nad blokovým zařízením DRBD se pak používají souborové systémy (EXT2, EXT3, XFS, JFS, ...). Protože se jedná o standardní FS, které nejsou upraveny pro paralelní výpočetní systémy, může být pouze jeden uzel ve stavu PRIMARY(probíhá zápis dat) a druhý uzel ve stavu SECONDARY (zápis zrcadlených dat). V DRBD verze 8 je možné provozovat režim Active-Active za použití sdíleného filesystému (GFS, OCSF2). Aktuální verze na domovských stránkách je Drbd-8.2.6 z května 2008.



Obr. 4: Jednotlivé bloky DRDB

3.1.2Lustre

GNU GPL softwarové řešení od firmy SUN Microsystems je škálovatelným, robustním clusterem s vysokou dostupností k souborovému systému s názvem Lustre[3] [5]. Hlavním cílem je vytváření nové generace úložných clusterů, které mohou obsluhovat uskupení složené z 10.000 uzlů. Tzv. next-clustery by měly poskytovat petabajty úložného prostoru, data by se měli přenášet rychlostmi 100 GB/sec s technikou state-of-the-art security (nejvyšší úroveň vývoje zabezpečení) a správou infrastruktury.

Lustre se používá na několika největších linuxových clusterech na světě a je také jako klíčový prvek distribuován se softwarovým vybavení clusterů světových značek. (HP SFS, Cray XT3, XD1 supercomputers). Současná dostupná verze na stránkách výrobce[4] je Lustre 1.6.6[5].

3.1.3SAN (Storage arrea network)

Volně přeloženo, jedná se o síťový odkládací prostor. Architektura SAN byla navržena k tomu, aby umožnila připojit vzdálené paměťové médium (např. diskové pole) na vzdáleném serveru. V operačním systému se pak zařízení tváří jako připojené lokálně. Velké úložné

[3] dostupné na internetu http://www.lustre.org/
[4] možnost stažení projektu http://www.sun.com/software/products/lustre/get.jsp
[5] listopad 2008

clustery používají pro přenos dat mezi disky a serverem protokol SCSI. Důvodů je hned několik.

Ve srovnání se standardními protokoly IDE/ATA či SATA je možné připojení většího počtu zařízení (podle daného protokolu SCSI až tisíce), sběrnice dosahuje vyšších přenosových rychlostí a propustností (např. Ultra-640 SCSI – 640 MB/s), tím i celkového výkonu. SCSI disky měly a mají zpravidla větší otáčky ploten, kratší přístupovou dobu a díky zaměření i větší životnost.

U technologie iSCSI se nepoužívají nízko-úrovňové fyzické rozhraní (kabely...), protože používaná sběrnicová topologie není vhodná pro připojení do sítě. Protokol iSCSI umožňuje zasílat SCSI příkazy na vzdálené servery a ovládat disky, které se tváří jako lokálně připojené. Kromě iSCSI (internet SCSI), která mapuje disky pomocí dostupné síťové technologie, se používají pro tvorbu SAN následující technologie:

- ATA over Ethernet (AoE) – Jedná se o síťový protokol, navržený pro jednoduchý ale vysoce efektivní přístup k PATA či SATA diskům pomocí sítě ethernet. Umožňuje použít standardní technologii a tím docílit nízkých nákladů. AoE není přímo závislý na vrstvách ethernetu (IP, UDP, TCP), což ho činí srovnatelným s iSCSI. V důsledku této nezávislosti však nedokáže sít routovat proto je použitelný pouze pro malé sítě SAN.
- Fibre Channel (FC) – Je to technologie využívající gigabitovou síť, původně určena pro propojení superpočítačů. Postupem času se z ní stala standardní technologie, která využívá Fibre Chanel Protokol (FCP) pro tvorbu SAN. FCP je protokol připomínající TCP, a je určen pro přenos SCSI příkazů skrze FC sítě.
- FICON (Fibre CONectivity) – Technologie vytvořena IBM, umožňuje mapování disků přes FC, čehož využívají robustní výpočetní clustery. Přenosové rychlosti jsou 1,2 a 4 Gbps do vzdálenosti až 100 km.
- HyperSCSI – Síťový protokol umožňující odesílat a přijímat SCSI příkazy. Na rozdíl od iSCSI, obchází IP vrstvy a pracuje přímo s vrstvou ethernetu. Tímto se vyhýba segmentaci a skládání paketů potřebných pro IP.Ve srovnání s iSCSI tak získává na výkonu za cenu ztráty IP flexibility.
- iSCSI rozšíření pro RDMA (iSER), mapuje iSCSI přes InfiniBand (IB)

- iFCP nebo “SANoIP“ mapující SCSI přes FCP na sítich s IP.

Bližší popis o technologii SAN můžeme nalézt v [6].

3.2 Popis Open Source řešení pro výpočetní cluster

3.2.1 Oscar, Rocks

OSCAR [7] jedná se o HPC (High Performance Computing) cluster, který obsahuje aplikace pro správu. Celý systém můžeme spustit a dále provozovat bez pracné instalace a konfigurace, díky systému balíků s přítomností množství předchystaných aplikací a utilit. Oscar se instaluje jako rozšíření různých linuxových distribucí s individuálním výběrem balíků důležitých pro uživatele. Dále si systém upraví data na discích uzlů v clusteru, tak aby bylo možné využívat administrativní nástroje a klientský software. Výhodou Oscar HPC je obsah software pro zátěžové testy, které odhalí slabá místa v návrhu clusteru.

Ve výchozím nastavení využívá OSCAR některé MPI (Message Passing Interface), pro vědecké výpočty. Běžně využívané MPI jsou součást základního OSCAR package systému. Výhodná vlastnost systému je možnost instalace více implementací MPI na jednom clusteru a následně umožňuje jejich přepínání. Do budoucna vývojáři zařadí do OSCAR package systém další typy aplikací jako LB, HA a web clustering.

ROCKS [8] je OSS určená pro linuxové distribuce, která umožňuje koncovým uživatelům vybudovat výpočetní cluster nebo koncové uzly gridu. Od května roku 2000 se projekt rock zaměřil na problematiku spojenou se sestavením clusteru s cílem maximálně tento úkon zjednodušit. Jedná se o instalaci, správu, upgrade a škálovatelnost. Cílem projektu je tedy umožnit použití rock clusteru širokou veřejností a především vědci. Stabilní a snadno nastavovatelné PCS (parallel computing systems) pomohou ve všech oblastech výzkumu.

Informace o systémech OSCAR a ROCK je možno nalézt v literatuře [1].

3.2.2 Beowulf

Pro mnoho lidí je Beowulf [9] synonymem pro výpočetní cluster. Název pochází z anglické literatury, kde Beowulf je jméno epického hrdiny popisovaného „mající sílu mnoha“. Poprvé použili název vědci pracující pro NASA v roce 1994, jejichž úkolem bylo sestavit

paralelní počítač za použití běžného hardwaru v kombinaci s volně dostupným softwarem. Sestavili tedy výpočetní cluster s názvem Beowulf, který byl v té době nejlepším.

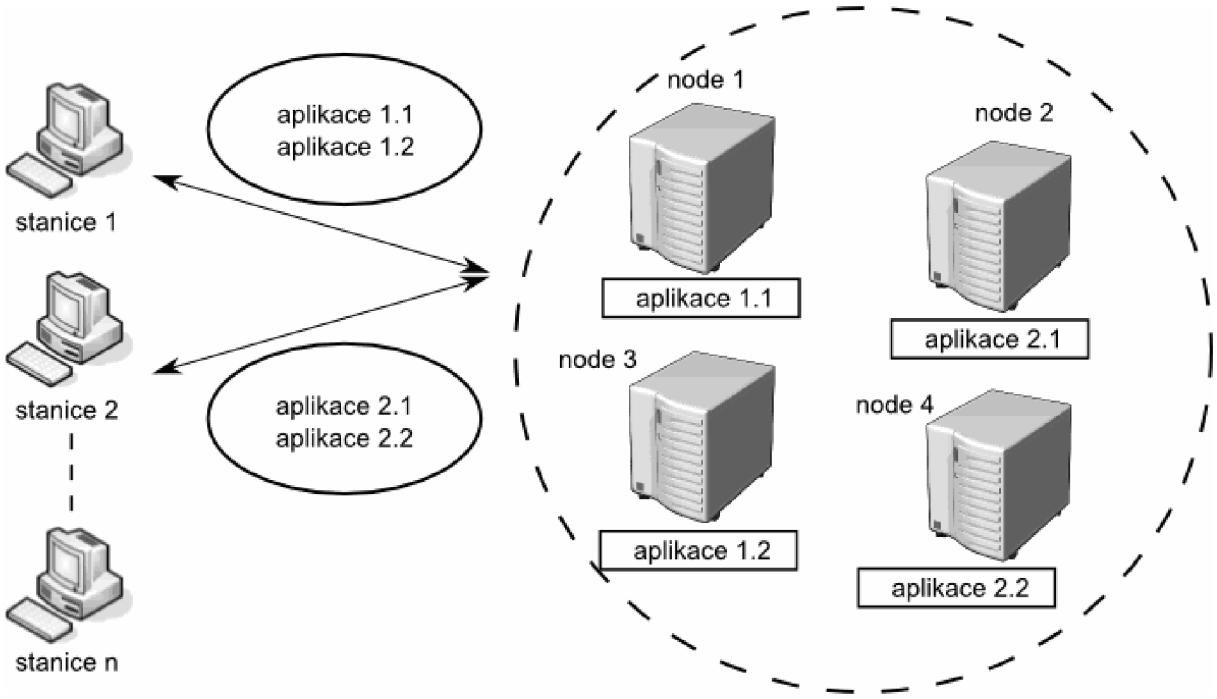
Pro vytvoření Beowulf clusteru je možné použít zastaralý hardware, který už nemá jiné využití. Jedná se o takzvaný COST (computer of the shelf), POP (pile of PCs) nebo cost-off komponenty. Takto poskládaný cluster je velmi levný konkurent superpočítače. Na druhou stranu je lepší pro stavbu použít stejnorodé prvky, výkonné uzly, které jsou vybrány pro daný účel clusteru. Pokud se jedná o klasické počítače můžeme je ke clusteru dynamicky připojovat či odpojovat. Naopak, pokud tuto možnost nepotřebujeme, zmenšují se nároky na hardwarové vybavení jednotlivých uzlů. Stačí přepínat sdílené komponenty jako monitor, myš a klávesnice. Počítače v clusteru dokonce nemusí mít ani pevný disk.

3.2.3 Open Mosix

OpenMosix [10] vznikl v roce 2002 jako volná odnož projektu MOSIX (Multicomputer Operating System for UNIX) v době kdy se MOSIX vzdaloval od GPL licence. Autorem projektu OpenMosix je Moshe Bar, člen původního vývojářského týmu MOSIXu. OpenMosix je software, který rozšiřuje Linuxové jádro tak, že se procesy mohou stěhovat mezi různými stroji uvnitř clusteru tak, aby bylo možné rovnoměrněji rozložit zátěž a využít výkon uzlů. Přesuny probíhají transparentně a uživatel je vůbec nezaznamená.

Například pokud je potřeba zpracovat více operaci (viz. obrázek 5) náročných na výpočetní výkon procesoru (např. komprese videa), umožní nám OpenMosix zpracovávat všechny operace najednou. Na hlavním uzlu bude probíhat jedna operace komprese a ostatní se podle určitých pravidel rozmístí na další uzly v clusteru. Po skončení úkolů se výsledky opět vrátí na hlavní uzel. Náročné operace tedy zabírají jen o málo více času něž jedna úloha spuštěná na hlavním uzlu.

Od 1. března 2008 je vývoj projektu OpenMoxis zastaven. Důvodem ukončení vývoje je vysoká dostupnost levných a výkonných více jádrových procesorů a tudíž clusterování postrádá smysl. Projekt je stabilní s Linuxovým jádrem 2.4.x pro x86 architektury. Portování do jádra 2.6 se zastavilo v testovacím stádiu alfa stage, kde je podpora pro 64-bit technologii AMD64. Zdrojové kódy projektu je možné si stáhnout ze stránek SourceForge.



Obr. 5: Princip funkce OpenMosix

3.2.4 OpenMPI

Jedná se o Open Source projekt, který využívá volné implemetace MPI-1 a MPI-2 (modely pro předávání zpráv). OpenMPI [11] software tím dosahuje vysokého výkonu. Od začátku 90. let minulého století existovaly různé realizace MPI např.: systémy předávání zpráv jako produkty jednotlivých výrobců nebo vývojářských skupin, které však byly avzájem nekompatibilní. Z toho důvodu vznikla potřeba vytvořit standard pro více-procesorové systémy.

V roce 1992 vznikla standardizační pracovní skupina, která v listopadu téhož roku zasadila o vznik tzv. MPI fóra[6] a nově vyvíjeném standardu začalo pracovat mnoho odborníků z významných organizací, oblasti výzkum paralelních počítačů, vývoje software a vědců z aplikační oblasti. Po roce byl vydán první návrh, který v dubnu 1994 dal vzniknout oficiálnímu standardu MPI verze 1.0. Později byl pak byli vydány verze 1.1 (1995) a 1.2 (1996).

Hlavní cíle projektu MPI byly:

[6] MPI fórum http://www.mpi-forum.org/

- zajistit přenositelnost paralelních aplikací na úrovních zdrojového kódu
- umožnit efektivní implementaci modelu předávání zpráv

Další, doplňující požadavky:

- podpora heterogenních paralelních architektur
- funkce nezávislá na programovacím jazyku
- jazykové rozhraní pro C/C++ a Fortran
- podobnost chování s existujícími nástroji

Velké změny pak přinesl standard MPI-2 z července 1996, s množstvím nových funkcí. Navíc tedy MPI-2 obsahuje :

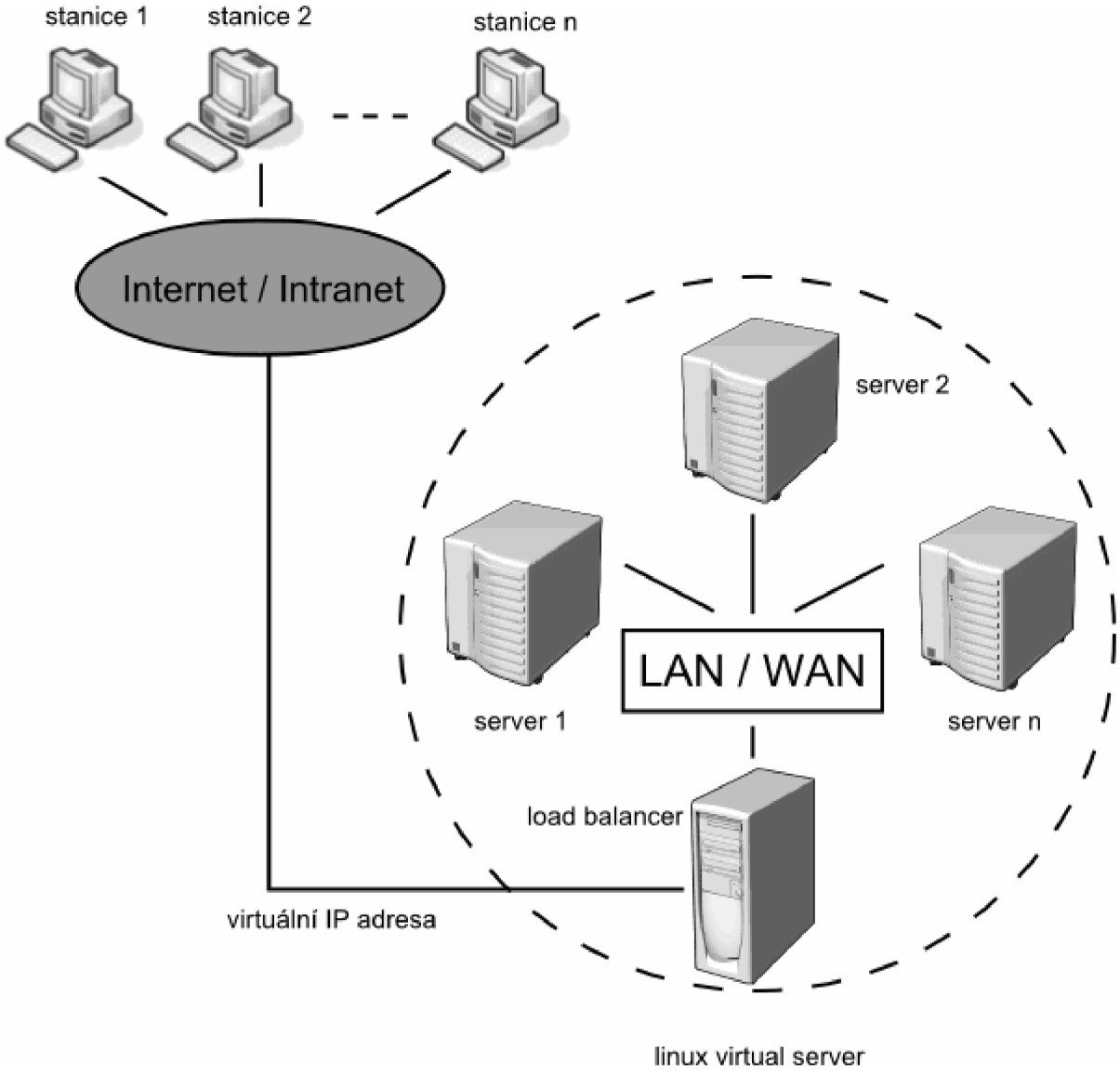
- dynamické procesy
- jednostrannou komunikaci (remote memory access, RMA)
- neblokující verze kolektivních komunikací
- komunikaci mezi komunikátory (inter-comm)
- podporu I/O, která v MPI-1 zcela chybí
- podporu zpracování v reálném čase
- podporu zpracování stylem klient/server

