EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA MEDZINÁRODNÝCH VZŤAHOV

Evidenčné číslo: 103004/I/2021/36124048426955012

TESTOVANIE EXEKUČNEJ VÝKONNOSTI PARALELNEJ APLIKÁCIE NA CLUSTRI