Poslední verzí z roku 2007 je MPI 2.1, kterou dobře přijali uživatelé, programátoři i výrobci paralelních počítačů. MPI-1 jen v současné době implementován prakticky ve všech víceprocesorových platformách. MPI je zástupcem explicitní paralelizace, kdy je plně na programátorovi, aby detekoval paralelismus v algoritmu a implementoval ho prostřednictvím konstruktů MPI v jeho kódu. Pomocí MPI jsou programovány všechny programy pro největší superpočítače na světě. MPI je také široce používán s jazyky Python, Pearl a Java. MPI je implementováno také v LAM / MPI, které poskytuje knihovny jazyka C a Fortran. To umožňuje psát paralelní programy s využitím MPI s využití funkcí knihoven daného jazyka.

3.3 Popis Open Source řešení pro rozložení zátěže

3.3.1 Linux Virtual Server (LVS)

LVS [12] je nástroj pro rozložení zátěže. Pomocí LVS jsme schopní vytvořit virtuální server (viz obr. 6), který se skládá z několika reálných serverů spojených do clusteru.



Obr. 6: Zapojeni LB při použití LVS

Servery jsou spojeny pomocí vysokorychlostní LAN či rozsáhlejší rozptýlené WAN. Hlavní částí je load balancer. Ten rozděluje vstupní žádosti jednotlivým serverům a zároveň umožňuje vytvářet paralelní služby clusteru tak, aby se jevili jako virtuální – běžící na jedné IP adrese. Architektura clusteru je pro koncového uživatele transparentní, tudíž uživatelé vidí jen jediný virtuální server. Dobrá rozšiřitelnost virtuálního serveru je zajištěna jednoduchým přidáváním či odebíráním serverů z clusteru. Vysokou dostupnost je zajištěna detekčním uzlem nebo démonem (definice) kontrolující selhání. Ten je v případě nutnosti schopen vhodně překonfigurovat systém. LVS tedy poskytuje výkonný linuxový server s vysokou dostupností založený na clusterování, které kombinuje dobrou rozšiřitelnost, spolehlivost a užitečnost.

3.4 Popis Open Source řešení pro vysokou dostupnost

3.4.1 Linux-HA

Jedná se projekt vzniklý v roce 1999, zabývající se řešením vysoké dostupnosti výpočetních systémů. Linux-HA není závislý na platformě operačního systému a je možné jej provozovat pod různými linuxovými distribucemi (Debian, Gentoo, Red Hat), také pod UNIX-based OS jako je FreeDSB, Solaris, OpenDSB. Autoři uvádí, že je možné Linux-HA zprovoznit dokonce na systémech MacOS/X. Hlavní modul Linux-HA se nazývá Heartbeat, který můžeme najít v podobě balíčku ve většině linuxových distribuci.

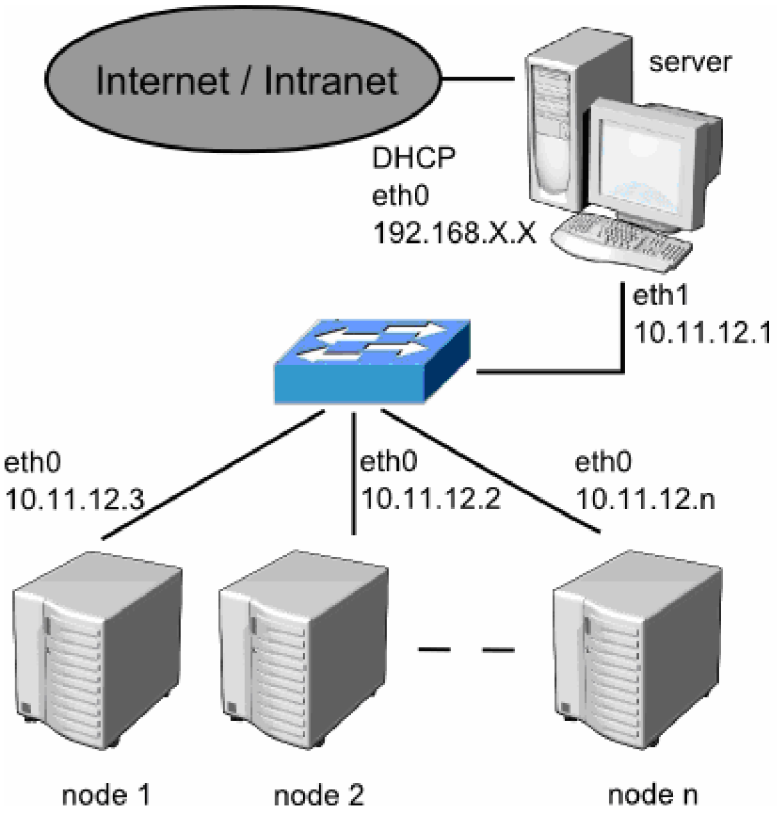
V současnosti stabilní verze Linux-HA podporuje služby dostupné pro clustery složené z N uzlů typicky jsou to:

- Databázové servery
- ERP aplikace
- Webové servery
- LVS (load balancer)
- Poštovní servery
- Firewally
- Databázové servery
- DNS servery
- DHCP servery
- Proxy Caching servery
- Uživatelské aplikace
- atd.

Heartbeat je použitelný prakticky v každém tržním segmentu, průmyslu a organizační velikosti.

4 Sestavení clusteru

Pro účel testování výkonu výpočetního clusteru byl na vybraný počítač nainstalován operační systém debian GNU/Linux 5.0[7]. Tento počítač slouží jako frontnode, na kterém běží služby dostupné připojeným uzlům. Pro jednoduchost byla snaha nastavit tento server tak aby mohly připojené koncové uzly nabootovat ze serveru operační systém a to jen do operační paměti tak jako je to u linuxových live distribuci. Koncové uzly tedy mohou být bezdiskové.



Obr. 7: Zapojení clusteru

Front node má k dispozici procesor Intel Core 2 Duo @ 1.866 GHz, 1 GB RAM, HDD 40GB, 2x síťová karta podporující 10/100Mbps Ethernet LAN.

[7] www.debian.org

Pevný disk byl rozdělen následovně :

```
Velikost: 28GB ext3, Label: root, bootable
Velikost: 10GB, ext3, Label: nodes, bootable
Velikost: 2GB, swap
```

Na první partition byl instalován OS debian GNU/Linux x86_am64, na druhou partition určenou jako systém pro připojované uzly OS debian GNU/Linux x86. Uzly mají 32b OS z důvodu možnosti testování ve virtuálních strojích, které 64b jádro nespustí.

Po instalaci základního 64b systému bylo potřeba nastavit dostupná síťová rozhraní, aby bylo možné doinstalovat potřebné balíky ze síťových zrcadel. Toto provedeme editací souboru `/etc/network/interfaces`

```
auto eth0
iface eth0 inet ducl
auto eth1

iface eth1 inet static
address 10.11.12.1
netmask 255.255.255.0
broadcast 10.11.12.255
```

Po následném update zdrojů příkazem `apt-get update` je nutné instalovat balíky pro podporu MPI, kompilátory a překladače služby potřebné pro start bezdiskových stanic.

```
apt-get install rsh-server rsh-client openssh-server
```

```
apt-get install dhcp-server tftpd-hpa nfs-kernel-server

apt-get install libatlas-headers libatlas3gf-base python2.5
python-central python-dev

apt-get install openmpi-dev openmpi-bin openmpi-common lam4-
dev lam-runtime

apt-get install gfortran build-essential fping htop
```

Instalací ssh a rsh jsme získali vzdálenou správu systému a šifrovanou komunikaci mezi připojenými stanicemi. Na frontnode jsme nainstalovali potřebné služby DHCP, NFS, TFTP k nastartování bezdiskových stanic. Dále pro je nutné nainstalovat knihovny ATLAS[8] pro generování funkcí lineární algebry, který je nutný pro zprovoznění HPL. Knihovny jazyka python jsou nutné pro funkci rozhraní MPI, které je instalováno z repositářů v podobě openMPI a LAM. Nakonec jsou instalovány překladače gcc make gfortran pro kompilaci zdrojových balíků a utility fping a htop.

Instalace stejných balíků byla provedena pro OS určený pro uzly. Pomocí příkazu `chroot /nodes` se přihlásíme do systému a můžeme instalovat.

V systému pro uzly je nutné dále nastavit připojování `/home` pomocí NFS (network file system), tak aby byli síleny data a změny se projevily pro všechny připojené uzly. Je nutné editovat soubor `/etc/fstab`

```
proc                /proc     proc          defaults  0  0
10.11.12.1:/home    /home     nfs     auto  defaults  0  0
```

Adresář `/home` na serveru s IP 10.11.12.1 je připojen uzlům a nahrazuje tedy jejích lokální domovskou složku.

Další krok je úprava souboru `/etc/initramfs-tools/initramfs.conf` a inicializace ramdisku. Změníme pouze řádek `BOOT=local` a provedeme update ramdisku.

[8] Možnost stažení http://www.netlib.org/atlas/

```
BOOT=nfs
update-initramfs -u
```

Tímto je nastaven OS pro uzly a pokračujeme v nastavování OS na front node. Pro nabootování ze sítě je nutné nastavit služby serveru DHCP, TFTP, NFS. BOOTP (Bootstrap Protocol). Slouží k nastavování síťových parametrů s pomocí serveru bez zásahu uživatele. To umožňuje centralizovanou správu síťových adres, bez nutnosti vytvářet na každém počítači v síti konfigurační soubor. DHCP (Dynamic Host Configuration Protocol) je založený na BOOTP, který rozšiřuje o další možnosti a je s BOOTP zpětně kompatibilní.

Konfigurace DHCP serveru se provádí v `/etc/dhcp3/dhcpd.conf`

```
allow booting ;
allow bootp ;
default-lease-time 600;
max-lease-time 7200;
subnet 10.11.12.0 netmask 255.255.255.0
{next-server 10.11.12.1;
filename " pxelinux .0";
option subnet-mask 255.255.255.0;
range 10.11.12.2 10.11.12.50;}
```

Počítačům bude tedy přidělena IP z výše definovaného rozsahu s danou maskou z uzlu. Počítače následně hledají soubor pxelinux.0 který je umístěn na TFTP serveru s IP definovanou na řádku next-server.

K přenosu operačního systému se používá TFTP (Trivial File Transfer Protocol). TFTP je protokol pro přenos souborů jako FTP, ale proti FTP je hodně zjednodušený. Neobsahuje autentizaci a běží nad UDP. Kořenový systém souborů bude připojen pomocí NFS. Nastavení

FTFP serveru se provádí `/etc/default/tftpd-hpa`, nastavením adresáře s jádrem zaváděného systému a povolením spuštění daemona.

```
RUN_DAEMON =" yes"
OPTIONS ="-l -s  tftpboot"
```

Je nutné vytvořit adresář `/tftpboot` do kterého jsou nakopírovány soubory jádra a ramdisku pro zavedení.

Na konec je nutné nastavit exportované adresáře pro NFS (Network File System), konfiguraci souboru `/etc/exports`:

```
/nodes  10.11.12.0/24(rw, no_root_squash, no_subtree_check)
/home   10.11.12.0/24(rw, no_root_squash , no_subtree_check)
```

Na uzlu frontnode pak spustíme dané služby jejich restartováním.

```
/etc/init.d/networking restart
/etc/init.d/dhcp3-server restart
/etc/init.d/tftpd-hpa restart
/etc/init.d/nfs-kernel-server restart
```

Tímto je server nastaven a uzly mohou bootovat pomocí PXE do chrootovaného systému umístěného na disku frontnode.

5 Praktické testy

Na stránkách Top 500 [3] můžeme vidět aktuální seznamy jak nejvýkonnějších superpočítačů, tak 500 nejvýkonnějších clusterů na světě. Pro měření výkonu se používá HPL.

5.1 High performance linpack – HPL

Jedná se o OSS benchmark postavený na základě algoritmu linpack [13]. Linpack napsaný v jazyku Fortran [14], pracuje s rutinami lineární algebry tak, aby vypočítal faktorizace matice, řeší lineárních systémy různých druhů metodou nejmenších čtverců a další běžné operace lineární algebry. HPL je verze linpacku, která využívá rozhranní MPI k řešení rozměrného systému lineárních rovnic s použitím dvojité aritmetické přesnosti (64b). HPL [15] je určen pro testování počítačů s rozloženým výpočetním výkonem tj.clusterů. Linpack je také napsaný v jazyce java[9].

Balík HPL umožňuje testovat a měřit čas vykonávání programu ke stanovení přesnosti dosaženého výsledku a doby výpočtu. Z těchto výsledků pak určuje celkový výkon systému. Naměřený výkon závisí na mnoha různých faktorech, nicméně tyto výsledky daného algoritmu se dají srovnávat mezi různými systémy.

Program HPL potřebuje pro svoji funkci v operačním systému dostupnou implementaci MPI, dále implementaci BLAS[10] (Basic Linear Algebra Subprograms) nebo alternativně ATLAS[11] (Automatically Tuned Linear Algebra Software) a VSIPL[12] (Vector Signal Image Processing Library). Tento software i ostatní balíky je dostupný pro velké množství operačních systémů.

[9] http://www.netlib.org/benchmark/linpackjava
[10] http://www.netlib.org/blas/
[11] http://math-atlas.sourceforge.net/
[12] http://www.vsipl.org/

5.1.1 HPL Algoritmus

HPL obsahuje ve skutečnosti mnoho možných nastavení pro různé operace. Možnosti se mohou měnit dokonce během výpočtu. Požadavky na výkon se mohou lišit a proto je možné experimentálně zjistit optimální nastavení pro každý použitý stroj. Z hlediska numerické přesnosti je možné říct, že kombinace výpočtů jsou navzájem ekvivalentní, i když se výsledky mohou mírně lišit v závislosti na bitových operacích (AND, OR, XOR, NOT). Pokud se pomocí bitových operací provádí násobení nebo dělení je doba provádění s pomocí bitových operací výrazně kratší.

5.1.2 Hlavní Algoritmus

Softwarový balík řeší systém lineárních rovnic v pořadí n: A*x = b.

Nejprve dochází k výpočtu LU faktorizace. Kdy se matice A rozloží na:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ I_{21} & 1 & 0 \\ I_{31} & I_{32} & 1 \end{pmatrix} \tag{1.0}$$

(lower) dolní triangulární matici a

$$U = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} \tag{1.1}$$

(upper) horní triangulární matici.

Pomocí částečné řádkové úpravy (row partial pivoting) n*(n+1) dostáváme koeficient matice.

$$\left[A * \vec{b}\right] = \left[\left[L, U\right] * \vec{y}\right] \tag{1.2}$$

Dochází k posunu řádků tak aby na hlavní diagonále byli největší čísla. Tato úprava je aplikována pouze na matici U, matice L zůstává beze změn a nevrací žádný výsledek.

1. Krok: z rovnice

$$L * \vec{y} = \vec{b} \tag{1.3}$$

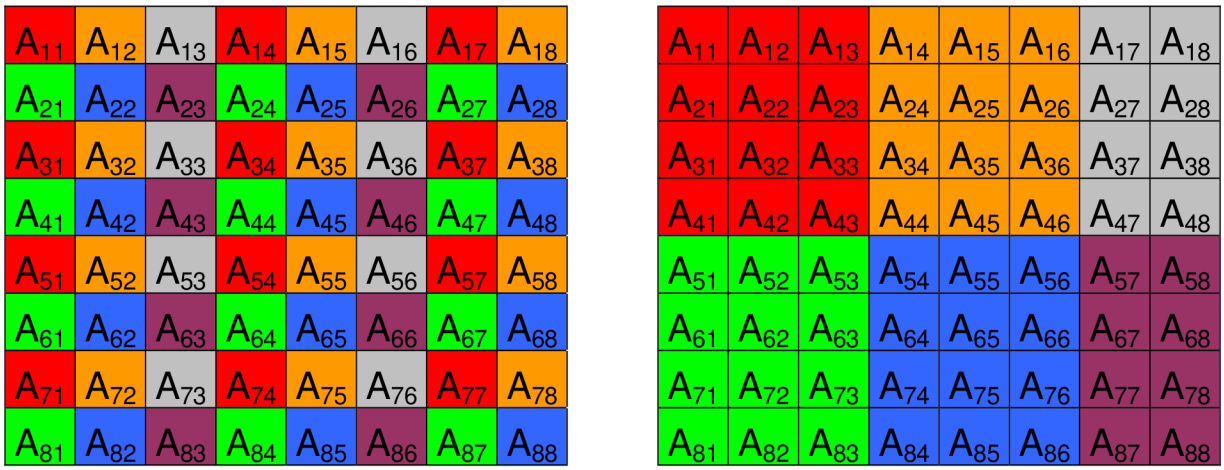
(známe $\vec{b}$) získáme úpravami tvar $\vec{y}$.

2. Krok: z rovnice

$$U * \vec{x} = \vec{y} \tag{1.4}$$

(známe $\vec{y}$) získáme úpravami tvar $\vec{x}$.

Data jsou distribuovány ve dvoj-rozměrné tabulce (mřížce) procesů s rozměry P x Q uspořádány podle blokově cyklického schématu k zajištění co nejlepšího rozdělení zátěže mezi uzly a škálovatelnosti algoritmu. Koeficient matice n*(n+1) je nejprve logicky rozdělen na bloky o velikosti nb*nb které jsou cyklicky řešeny pomocí tabulky procesů P x Q. Řešení je prováděno v obou rozměrech matice. Obrázek 8 je převzat z [15].



Obr. 8: Mřížka procesů PxQ

Pro hlavní smyčku z LU faktorizace byla zvolena right-looking varianta. To znamená, že v každé iteraci smyčky je faktorizován sloupec nb z panelu a koncová submatice je

aktualizována. Všimněte si, že tento výpočet je tedy logicky rozdělených ve stejném bloku velikosti nb, který byl použit pro distribuci dat.

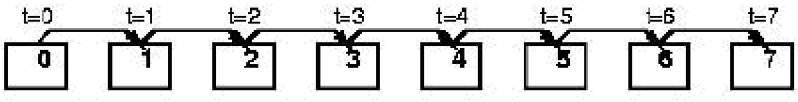
5.1.3 Faktorizace panelu

Na dané iteraci z hlavních smyčky, každý panel faktorizace se vyskytuje v jednom sloupci procesů, z důvodu vlastností kartézského systému. Tato část výpočtu je velmi důležitá součást celkového algoritmu. Uživatel má na výběr ze tří druhů rekurzívního násobení matic (Crout, left a right-looking). Software také umožňuje uživatelům vybrat si na kolik sub-panelů by měl být rozdělen zvolený panel a to v průběhu rekurze. Dále lze také za běhu vybrat kritérium pro zastavení rekurze, pokud zbývá dostatečný počet sloupců k faktorizaci. Pokud je tato hranice je dosaženo, sub-panel bude faktorizován pomocí jednoho ze tří zmíněných pravidel, založených na vektorově orientovaných maticích. Nakonec, pro každý sloupec panelu dochází k pivotovanému (otáčivému) vyhledávání a operace přehození a vysílání pivotních řádků jsou sloučeny do jednoho komunikačního kroku. Funkce binary-exchange (leave-on-all) redukuje tyto tři operace na jednu.

5.1.4 Popis vysílání

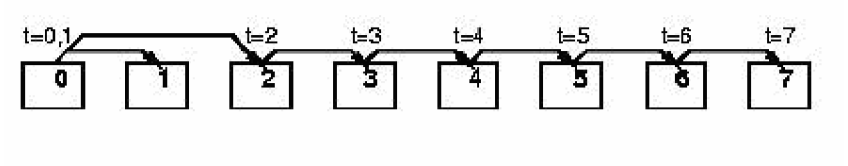
Jakmile je dopočítána faktorizace panelu je vypočítaný panel sloupců vysílán na druhý proces sloupců s využitím virtuální kruhové topologie. Existuje mnoho možných vysílacích algoritmů a software v současné době nabízí 6 variant, ze kterých si můžeme vybrat. Tyto varianty jsou popsány níže za předpokladu, že proces 0 je zdrojem vysílání. (význam "->" je "zašle"). Obrázky 9 – 15 jsou převzaty internetových stránek [15].

1) **Increasing-ring**: 0 -> 1; 1 -> 2; 2 -> 3 atd. Tento algoritmus je klasika ,nevýhodou je že proces 1 posílá zprávu.



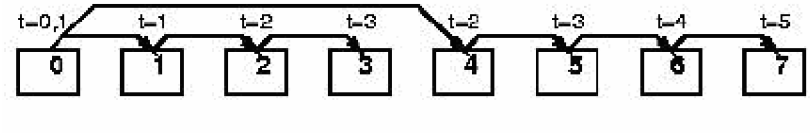
Obr. 9: Algoritmus Increasing-ring

2) **Increasing-ring (modified)**: 0 -> 1; 0 -> 2; 2 -> 3 atd. Proces 0 vysílá dvě zprávy a proces 1 jenom přijímá 1 zprávu. Tento algoritmus je většinou vždy lepší než předchozí ,ne-li nejlepší..



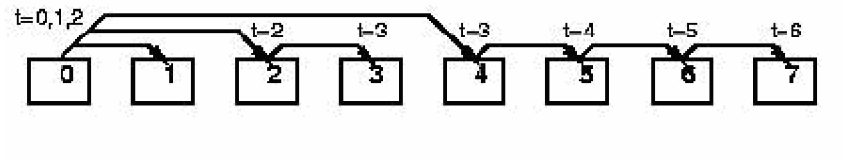
Obr. 10: Algoritmus Increasing-ring modified

3) **Increasing-2-ring**: Proces Q je rozdělen na dvě části : 0 -> 1 a 0 -> Q/2; Pak je proces 1 a Q/2 se stane zdrojem dvou kruhů. Předávání informace je 1 -> 2, Q/2 -> Q/2+1; 2 -> 3, Q/2+1 -> to Q/2+2 atd. Tento algoritmus má výhodu redukce času, za kterým poslední proces dostane panel za cenu že proces 0 vysílá 2 zprávy:



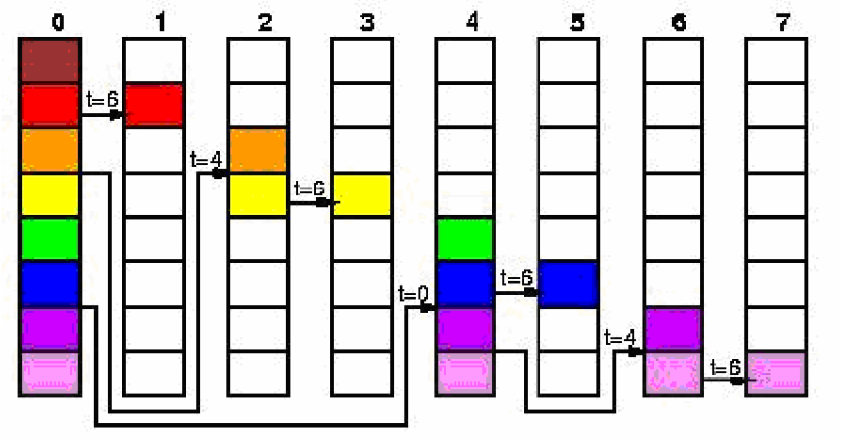
Obr. 11: Algoritmus Increasing-2-ring

4) **Increasing-2-ring (modified)**: Nejdříve proces 0 -> 1, dále zbylé procesy Q-1 jsou rozděleny do dvou polovin: 0 -> 2 a 0 -> Q/2; Procesy 2 a Q/2 vystupují jako zdroje dvou kruhů: 2 -> 3, Q/2 -> Q/2+1; 3 -> 4, Q/2+1 -> to Q/2+2 atd. Tento algoritmus je pravděpodobně největším konkurentem varianty: increasing ring modified.



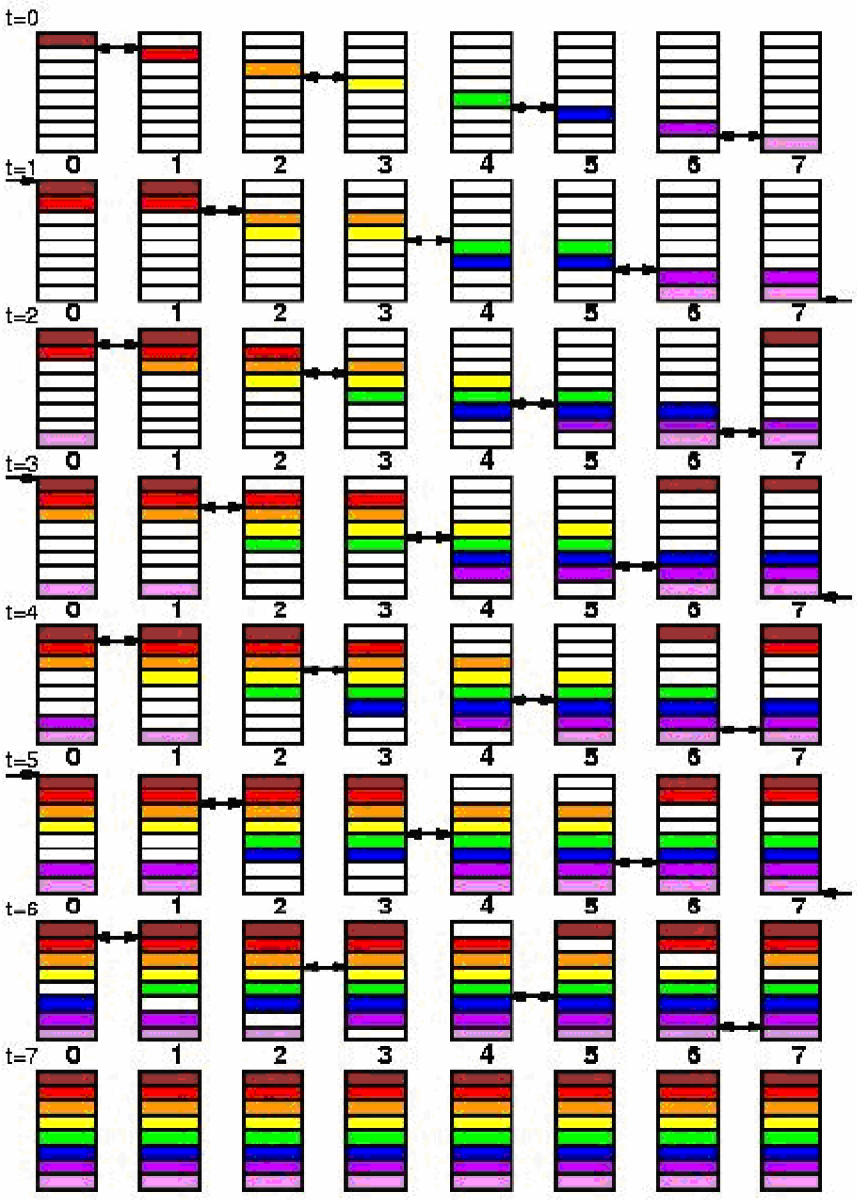
Obr. 12: Algoritmus Increasing-2-ring modified

5) **Long (bandwidth reducing)**: jako protiklad předchozího algoritmu, tato varianta synchronizuje všechny procesy. Zpráva je rozdělena na Q rovnocenných částí a rozptýlena do Q procesů.



Obr. 13: Algoritmus Long

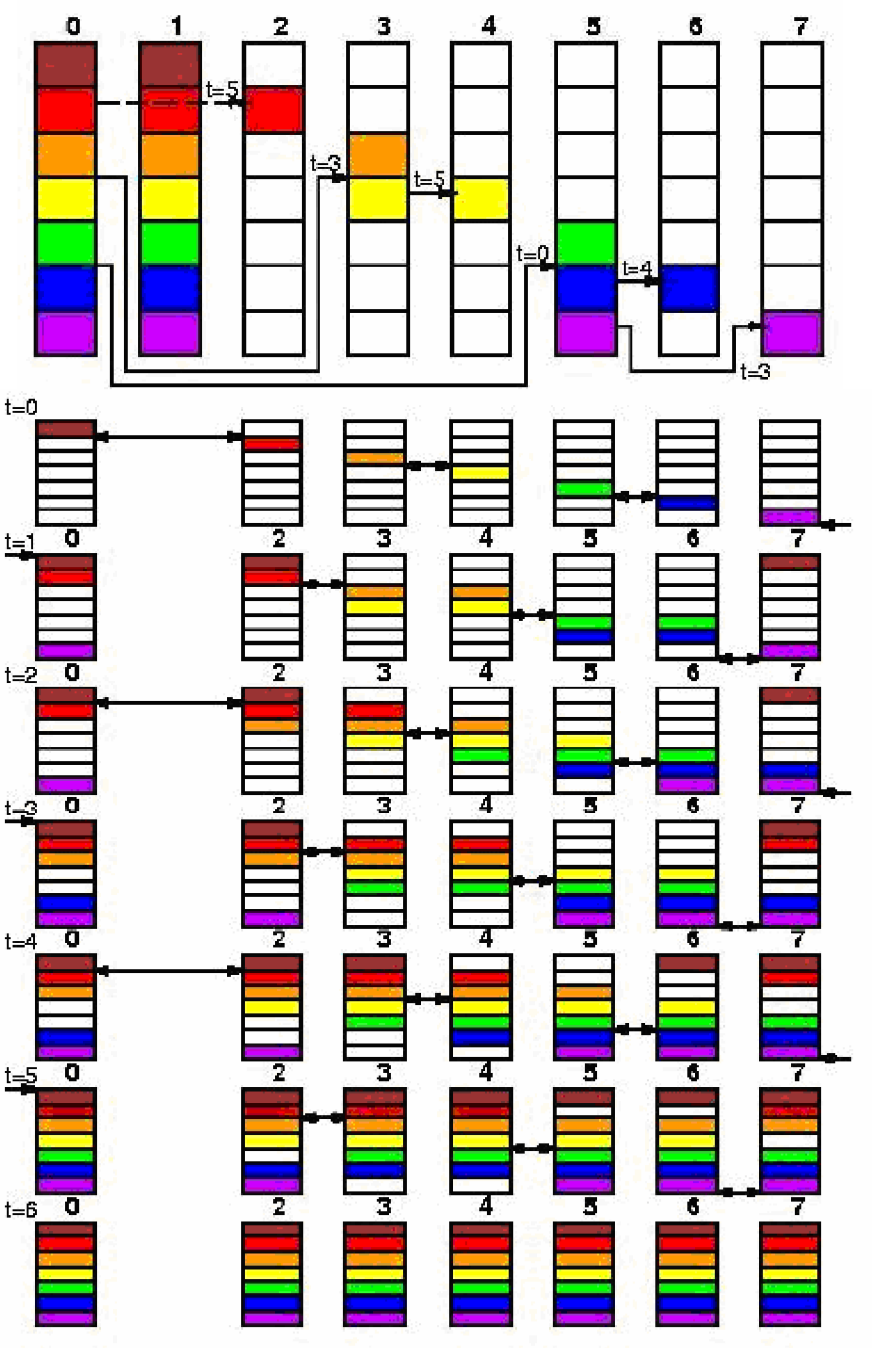
Časti jsou v Q-1 krocích přesunovány. Ve fázi rozptýlení se používá binární strom a ve fázi přesunu se výhradně užívá oboustranné výměny zpráv. V lichých krocích 0 <-> 1, 2 <-> 3, 4 <-> 5 atd.; v sudých krocích Q-1 <-> 0, 1 <-> 2, 3 <-> 4, 5 <-> 6 atd. Obrázek je převzat ze internetových stránek [15].



Obr. 14: Postupná výměna dat pro algoritmus Long

Je přeneseno více zpráv, ale celkové množství přenesených informací je nezávislé na počtu Q, což dělá tento algoritmus částečně vhodný pro velké zprávy. Algoritmus je tady využitelný pokud jsou uzly “velmi rychlé“ ve srovnání se sítí, která je “velmi pomalá“.

6) **Long (bandwidth reducing modified)**: stejný jako předchozí, postup nejprve 0 -> 1, dále je varianta Long použitá na procesu 0,2,3,4 .. Q-1.



Obr. 15: Postupná výměna dat při algoritmu Long modified

Vyjadřuje výměnu dat algoritmu Long modified.

Kruhové varianty se odlišují způsobem sondovacího mechanizmu, který je aktivuje. Jinými slovy, proces zahrnutý v broadcastu je odlišný od procesu zdroje a asynchronně zkoumá zprávy k přijetí. Pokud je zpráva dostupná je odvysílána a zároveň funkce vrátí potvrzení. Toto umožňuje prokládání broadcastových operací s fází aktualizace. Toto přispívá k redukci času nečinnosti když procesy čekají na faktorizovaný panel. Tento mechanizmus je nezbytný k pokrytí rozdílného poměru vypočet – komunikace.

5.1.5 Ověření vypočítaného výsledku

K ověření dosaženého výsledku je generovaná výstupní matice pravých stran. Jsou vypočtena tři rezidua (zbytky) a výsledek je uznán za numericky správný. Pokud jsou všechny hodnoty menší než prahové hodnoty v řádu 1.0.

pozn. Eps je relativní přesnost stroje pro distribuované systémy.

- ||Ax-b||_oo / (eps * ||A||_1 * N)
- ||Ax-b||_oo / (eps * ||A||_1 * ||x||_1)
- ||Ax-b||_oo / (eps * ||A||_oo * ||x||_oo)

5.1.6 Ladění parametrů HPL (tunning)

Po zkompilování překladu zdrojových kódů vzniká spustitelný soubor umístěný v adresáři `hpl/bin/<arch>/xhpl`. Pro dosažení co nejlepších výsledků je nutné správně nastavit vstupní data konfigurací souboru `HPL.dat`[13]. Tento soubor obsahuje parametry benchmarku jako: velikost problému N, konfiguraci strojů a další funkce algoritmu. Nastavené parametry se pak vypisují do výstupního souboru nebo na obrazovku. Uvádím použitý konfigurační soubor pro 1 PC (2 jádra CPU):

```
HPLinpack benchmark input file
```

[13] Podrobný popis a příklady konfigurace http://www.netlib.org/benchmark/hpl/tuning.html

```
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
10240         Ns
1            # of NBs
128           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
```

```
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Popis jednotlivých řádků použitého konfiguračního souboru HPL.dat

Line 1, 2: Textový popis slouží jako poznámky. Neovlivňuje chování benchmarku.

Line 3: Uživatel si může zvolit, jakým způsobem přesměrován výstup. Uvádí se zde název, do kterého je přesměrován výstup benchmarku.

Line 4: Uvádí se kam bude přesměrován výstup. Podle uvedeného čísla, program rozhodne zda bude směřován výstup na standardní výstup (6), standardní chybový výstup (7), či do souboru (8) jehož název je na řádku 3.

Line 5, 7, 10, 14, 16, 18, 20, 22, 24: Udává počet opakovaní testu s hodnotami na následujícím řádku

Line 6: Udává počet problému N_s (number of problem = problem size), které chceme spustit. Souvisí s řádkem 5. Abychom dosáhli změření co největšího výkonu snažíme se využít co nejvíce dostupné sdílené operační paměti. Ideálně asi 80-90%, kdy zbytek musíme nechat k dispozici OS. Viz. Níže uvedený příklad výpočtu N_s.

Line 8: Zde uvádíme velikost NB (block size NB). Počet čísel musí souhlasit s počtem na uvedeným řádku 7.

Line 9: Řádek specifikuje jakým způsobem budou MPI procesy mapovány na uzlech dané platformy. Je zde na výběr ze dvou druhů row-major a column-major. Toto nastavení se využívá pokud mají uzly více-jádrové procesory. Mapování row-major je doporučeno.

Line 11-12: Dva řádky udávají kolik procesu bude počítáno na uzlech v clusteru. Násobek PxQ koresponduje s počtem procesoru použitých pro benchmark. Počet dvojic PxQ musí odpovídat hodnotě řádku 10.

Line 13: Zde definujeme prahovou hodnotu, se kterou bude porovnávána hodnota rezidua. Reziduum by mělo být prvního řádu, v praxi však může být i řádu nižšího (0.001).

Hodnota 16 pokrývá většinu případů. Pokud dojde k tomu, že reziduum bude větší než threshold je výsledek označen za chybný. Pokud uvedeme jakoukoliv zápornou hodnotu kontrolu je zrušena.

Line 15: PFACTs udává jaká maticově vektorová operace založená na faktorizaci se bude provádět.

Line 17: NBMIN je mezní hodnota pro zastavení rekurze. Pokud bude zvolený panel tvořen méně nebo rovno NBMIN sloupců dojde k zastavení rekurze.

Line 19: NDIV udává na kolik subpanelů bude v každém kroku rozdělen panel při jeho faktorizaci (maticově maticová operace).

Line 21: RFACTs nastaveni rekurzivní panelové faktorizace.

Line 23: BCASTs vybíráme z 6ti popsaných variant virtuální kruhové topologie

Line 25: DEPTHs udává kdy se bude faktorizovat další panel, 0 – po kompletním dokončení zpracovávaného panelu, 1 – okamžitě po aktualizaci nového, až pak aktualizace a dokončení zpracovávaného.

Line 26: Možnost výběru swapovacího algoritmu. Možnost binární výměny, Long (rozprostření) nebo jejich kombinace Mix. Pro velké N je lepší užít Long.

Line 27: Threshold se uplatní při výběru Mix režimu. Pro hodnoty menší se uplatní binary-exchange, pro vetší hodnoty algoritmus rozprostření (long).

Line 28: Nastavení v jaké formě se bude ukládat horní triangulární matice tvořená sloupci panelu

Line 29: Nastavení v jaké formě se bude ukládat panel řádků U

Line 30: Povoluji či zakazuje vyrovnávací fázi

Line 31: Specifikuje zarovnávaní paměťového prostrou alokovaného pro HPL. Na moderních PC pravděpodobně hodnota 4,8 nebo 16. Toto nastavení může způsobit malé plýtvání pamětí.

Podrobně se volbou hodnot a nastavení konfiguračního souboru benchmarku HPL.dat zabývá zpráva od společnosti SUN Microsystems[14]. Zpráva z názvem *ARCHITECTURE AND*

[14] http://www.sun.com/

PERFORMANCE OVERVIEW [18] z roku 2007 podrobně testuje vliv volby jednotlivých parametrů HPL na celkový dosažený výkon clusteru.

Přiklad výpočtu poctu N_s v závislosti na velikosti paměti RAM

Cluster s 25 uzly kde každý má 1GB RAM, celkem tedy **25GB** RAM. Ideálně je potřeba zabrat kolem **80%** operační paměti tj. **25*0.8=20GB**. Převedeme GB na B násobením 1024 tj. **20*1024^3= 21474836480**. Pokud počítáme s dvojitou přesností je potřeba vydělit číslo B/8 tj.**2684354560**. Toto číslo pak vypočítáme druhou odmocninu a dostáváme číslo N tj. **51810.7**

Dvojitá přesnost (**double precision**) je formát čísla, který zabírá dvě sousední místa v paměti počítače. DP může být datový typ integer, fixed point, floating point. Moderní 32-bit procesory (single precision) jsou emulovat 64-bit double precision. Pro dekódování binárních nebo desítkových čísel 8-bit čísel s plovoucí desetinnou čárkou je definována standardem IEEE754. [19]

5.2 OCTAVE

Jedná se o volně šiřitelný OSS program pod licencí GNU GPL, který můžeme upravovat a dále nekomerčně šířit. Autorem je John W. Eaton a další.

GNU Octave[15] je programovací jazyk vysoké úrovně, primárně určen k numerickým výpočtům. Poskytuje rozhranní příkazové řádky k řešení lineárních a nelineárních numerických problémů. Také je možné octave využít pro experimentální numerické výpočty kompatibilní s programem Matlab[16] a vykonávání dávkových souborů.

Do octave je také možné přidat mnoho různých rozšiřujících nástrojů pro řešení běžných problémů lineární algebry, hledání kořenů nelineárních rovnic atd. Sám uživatel může napsat své vlastní funkce v interním programovacím jazyku nebo přidávat moduly napsané v C/C++, Fortranu a dalších. Jedná se tedy o program který je možno přizpůsobit potřebám uživatele.

[15] http://www.octave.org/
[16] http://www.mathworks.com/products/matlab/

5.2.1 OCTAVE MPI tool box

MPITB[17]. Jedná se o sbírku provázaných funkcí a přídavných skriptů, které umožňují použití MPI funkcí v programech napsaných v octave, což umožňuje jejích paralelní spuštění na clusterech či superpočítačích. Autorem je Fernández, J., Anguita, M., Ros, E., & Bernier, J. L. (2006). Více popisují dokumenty [23], [24].

5.3 *PEA* parameterized expectations algorithm

PEA [22] byl implementován v roce 1988. Jedná se o známý mechanismus výpočtu nelineárních dynamických stochastických (náhodných) modelů s racionálním očekáváním. (autoři: (Marcet and Marshall, 1994; Marcet and Lorenzoni, 1999; Christiano and Fisher, 2000).Existuje množství různých implementací, které používají různé metody výpočtu váhových rezidui nebo výpočet integrálů. Macet v roce 1988 implementoval to, co Christiano a Fisher popisovali jako konvenčni PEA. Tato verze nahrazuje chybějící funkci parametrickou aproximační funkci, ze které je pak generována dlouhá série simulovaných dat.

Další aproximační funkce je určena pro generování dat a aktualizaci parametrů. Algoritmus opakuje tyto kroky dokud se už dále parametry nemění a dosáhne konvergence (ustálení).

Výhoda konvenčního PEA je ta, že se jedná o jednoduchý algoritmus, který je lehký na pochopení a jeho chování lze dobře teoreticky odůvodnit. Na druhou stranu nevýhodou může být potřeba použití extrémně dlouhé simulace pro dosažení shodných výsledků jako aproximační funkce PEA. Což způsobuje výpočetní náročnost.

Nicméně konvenční PEA lze jednoduše upravit pro paralelní spuštění na více procesorech. Jak délka simulace, tak i krok aktualizace parametru může být paralelizován.

[17] http://atc.ugr.es/javier-bin/mpitb

Jestliže je použit dostatečný počet procesorů, dochází k velké redukci celkového času nalezení řešení při dané přesnosti.

5.4 John the ripper s podporou MPI

John the Ripper (JTR) [20] je nejrychlejší lamač hesel, aktuálně dostupný pro mnoho operačních systémů Unixového typu (11 oficiálně podporovaných, nepočítaje OS pro různé architektury CPU), DOS, Win32, BeOS a OpenVMS. Primární účel tohoto OSS je detekovat slabá Unixová hesla. Mimo jiné dokáže luštit běžné typy hashů hesel, které se nachází na různých druzích unixových systémů, s podporou Kerberos/AFS a Windows LM hashů, navíc je zde možnost rozšíření o další druhy hesel pomocí patchů.

Používam verzi john-1.7.3.1-all-2-mpi8[18], která aktualizována patchem (autor Ryan Lim) pro podporu MPI, ve snaze o dosažení vyššího výkonu (rychlosti) při lámání hesel. MPI umožnuje spustit JTR na více procesorech jak na jednom stroji, tak na výpočetním clusteru. Dá se říci, že v dnešní době rozmachu vícejadrových procesorů je tento patch velmi praktický.

5.4.1 Vlastnosti a výkon

JTR je navržen tak, aby nabízel co nejvíce schopností a zároveň zůstával rychlý. Kombinuje několik crackovacích módů a je plně konfigurovatelný pro různé potřeby uživatelů (možnost definovat vlastní crackovací mód pomocí interního překladače podporující jazyk C). Zároveň je dostupný JTR pro několik různých platforem, což umožňuje používat všude stejný crackovací program (možnost pokračování v přerušeném crackování na různých platformách).

Po nainstalování JTR podporuje (a automaticky detekuje) následující Unixové typy hashů[19]: tradiční DES a DES s dvojtou délkou, BSDI rozšířený DES, FreeBSD založený na

[18] dostupná ze stránek http://www.bindshell.net/tools/johntheripper
[19] Hash je heslo nebo tajná fráze zašifrovaná známým algoritmem tak, že se zpětně k heslu nedá dopracovat.

MD5 (užívaný pro linux či cisco IOS) a OpenBSD Blowfish (užití v některých distribucích linuxu). Je schopen luštit hesla (hashe) s Kerberos /AFS a windows LM.

Rozšiřující patche přidávají podporu pro mnoho dalších typu hesel a hashů, včetně hashe windows NTLM (MD4), běžné typy hashů používaných na Open VMS, MySQL,Netscape LADP server, Eggdrop IRC bot, S/Key skeykeys soubory, Kerberos v4 TGTs a další.

Na rozdíl od ostatních crackovacích programů JTR nepoužívá postup crypt(3)[20]. Místo něj má vlastní vysoce optimalizované moduly pro různé typy hashů a architektur procesorů. Některé z algoritmů používají (např. Bitslice DES[21]) nemohou mít implementovány uvnitř crypt(3) API; tyto používají výkonnější rozhranní jako používá JTR. Navíc je zde možnost využít rutiny assembly jazyků pro běžné architektury procesorů, konkrétně pro x86 s podporou MMX a SSE2.

[20] http://linux.die.net/man/3/crypt
[21] http://www.darkside.com.au/bitslice/

6 Dosažené výsledky

6.1 HPL benchmark

Měření bylo prováděno stejnorodých počítačích v konfiguraci viz. tabulka, kde jsou uvedeny podstatné vlastnosti pro vykonávané benchmarky.

Tab. 1: Konfigurace počítačů použitých v clusteru

CPU	Intel Core 2 Duo @ 1.866 GHz
Operační paměť RAM	1024MB DDR2
Sítové adaptéry Ethernet LAN	100Mbps, 1Gbps
Operační systémy	Debian GNU/Linux 5.0 x86_amd64, x86

64b procesor použitý v každém uzlu clusteru byl podroben testům jak s 64b OS GNU/Linux 32b OS GNU/Linux.

Tab. 2: Tabulka naměřených hodnot HPL pro 32b a 64b OS

počet PC	block size N_s	výkon [Gflop/s]	doba výpočtu [s]	block size N_s	výkon [Gflop/s]	doba výpočtu [s]	rozdíl v použitém OS [%]
1	9600	3,399	173,55	9600	5,542	108,85	38,67
2	13000	5,232	280,9	14400	7,776	256,04	32,72
3	15000	6,676	337,07	16500	9,559	373,83	30,16
4	17500	8,2	435,36	18500	10,67	334,8	23,15
5	20250	9,4256	585,49	20250	11,7	473,05	19,44
7	24000	10,74	858,46	24000	12,94	702,27	17,00
10	32256	18,26	1225,1	28000	20,72	756,28	11,87
15	36000	20,53	1515,29	35000	25,95	1101	20,89
20	42000	29,44	1678,08	40000	32,66	1536	9,86
25	47000	35,35	1958,03	45000	36,83	1624,9	4,02
27	49000	38,27	2049,29	47000	42,94	1731,1	10,88
pozn.	32b OS			64b OS			pokles

Architektura procesoru použitého ve všech uzlech clusteru je x86_64. Tabulka 2 uvádí výsledky benchmarku HPL pro 32b OS i 64b OS. Rozdíl při použití 32b OS je kolem 38% snížení výkonu ve srovnání s 64b. Postupně je rozdíl smazáván z důvodu nedostatečné propustnosti použité 100Mbps ethernet LAN.

Při použití 32b OS na 64b CPU může být pokles výkonu obecně způsoben tím, do jaké míry dokáže kompilátor přizpůsobit překládanou aplikaci pro danou architekturu. Také je zřejmé, že pokud pracujeme s velkými čísli (větší než 32b) dochází ke zvýhodnění 64b architektury jelikož se čísla nemusí rozkládat na menší a tím vykonávat operace navíc.

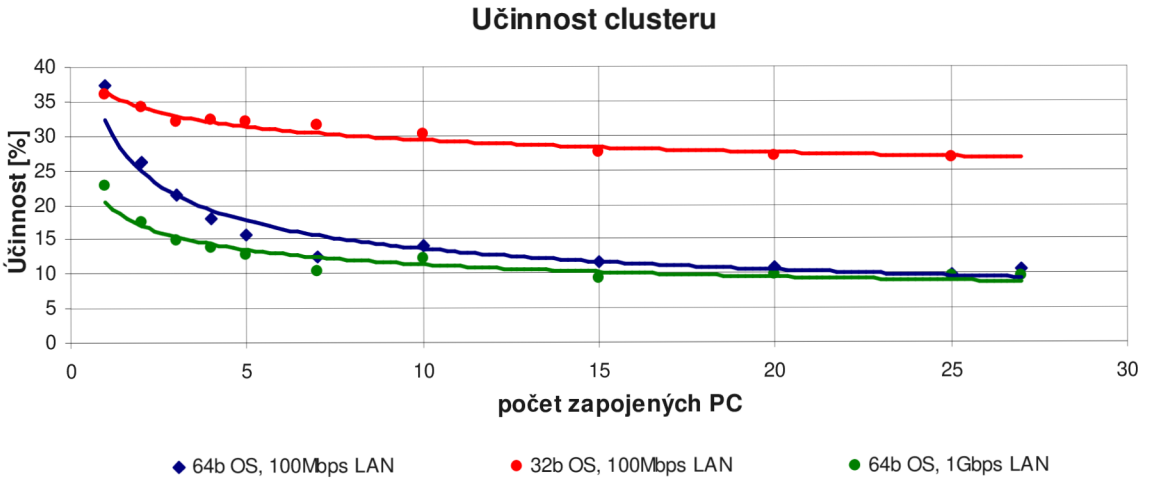
V další tabulce 3 můžeme vidět srovnání výsledků HPL při použití při použití 1Gbps LAN. Do výpočtu HPL bylo zapojeno 25 počítačů.

Tab. 3: Vliv LAN na výsledky HPL

Počet PC	block size N_s	výkon [Gflop/s]	doba výpočtu [s]	výkon [Gflop/s]	doba výpočtu [s]	zvýšení výkonu [%]	Nárůst doby výpočtu [%]
1	9600	5,542	108,85	5,35	110,3	-3,46	-1,33
2	14400	7,776	256,04	10,17	161,2	30,79	37,04
3	16500	9,559	373,83	14,26	210	49,18	43,82
4	18500	10,67	334,8	19,25	219,26	80,41	34,51
5	20250	11,7	473,05	23,8	258	103,42	45,46
7	24000	12,94	702,27	32,71	281,7	152,78	59,89
10	28000	20,72	756,28	44,86	401	116,51	46,98
15	35000	25,95	1101	61,64	659,27	137,53	40,12
20	40000	32,66	1536	80,4	659,27	146,17	57,08
25	45000	36,83	1624,9	99,2	697,79	169,35	57,06
Pozn.		64b OS, 100Mbps		64b OS, 1Gbps		výpočty	

V tabulce 3 vidíme v posledním sloupci zvýšení výkonu a nárůst doby výpočtu v %. První hodnota je záporná což je způsobeno chybou měření. Na tomto místě by správně měla být hodnota 0. Předpokládáme, že jeden uzel se po síti nic nepřenáší tedy by výkon měl být stejný jak na 100Mbps tak na 1Gbps LAN.

Zlepšení za použití 1Gbps je značné. Pro konkrétní cluster s 25ti uzly je zhruba 170%. To je způsobeno velkými nároky na přenášené data po síti. Můžeme říci že pro HPL benchmark je 100Mbps úzkým hrdlem, který omezuje možnosti clusteru.



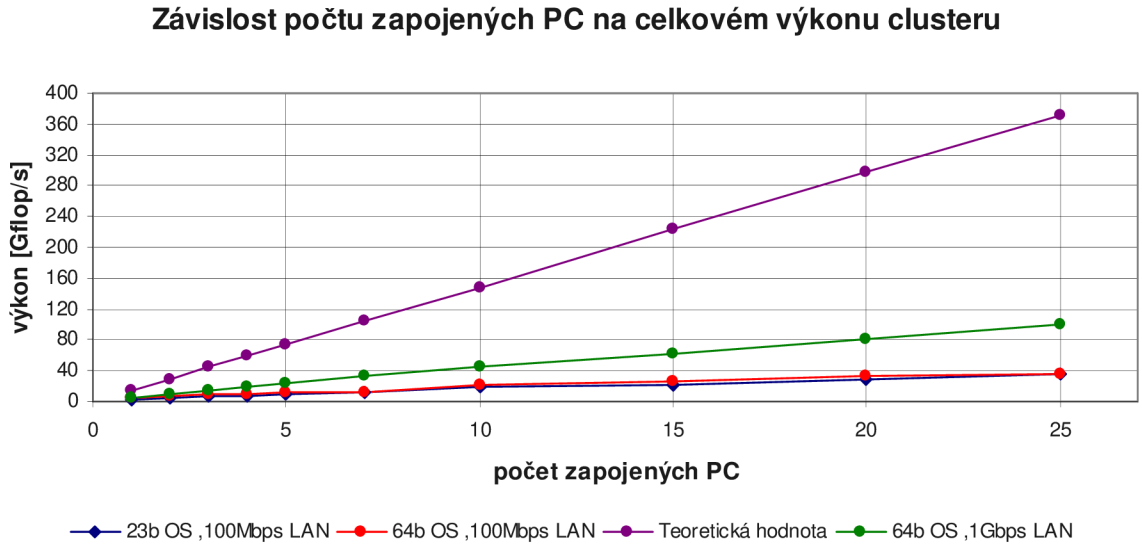
Obr. 16: Zobrazení účinnosti výpočtu

Při výpočtu účinnosti je důležité zjistit kolik je daný procesor schopen vykonat operací v jednom pracovním cyklu. Účinnost je pak počítána pro použitý procesor z jednoduchého vzorce. Pro zjištění **teoretické maximální hodnoty** výkonu procesoru $\mathbf{R_{peak}}$ vynásobíme počet obsažených jader, pracovní frekvenci a počet operací v jednom cyklu.

$$R_{peak} = N_{cores} * f * N_{operations} \tag{1.5}$$

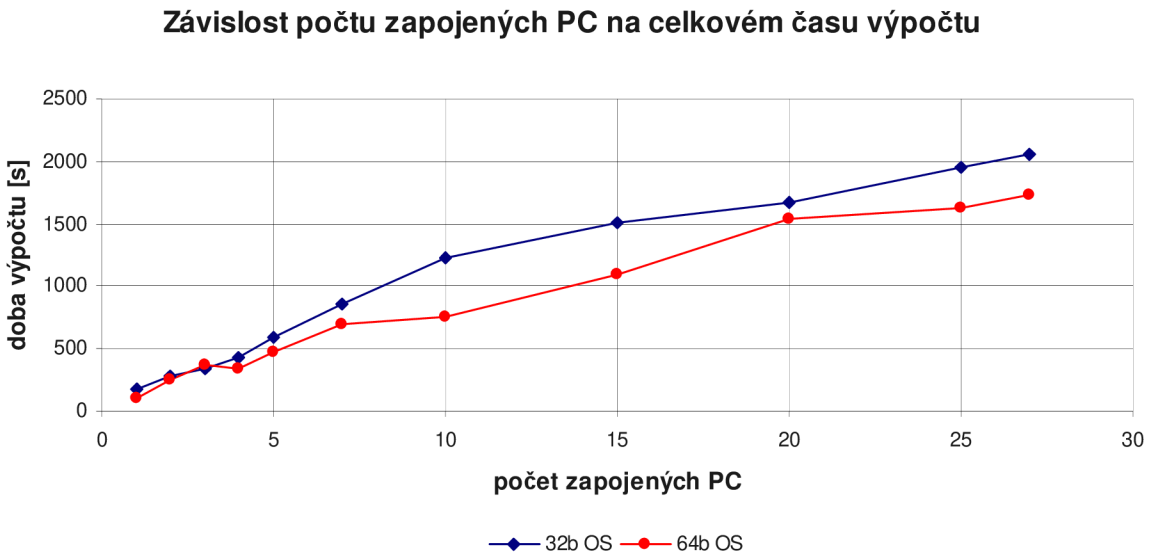
V případě použití Intel Core 2 Duo s frekvencí 1.866 GHz se tedy dostáváme k Rpeak = 14,88 Gflop/s pro jeden zapojený uzel.

Na obr. 17 vidíme závislost počtu připojených uzlů na účinnosti výpočtu. Na první pohled vidíme ze účinnost klesá méně při použití 1Gbps LAN než při použití 100Mbps LAN. To je způsobeno tím, že počítače připojené do clusteru jsou schopny zpracovávat velké množství vypočtu jejichž výsledky si musí uzly předávat prostřednictvím MPI po spojující sítí.



Obr. 17: Závislost zapojených PC na celkovém výkonu clusteru

Graf na obr. 17 udává závislost připojených uzlu na celkovém výkonu clusteru. Jsou zde naznačeny celkem 4 průběhy pro 32b OS, 64b OS na 100Mbps a 1Gbps LAN a teoretická maximální hodnota, které se snažíme co nejvíce přiblížit. Toto se nejvíce daří při použití 64b OS na uzlech propojených 1Gbps LAN. Gigabitový ethernet má ve srovnání s klasickým 100Mbps větší propustnost, šířku pásma a rychlejší odezvu. Což se pozitivně podepsalo jak na celkovém dosaženém výkonu, tak i na účinnosti clusteru.



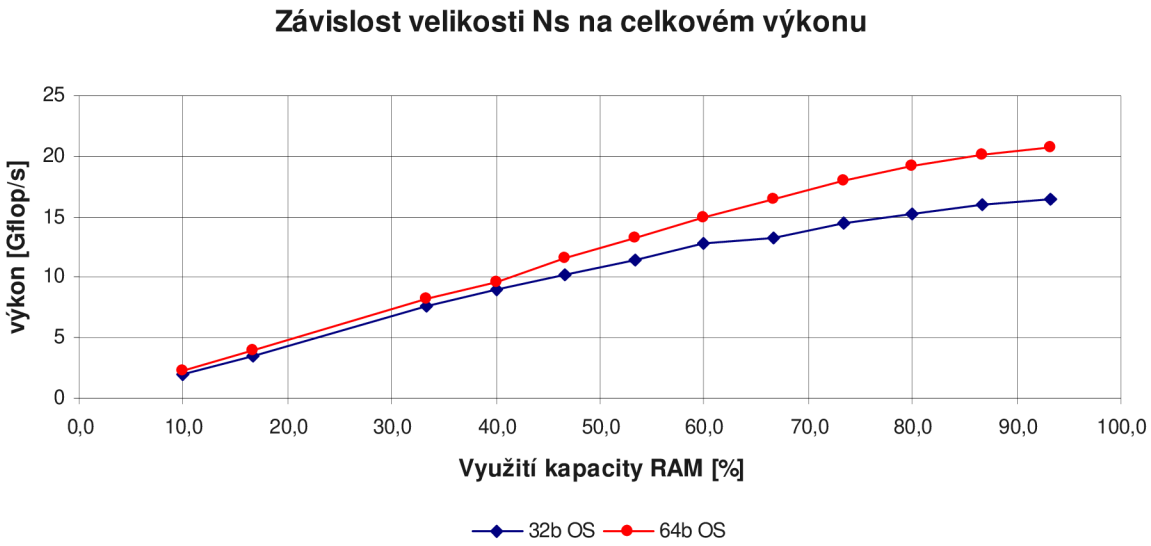
Obr. 18: : Závislost zapojených PC na celkové době výpočtu

Další z grafů popisuje závislost výpočetní doby pro maximální velikost N_s pro zapojené uzly. Závislost je do jisté míry lineárního charakteru. Opět je zde patrný rozdíl při použití OS pro architektury i386 a x86_64. Logicky je doba výpočtu kratší pro 64b OS. Měření bylo provedeno na 100Mbps LAN.

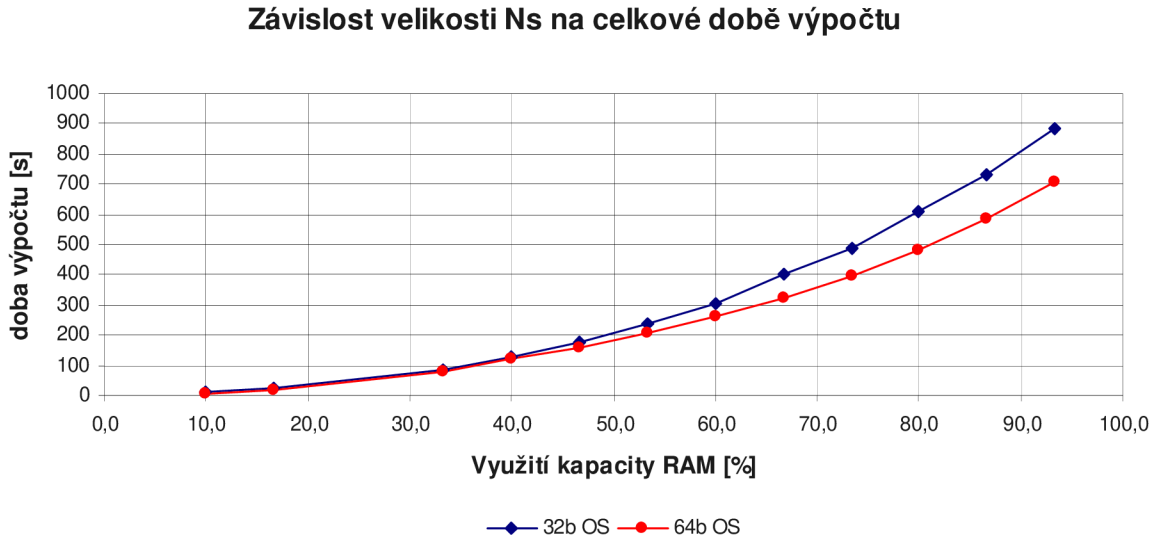
Tab. 4: Vliv parametru N_s na dobu výpočtu a výkon

block size N_s		Doba výpočtu [s]		výkon [Gflop/s]	
RAM [%]	N_s				
93,3	28000	887,14	706,28	16,5	20,72
86,7	26000	729,44	584,2	16,06	20,06
80,0	24000	606,77	478,7	15,2	19,25
73,3	22000	490,42	395,7	14,48	17,94
66,7	20000	403,42	324,25	13,23	16,45
60,0	18000	303,3	260,3	12,82	14,94
53,3	16000	239,5	206,2	11,4	13,25
46,7	14000	178,72	157,91	10,24	11,6
40,0	12000	127,78	119,65	9,017	9,63
33,3	10000	87,3	81,42	7,64	8,2
16,7	5000	23,67	21	3,522	3,972
10,0	3000	9,33	8	1,93	2,248
Pozn.		32b OS	64b OS	32b OS	64b OS

Bylo testováno také do jaké míry je ovlivněn celkový výkon clusteru složený z 10 uzlů při změnách velikosti zpracovávaného problému N_s. Z tabulky 4 vychází graf 19. Paměť RAM byla naplněna od 10% do 93%. Tendence v poslední části 80% je spíše konstantní. Tudíž při větší velikosti operační paměti by už rozdíl ve výkonu nebyl natolik patrný.



Obr. 19: Závislost velikosti problému N_s na celkovém výkonu clusteru



Obr. 20: Závislost velikosti problému N_s na celkové době výpočtu

Závislost velikosti zpracovávaného problému N_s na době výpočtu. Z grafu vyplívá, že čas roste exponenciálně. V poměru k výkonu je patrné, že pokud by byla k dispozici vetší kapacita operační paměti při dalším zvyšování N_s by docházelo k výraznému prodloužení výpočetní doby, ale zvyšování výkonu by bylo velmi malé. Tzn. Efektivita by byla stále menší.

Tab. 5: Srovnání clusterů

Druh CPU, výrobce, frekvence, použitá síťová technologie	Počet CPU	R_{max} [Gflop/s]	N_{max}	$N_{1/2}$	R_{peak} [Gflop/s]	Účinnost [%]
Self-Made Intel Pentium 4 Xeon(1.7GHz w/GigE)	208	197,2	90000		707	27,89
IBM xSeries(2.8 GHz Intel P4 w/Myrinet 2000)	126	443,7	125000		705	62,88
Fujitsu VPP5000/56 (3.33nsec)	56	492,4	228480	12768	538	91,52
NEC SX-6/64M8	64	495,2	122880	6656	512	96,72
Intel Pentium III (1 GHz w/100 Mb enet)	512	169,4	16000		512	33,09
Intel P 4 cluster(92-2.0GHz+6-1.7GHz w/Genet)	98	160,4	75500	24000	388	41,34
IBM eServer pSeries 655 (8-way 1.5 GHz POWER4+)	64	248,7	160000	11000	384	64,77
HP Superdome (1.5GHz Itanium 2, 6.0MB L3 Cache)	64	341,7	154080	15040	384	88,98
Pentium 4 (2 GHz w/Giganet)	91	157,8	73500	26000	364	43,35
HP rx26000 Itanium2 1.3GHz Cluster w/InfiniBand	64	278,7	98304	9216	332,8	83,74
SGI 1100 Cluster (Dual Pentium III, 1 GHz)	324	140,5	133000		324	43,36
Intel Core 2 Duo@1.866 GHz / GigE	50	99.2	47000		372	26,6

Tabulka 5 je vybrána z dokumentu vydaného a aktualizovaného univerzitou v Tennesse [21]. V tabulce jsou clustery nebo superpočítače, které mají podobné vlastnosti jako sestavený cluster. Konkrétně se jedná o přibližný špičkový výkon R_{peak}, počet použitých procesorů a u některých procesorů je to i výrobce. Mimo tyto hodnoty jsou zde uvedeny R_{max} největší dosažený výkon clusteru, N_{max} je maximální hodnota problému N_s, který je možný spustit na daném stroji. Uvedená hodnota $N_{1/2}$, je u některých clusterů v tabulce, značí problém N_s, který je potřeba k dosažení poloviního výkonu R_{max}.

Účinností můžeme sestavený cluster přirovnat k **Self-Made Intel Pentium 4 Xeon(1.7GHz w/GigE)**, který dosahuje mírně vyšši hodnoty a to 27,9%. Bylo zde zapojeno 208 procesorů a počítače byli propojen gigabitovou sítí.

Tab. 6: Srovnání procesorů

Druh procesoru	n= 100	n= 1000	R_{peak} [Mflop/s]	Účinnost [%]
Intel Pentium Woodcrest (1 core, 3 GHz)	3018	6542	12000	54,52
Intel Pentium Woodcrest (1 core, 2.67 GHz)		2636	10680	24,68
Intel Core 2 Q6600 Kensfield) (2 core, 2.4 GHz)		9669	19200	50,36
NEC SX-8/1 (1proc. 2 GHz)	2177	14960	16000	93,50
HP ProLiant BL20p G3 (2 x 3.8GHz Intel Xeon)		8185	14800	55,30
HP ProLiant DL360 G4 (2 proc, 3.6GHz/1MB Xeon)		7031	14400	48,83
HP ProLiant BL30p (2 proc. 3.20 GHz, Xeon)		6264	12800	48,94
Fujitsu Siemens hpcLine (2 proc Intel Xeon 3.2GHz)		5151	12800	40,24
Intel Core 2 Duo (2core, 1.866 GHz)		5542	14928	37,12

Ve výše uvedené tabulce 6 opět vybrané z dokumentu [21], můžeme srovnat jednotlivé typy procesorů s testovaným **Intel Core 2 Duo.** Tento procesor dosahuje účinnosti 37,12 %, vzhledem podobnou účinnost má také Fujitsu Siemens hpcLine (2 proc **Intel Xeon** 3.2GHz) 40,24 %. Ještě nižší hodnota je u **Intel Pentium Woodcrest** (1 core, 2.67 GHz). Naopak nejvyšší účinnost dosahuje CPU **NEC SX-8/1** (1proc. 2 GHz) a to 93,5 %. Účinnost ostatních CPU se pohybuje kolem hodnoty 50 %.

6.2 Měření s algoritmem PEA

Provedená měření s algoritmem PEA[22] spouštěným v prostředí OCTAVE s MPITB. Měření bylo prováděno na počítačích s procesorem Intel Core 2 duo s frekvencí 1.866 GHz s 1 GB operační pamětí RAM. Do výpočtu bylo zapojeno 10 počítačů, tj. 20 procesorů.

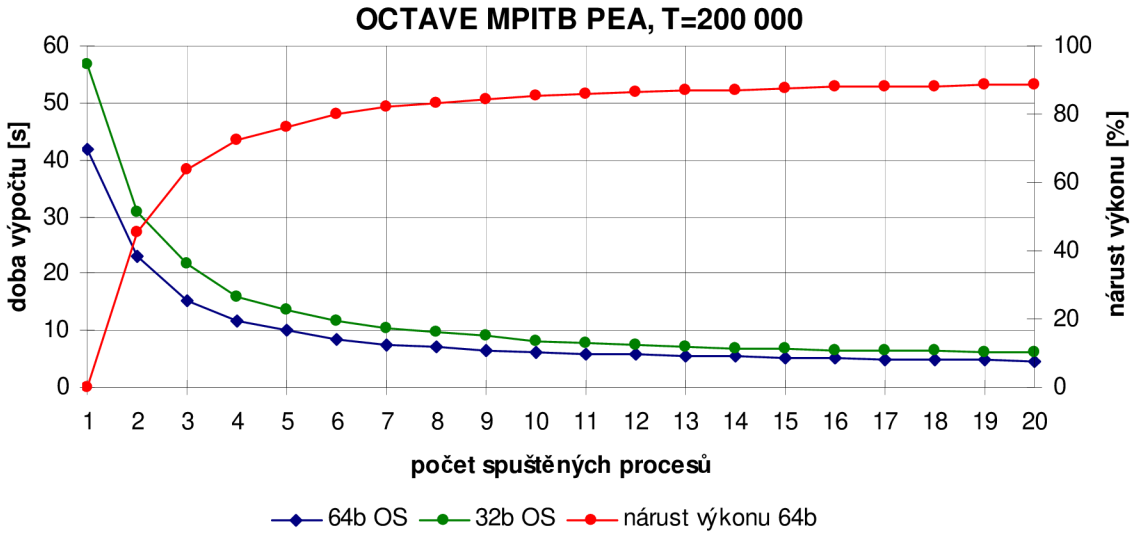
[22] Možnost stažení algoritmu: http://pareto.uab.es/mcreel/pea.zip

Výpočty s PEA OCTAVE MPITB sample size: 200 000, burin: 100, maxiters:30 10 PC (x2 core).

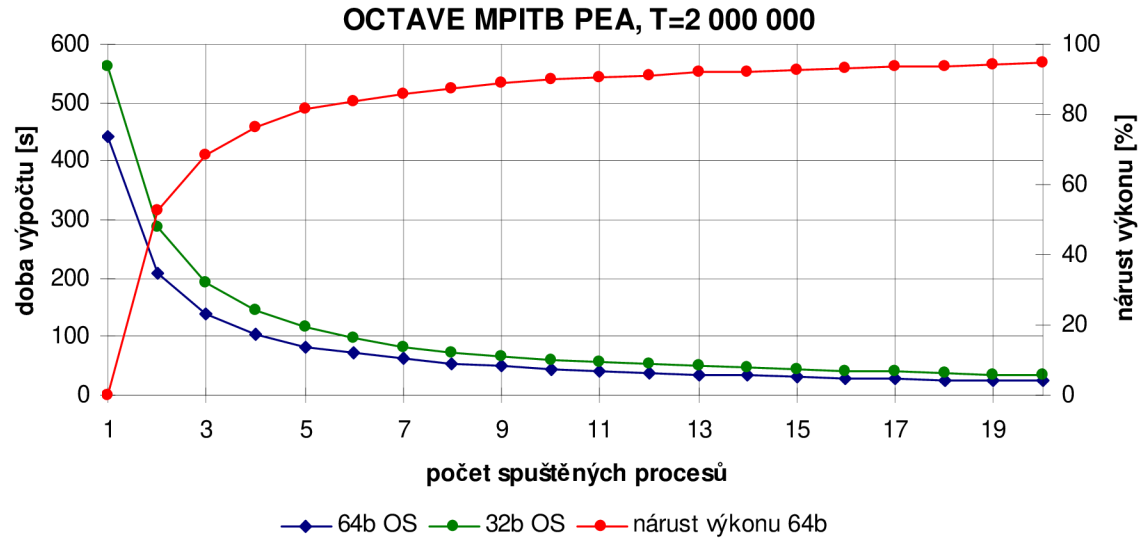
Tab. 7: Naměřené časy pro výpočet algoritmu PEA

počet procesů	doba výpočtu [s]				
použitý OS	32b	64b	64b	32b	64b
1	56,8	41,976	43,3	562,5	441,02
2	30,7	23	24,6	287,27	208,08
3	21,7	15,1	15,54	191,7	139,33
4	15,95	11,6	11,36	146,34	105,24
5	13,6	9,9	9,57	116,75	81,98
6	11,7	8,5	8,23	97,63	71,54
7	10,53	7,6	7,47	83,2	63,27
8	9,8	7,04	6,96	74,12	54,8
9	8,96	6,64	6,2	66,09	49,26
10	8,21	6,2	6,02	60,14	44,04
11	7,8	5,89	5,68	58,07	41,44
12	7,4	5,71	5,43	52,86	38,37
13	7,13	5,55	5,3	49,57	35,89
14	6,92	5,4	5,17	46,18	34,25
15	6,76	5,21	4,99	43,79	31,81
16	6,58	5,07	4,93	41,74	29,41
17	6,4	5,01	4,8	39,56	27,59
18	6,38	4,95	4,69	37,61	26,74
19	6,27	4,85	4,63	36,15	25,21
20	6,15	4,7	4,66	35,8	24,08
LAN	100Mbps	100Mbps	1Gbps	100Mbps	100Mbps
Sample size	T =200 000			T=2 000 000	

V tabulce vidíme záznam pěti provedených měření na 32b OS, 64b OS pro 100 Mbps LAN a také 1 Gbps LAN pro dvě nastavení algoritmu PEA. Jedná se o čas výpočtu 30 iterací PEA, při použití simulace délky 200 000 period s přídavnými 100 periodami vyjmutými z každé nezávislé simulace. Hlavní nastavené parametry pro algoritmus PEA byly sample size T= 200 000, burin 100, matrixes 30. Při druhém měření byl zvětšen sample size T na 2 000 000 period, čímž bylo dosaženo delšího výpočetního času.



Obr. 21: Redukce času výpočtu PEA pro T= 200 000



Obr. 22: Redukce času výpočtu PEA pro T= 2 000 000

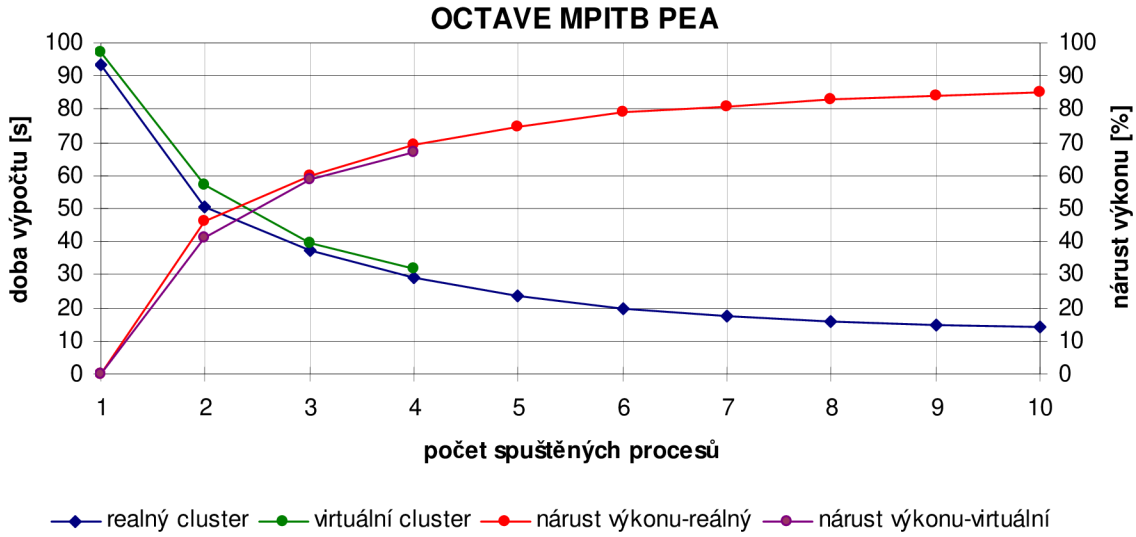
Na obrázcích 21 a 22 jsou grafy závislosti doby výpočtu algoritmu PEA na počtu spuštěných procesů na clusteru složeného z 10 uzlů.. Vidíme srovnání dvou použitých OS. Opět lepe vychází 64b OS na procesorech s x86_64 architekturou. 32b OS dosahuje

prodloužení výpočetní doby v porovnání s 32b OS kolem 30-40%. Průměrné zhoršení o 35% je patrné jak pro jeden výpočetní proces, tak pro 20 měřených procesů. Což ukazuje také to, co je vidět v tabulce 5 při použití 1Gbps ethernetové LAN, že propustnost sítě nemá výrazný vliv na výsledek. Pro oba druhy testované LAN dochází ke zvyšování výkonu až do 90%, ve srovnání s jedním procesem, toto vidíme také v grafu 21 a 22. Nejvyšší nárůst při výpočtu PEA je při zapojení prvních 5 procesů zvýšení výkonu o 80%.

Dosažené výsledky můžeme srovnat s výsledky dosaženými na clusteru složeného z deseti fyzických uzlů a také clusteru složeného ze čtyř virtualizovaných uzlů. Přesnější popis a tabulku nalezneme v [22]. V případě reálných PC měly jednotlivé uzly k dispozici CPU Pentium IV s pracovním taktem 3.0 GHz a byly propojeny 100Mbps ethernet LAN. Použita byla 32b linuxová distribuce knoppix. Virtuální cluster byl zprovozněn na dvou realných serverech s Intel Xeon 64 bit CPUs s frekvencí 3.6 GHz, oba spojeny 1Gbps LAN. Na každém severu běžely ve Vmware dva virtualizované stroje s 32b OS knoppix linux. Celkem tedy 4 virtuální uzly spojeny do clusteru. V tabulce vidíme naměřené hodnoty pro algoritmus PEA s parametry sample size T= 200 000, burin 100, matrixes 30.

Tab. 8: Fyzický a virtuální cluster pro srovnání

	Fyzický cluster		Virtuální cluster	
počet procesů	čas výpočtu [s]	nárůst výkonu [%]	čas výpočtu [s]	nárůst výkonu [%]
1	93,55	0	97,13	0
2	50,517	46	57,3067	41
3	37,42	60	39,8233	59
4	29,0005	69	32,0529	67
5	23,3875	75		
6	19,6455	79		
7	17,7745	81		
8	15,9035	83		
9	14,968	84		
10	14,0325	85		



Obr. 23: Srovnání hodnot měření

V grafu vidíme narůst výkonu v poměru k jednomu spuštěnému procesu. Druhý průběh ukazuje závislost výpočetní doby na počtu výpočetních procesů. Virtualizovaný server je ve výpočtech mírně pomalejší než reálný cluster se 4 uzly, což je nejspíše způsobeno poklesem výkonu při virtualizaci operačních systémů. Výsledné průběhy jsou srovnatelné s měřením na testovaném clusteru. Rozdíly jsou zde v době výpočtu, což je způsobeno jiným výpočetním výkonem použitých procesorů.

6.3 Benchmarky s JTR

John The Ripper byl vybrán jako praktická ukázka možností využití výpočetního clusteru. Pomocí tohoto programu můžeme ověřit jaké vlastnosti má zvolené heslo pro přístup do OS. Provedená měření s programem JTR byla prováděna na počítačích s procesorem Intel Core 2 duo s frekvencí 1.866 GHz s 1 GB operační pamětí RAM. Cluster byl sestaven z 10-ti uzlů, tj. 20 procesorů. Benchmarky byly prováděny pro 32b i 64b OS. Pro 64b OS bylo měření podrobnější s testem výkonu na clusteru s 25 uzly.

Z 36 podporovaných různých druhu šifrování bylo zvoleno 5 dobře známých šifer či hashovacích funkcí. Konkrétně tradiční DES, Microsoft Hash, hrubá MD5, MD5 pro FreeBSD, a SHA1, které vidíme v tabulkách níže.

Tab. 9: Hodnoty JTR benchmarku pro 32b OS

počet procesů	Traditional DES [128/128 BS SSE2]		M$ Cache Hash [Generic 1x]		FreeBSD MD5	Raw MD5 [raw-md5]	Raw SHA-1 SSE2
	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]
1	1592000	1336000	11306000	4242000	5136	3723000	6205000
2	3185000	2674000	22643000	8372000	10272	7423000	12377000
10	15934000	13302000	113676000	42172000	51312	36868000	61946000
20	31826000	26567000	226950000	84343000	102581	73978000	123741000
poznámka	Many salts	Only one salt	Many salts	Only one salt			

JTR benchmark pro 32 OS na clusteru složeného z 10 uzlů.

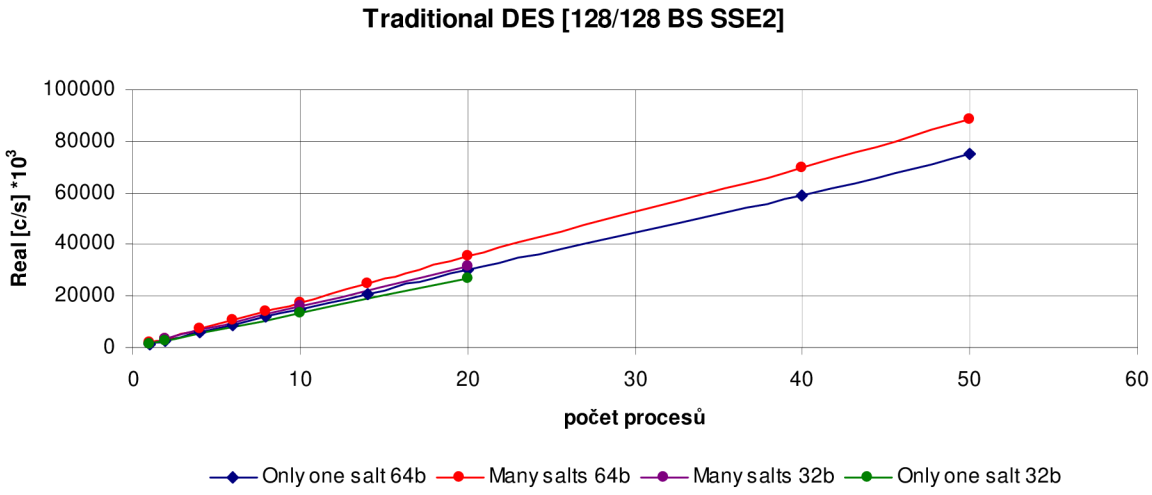
Tab. 10: Hodnoty JTR benchmarku pro 64b OS

počet procesů	Traditional DES [128/128 BS SSE2]		M$ Cache Hash [Generic 1x]		FreeBSD MD5	Raw MD5 [raw-md5]	Raw SHA-1 SSE2
	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]
1	1778000	1507000	10496000	4176000	7984	3442000	2476000
2	3548000	3016000	20950000	8352000	15964	6812000	4916000
4	7115000	6034000	41961000	16690000	31942	13673000	9866000
6	10674000	9053000	62909000	25016000	47854	20579000	14748000
8	14230000	12069000	83764000	33388000	63884	27397000	19691000
10	17760000	15085000	104703000	41756000	79782	34228000	24776000
14	24903000	21111000	146790000	58459000	111508	47990000	34627000
20	35570000	30157000	209496000	83415000	159344	68573000	48963000
40	69655000	59192000	411302000	163897000	312527	134423000	96612000
50	88743000	75420000	524106000	208529000	398740	171330000	123319000
poznámka	Many salts	Only one salt	Many salts	Only one salt	-	-	-

V obou výše uvedených tabulkách 9 a 10 jsou uvedeny hodnoty benchmarku JTR. Hodnoty jsou v jednotkách **c/s** což znamená kombinace uživatelského jména a hesla za sekundu (**combination per second**), nikoliv šifry za sekundu (crypts per sekond) jak by se mohlo zdát. Výstupem JTR benchmarku jsou vždy dvě hodnoty a to **real** a **virtual,** které udávají hodnoty c/s při aktuální zátěži uzlů a také teoretické hodnoty bez zátěže. Já jsem

zvolil hodnoty real jelikož benchmark byl spouštěn vždy na nezatíženém clusteru složeného až 25 uzlů s 64b OS.

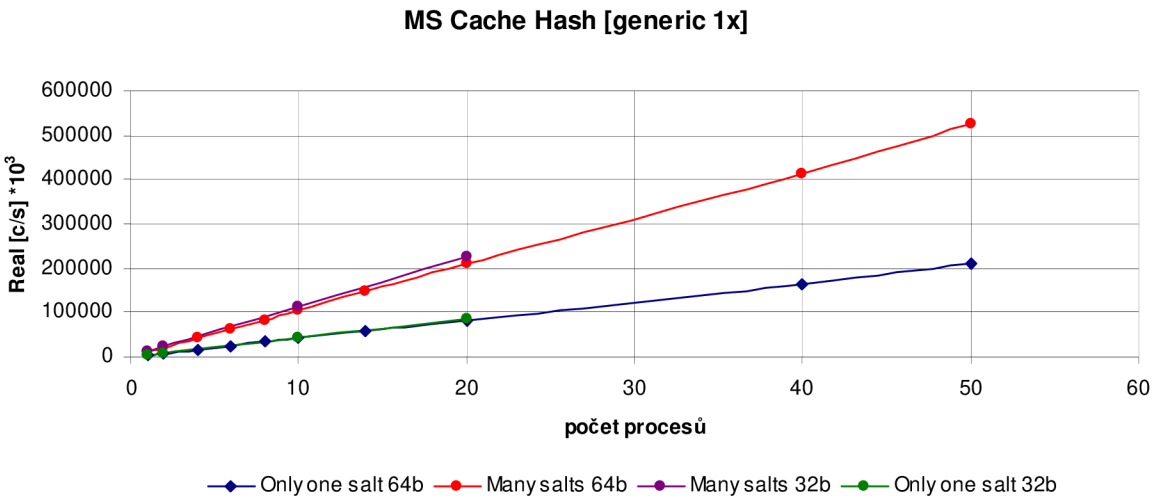
V poznámce u tabulek vidíme výrazy Many Salt a Only One Salt. Salt v kryptografii, slouží ke zvýšení bezpečnosti uživatelského hesla. Jedná se o kombinací náhodných bitů přidávaných k heslu před vytvořením hashe. Případný útočník tak musí odhalit i přidaný salt před rozluštěním hesla. Salt tedy komplikuje porovnávání předvytvořených hashů ze slovníku (slovníkový útok, rainbow tables[23]). Každý přidaný bit saltu, zbůsobí zdvojnásobení potředu diskové a výpočetní kapacity. Salty můžeme tedy kombinovat a přidávat na různá místa v heselné frazi.



Obr. 24: Zvýšení počtu kombinací za sekundu u klasické šifry DES

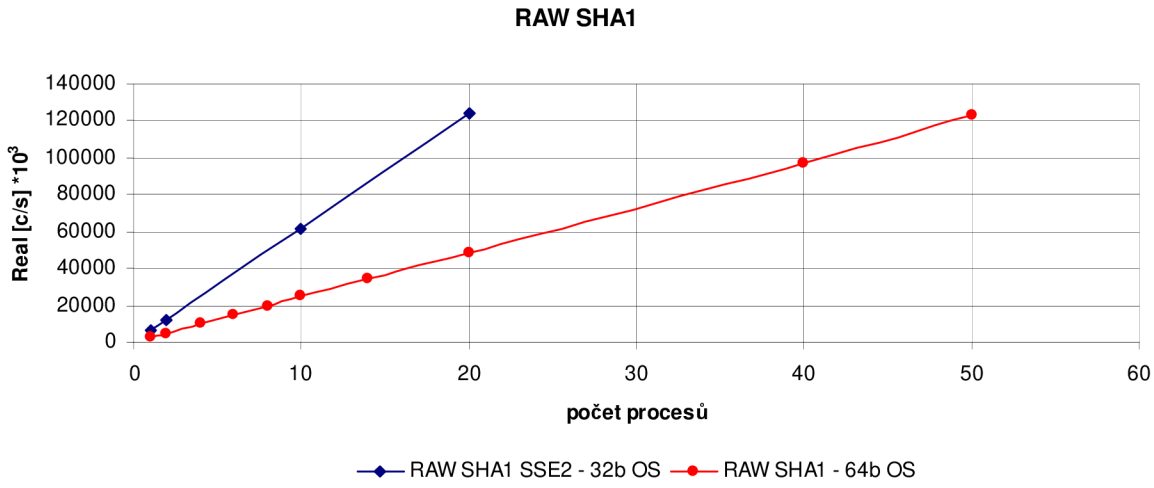
Na grafu 24 vidíme, že 64b OS je na tom asi o 10 % lépe než 32b OS.

[23] http://en.wikipedia.org/wiki/Rainbow_table



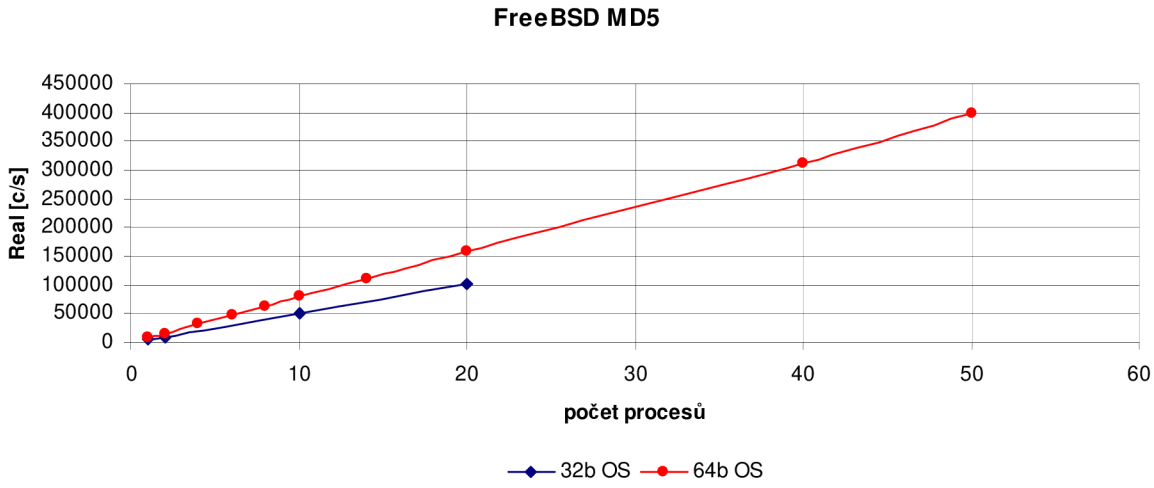
Obr. 25: Zvýšení počtu kombinací za sekundu u Microsoft Cache Hash

Pro Microsoft cache hashe je rozdíl v použitém OS velmi nepatrný.



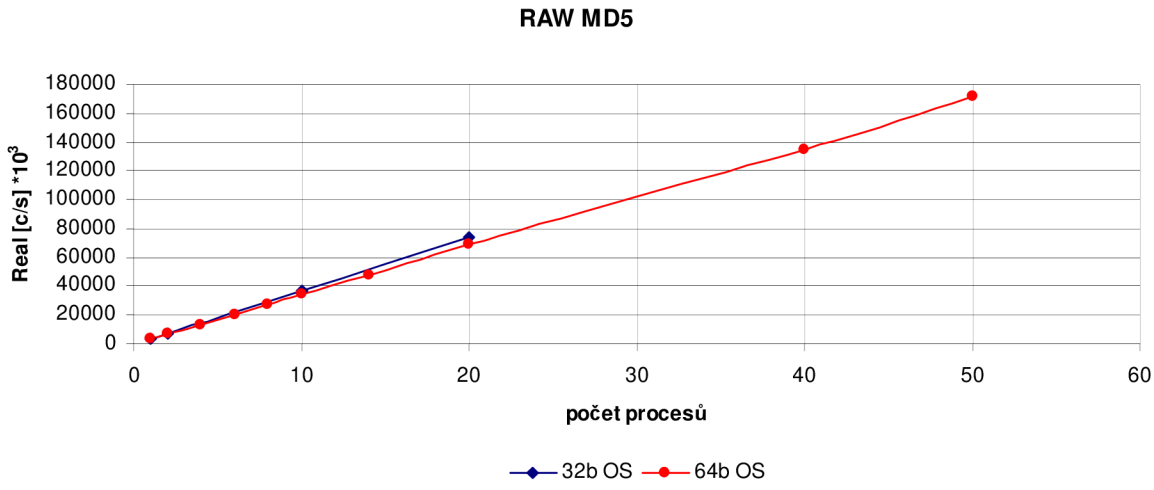
Obr. 26: Zvýšení počtu kombinací za sekundu u klasické šifry SHA1

Na grafu č. 27 vidíme výrazný rozdíl při použití instrukcí SSE2. Výkon pro 20 spuštěných procesů na 32b OS, se zvednul v porovnání s 64b OS trojnásobně, v poměru to znamená nárust výkonu o 40 %.



Obr. 27: Zvýšení počtu kombinací za sekundu u šifry MD5 používané ve FreeBSD

Výkon pro šifru MD5 používanou na systémech FreeBSD je na 64b OS o 35 % větší než na 32b OS.



Obr. 28: Zvýšení počtu kombinací za sekundu u klasické šifry MD5

Rozdíl v použitém OS je u klasické šifry MD5 minimální.

Ze všech grafů je patrný linearní nárust hodnot kobminací za sekudnu, jinými slovy není patrné žádné omezení ze strany sítě. To je způsobeno tím, že při tomto benchmarku není

potřeba tak velké množství přenosové režie. Na začátku jsou uloženy všechny hashe v systému na každém uzlu ale i při této aktivitě je přenos stále minimální. Možný pokles by nastal v případě připojení velkého počtu uzlů v kombinaci s použití náročné šifry. (např. Šifra blowfish je pomalá – malý přenos atp.) U reřimu test (benchmark) je linearní nárust způsoben tím, že JTR bere čistě vypočtená čísla která spolu sčítá. V průběhu výpočtu není potřeba si vyměnovat další informace po síti dokud není výpočet dokončen.

Výsledky můžeme srovnat s následující tabulkou 11 dostupnou na internetových stránkách[24]. Další výsledky pro jednotlivé počítače je možné nalézt na stránkách JTR komunity[25].

Tab. 11: Hodnoty DES pro srovnání JTR benchmarku

počet procesů	Quad Core Intel Core2 Quad Q6600 @2304MHz		Dual Core Intel Core2 Duo, 2.16GHz, MacBook Pro MacOSX		Dual AMD Opteron 250 (2.2ghz), Gentoo Linux 64bit	
	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]	Real [c/s]
2	-	-	3433087	2852658	2132632	1951692
4	8749000	7474000	-	-	-	-
poznámka	Many salts	Only one salt	Many salts	Only one salt	Many salts	Only one salt
test	Traditional DES [128/128 BS SSE2-16]					
JTR ver.	1.7.2-bp17-mpi7		1.7.2-bp17-mpi4		1.7.2-bp17-mpi	

Hodnoty naměřené na sestaveném clusteru (2 x Intel C2C@1.866 GHz) jsou Real 3548000 c/s (many salts) a Real 3016000 c/s (only one salt) jsou nejblíže sestavě Dual Core Intel Core2 Duo, 2.16GHz, MacBook Pro MacOSX. Pro 4 spuštěné procesy bylo dosaženo pro Tradiční DES Real 7115000 c/s (many salts) Real 6034000 c/s (only one salt), které jsou taktéž srovnatelné s tabulkou, konkrétně s Quad Core Intel Core2 Quad Q6600 @2304MHz.

[24] http://www.bindshell.net/tools/johntheripper
[25] http://openwall.info/wiki/john/benchmarks

7 Závěr

V této diplomové práci byl proveden rozbor OSS, který umožňuje využívat počítačové clustery. Byla zde prozkoumána problematika clusteringu a sestavování clusterů. Dále byl popsán OSS operační systém GNU/Linux, na kterém byly prováděny veškeré instalace a správa clusteru. Byly zhodnoceny obecné vlastnosti počítačového clusteru a vyzdviženy kladné i záporné vlastnosti. Clustery byly rozděleny podle hlavní funkce na pět základních odnoží. Pro každý typ byl popsán jeden, či více OSS, který umožňuje vytvořit a provozovat daný cluster v prostředí GNU/Linux. Byly zde také teoreticky rozebrány vlastnosti použitých benchmarků a algoritmů použitých pro paralelizaci.

Pro praktickou část byl vytvořen počítačový cluster složený 25 uzlů. Na tomto clusteru byl instalován software umožňující spouštět a zpracovávat paralelní výpočty. Postup sestavení, instalace a nastavení jednotlivých konfiguračních souborů je popsán v části 4.

Všechny testy byly provedeny jak na 32b a 64b OS. Výsledky byly téměř jednoznačně lepší při užití 64b OS pro architekturu procesorů x86_64. Při použití 32b OS na 64b CPU může být pokles výkonu obecně způsoben tím, do jaké míry dokáže kompilátor přizpůsobit překládanou aplikaci pro danou architekturu. Také je zřejmé, že pokud pracujeme s velkými čísli (větší než 32b), dochází ke zvýhodnění architektury x86_64 jelikož se čísla nemusí rozkládat na menší a tím vykonávat operace navíc.

Benchmarky byly provedeny na 100Mbps a 1Gbps lokální síti, kdy opět lépe vychází gigabitová sít, což vidíme hlavně na obrázku 16, kde není tak rapidní pokles výkonu.. Pro benchmark HPL bylo dosaženo maximálního výpočetního výkonu 99,2 Gflop/s viz. tabulka 3. V tabulce 5 vidíme, že účinnost clusteru dosahuje pouze 26,2 % teoretické hodnoty (372Gflop/s). Hlavním důvodem proč bylo dosaženo nízké účinnosti je nastavení parametrů v konfiguračním souboru HPL.dat. Ovlivnění sítě by mohlo zlepšit použití jiného algoritmu konkrétně změna Increasing-ring (modified) z na Long, který měl pracovat lépe s velkými nároky na přenosy dat. Správné nastavení hodnot je velmi složité a zabývá se jím i dokument [18].

Jako další testovací nástroj posloužil algoritmus PEA, který se zabývá výpočtem nelineárních dynamických stochastických modelů. Testy byly prováděny v programu OCTAVE a výsledky pro sestavený cluster (10 uzlů) jsou uvedeny v tabulce 7. Grafy 21 a 22

vycházející z této tabulky, znázorňují zrychlení výpočtu při použití paralelizace. Zrychlení výpočtů je nejvíce patrné pro spuštěných výpočetních 5 procesů a to až 80% v poměru k jednomu procesu. Při dalším zvyšování počtu zapojených procesů dochází k pomalému nárůstu až na 90%. Poměrně velký vliv měl použitý druh OS. Při OS pro architekturu i386 dochází k prodloužení doby výpočtu průmětně o 35%. Naopak patrné zlepšení nenastalo při použití 1Gbps LAN. Naměřené hodnoty můžeme srovnat s tabulkou 8 získanou z [22]. Vidíme zde jiné hodnoty časů z důvodu použití jiných procesorů, ale průběhy získané na sestaveném clusteru jsou srovnatelné s grafem 23.

V poslední části bylo mým cílem ukázat praktické využití clusteru. K tomu jsem použil program pro luštění přihlašovacích hesel s názvem John The Ripper. Tento program opatřený MPI patchem umožňující paralelní zpracování dat a tím využít potenciál výpočetního clusteru. Součástí programu je benchmark ukazující rychlost kombinaci znaku za sekundu. Bylo vybráno 5 základních šifrovacích či hashovacích algoritmů, konkrétně tradiční DES, Microsoft Hash, hrubá MD5, MD5 pro FreeBSD, a SHA1. V tabulkách 9 a 10 vidíme hodnoty JTR benchmarku pro OS i386 a x86_64. Pro tento benchmark nejsou změny výkonu při použití rozdílných OS natolik patrné jako HPL a PEA. Např. pro šifru DES na obrázku 24 je změna výkonu okolo 10%. Největší změnu výkonu byla patrná na obrázku 27 kde při výpočtu šifry SHA1 nejsou použity instrukce SSE2. V tomto případě došlo k poklesu výkonu o 40%.

Hodnoty naměřené při JTR benchmarku je možné srovnat s tabulkou 11, dostupnou na internetových stránkách komunity. Podobné hodnoty dosáhly procesory z rodiny Intel. Intel Core2 Duo, 2.16GHz, MacBook Pro MacOSX a Quad Core Intel Core2 Quad Q6600 @2304MHz.

8 Použitá literatura

[1] SLOAN, Joseph. High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI. [s.l.] : O'Reilly Media, Inc. , 2004. 368 s. ISBN 978-0596005702.

[2] LUCKE, Robert W. Building Clustered Linux Systems. 1st edition. [s.l.] : Prentice Hall, 2004. 648 s. ISBN 978-0-13-144853-7.

[3] TOP 500 supercomputer site [online]. c2000-2007 [cit. 2008-11-30]. Dostupný z WWW: <http://www.top500.org/>.

[4] Seti@home : Search for Extraterrestrial Intelligence [online]. 05/1999. c2008 [cit. 2008-11-30]. Dostupný z WWW: <http://setiathome.berkeley.edu/>.

[5] Sun Microsystems, Inc. Lustre File System [online]. c1994-2008 [cit. 2008-11-30]. Dostupný z WWW: <http://www.sun.com/lustre/>.

[6] ITSO IBM Redbooks. Introduction to Storage Area Networks [online]. [1998] , 07/2008 [cit. 2008-11-30]. SG24-5470-03.pdf. English. Dostupný z WWW: <http://www.redbooks.ibm.com/abstracts/sg245470.html?Open>. ISBN 0738495565.

[7] Open Cluster Group. Oscar Project : Open Source Cluster Application Resources [online]. c2000-2008 , 11/11/08 [cit. 2008-11-30]. Dostupný z WWW: <http://svn.oscar.openclustergroup.org/>.

[8] Rocks Core Development . Rocks Clusters [online]. 05/2000. [2003] [cit. 2008-11-30]. Dostupný z WWW: <http://www.rocksclusters.org>.

[9] Bay Area Beowulf User Group . Beowulf [online]. 1994. c2004-2007 [cit. 2008-11-30]. Dostupný z WWW: <http://www.beowulf.org/>.

[10] BAR, Moshe . The Open Mosix Project [online]. 2002. Amnon Barak, c2002-2008 , 02/29/2008 [cit. 2008-11-30]. Dostupný z WWW: <http://openmosix.sourceforge.net/>. Http://sourceforge.net/projects/openmosix/ .

[11] OpenMPI team. The Open MPI Project [online]. 2004. 2004-2008 , 25.11.2008 [cit. 2008-11-30]. English. Dostupný z WWW: < http://www.open-mpi.org>.

[12] Linux Virtual Server [online]. [1998] [cit. 2008-11-30]. Dostupný z WWW: <http://www.linuxvirtualserver.org>.

[13] DONGARRA, Jack, et al. Linpack : Linear algebra package [online]. 1965. [1998] [cit. 2008-11-30]. Dostupný z WWW: <http://www.netlib.org/linpack/>.

[14] BACKUS, John (IBM). FORTRAN : FORmula TRANslation [online]. 1957. [1990] , 11/14/99 [cit. 2008-11-30]. Dostupný z WWW: <http://www.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html>.

[15] PETITET, Antoine, et al. High-Performance Linpack Benchmark : HPL [online]. 09/2008. [1998] , 10/09/08 [cit. 2008-11-30]. Dostupný z WWW: <http://www.netlib.org/benchmark/hpl/>.

[16] Lapack : Linear algebra package [online]. February 29, 1992 . The University of Tennessee, c1992-2008 , November 18, 2008 [cit. 2008-11-30]. Dostupný z WWW: <http://www.netlib.org/lapack/>.

[17] BLACKFORD, L. S., et al. ScaLAPACK [online]. February 28, 1995 . Philadelphia, PA : Society for Industrial and Applied Mathematics, 1997 , April 5 2007 [cit. 2008-11-30]. Dostupný z WWW: <http://www.netlib.org/scalapack/scalapack_home.html>. ISBN 0-89871-397-8.

[18] HASHIZUME, Nobu. ARCHITECTURE AND PERFORMANCE OVERVIEW : Sun BluePts™ On-Line rin. TOKYO INSTITUTE OF TECHNOLOGY SUPERCOMPUTER GRID. 4150 Network Circle, Santa Clara, CA 95054 USA : Sun Microsystems, Inc, 2007. 33 s. Dostupný z WWW: <http://www.sun.com/blueprints/0207/820-0831.pdf>.

[19] IEEE Standard for Floating-Point Arithmetic. [s.l.] : [s.n.], 2008. 58 s. Dostupný z WWW: <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>. ISBN 978-0-7381-5753-5.

[20] Solar Designer. John the Ripper : password cracker [online]. 1.7.0.2. Free Software Foundation. , [1996-2008] [cit. 2009-04-13]. Dostupný z WWW: <http://www.openwall.com/john/doc/>.

[21] DONGARRA, Jack J. . Performance of Various Computers Using Standard. [s.l.] : [s.n.], 2009. 103 s. Dongarra@eecs.utk.edu. Dostupný z WWW: <http://www.netlib.org/benchmark/performance.ps>.

[22] CREEL, Michael. Using Parallelization to Solve a Macroeconomic Model: A Parallel Parameterized Expectations Algorithm.Computational Economics [online]. 2008, vol. 32, no. 4 [cit. 2009-04-20], s. 343-352. Dostupný z WWW: <http://www.springerlink.com/content/b2l43gg723781m87>. ISSN 0927-7099.

[23] Lecture Notes in Computer Science. Computational Science – ICCS 2006 [online]. 2006, vol. 3992 [cit. 2009-04-20], s. 518-525. Dostupný z WWW: <http://www.springerlink.com/link.asp?id=105633>. ISSN 0302-9743.

[24] FERNÁNDEZ, J, ANGUITA, M, ROS, E. SCE Toolboxes for the development of high-level parallel applications. Computational Science – ICCS 2006 [online]. 2006, vol. 3992 [cit. 2009-04-20]. Dostupný z WWW: <http://www.springerlink.com/content/h586kvrl0577/>. ISSN 0302-9743.

SEZNAM POUŽITÝCH ZKRATEK, VELIČIN A SYMBOLŮ

OSS – Open Source Software

GNU/ Linux – Volně dostupný operační system s jádrem Linux

GPL – General Public License

GNU – GNU Is Not Unix

HA – High Availability – vysoká dostupnost

NFS – Network File System

Flop/s – floating-point operations per second

Frontnode – hlavní uzel, server, dohlíží na ostatní uzly v clusteru

LB – Load Balancer

LVS – Linux Virtual Server

GFS – Global File System

DNS – Domain Name System

NIC – Network Interface Card

SAN – Storage Area Network

HPC – High Performance Computing

MPI – Message passing interface

OpenMPI – Volně dostupná implementace MPI

MPITB – MPI toolbox rozšíření pro program Octave

PEA – Parameterized Expectations Algorithm

DES – Data Encryption Standard – kryptografická šifra

MD5 – Message-Digest Algorithm 5

Intel C2D – Intel Core 2 Duo dvoujádrový procesor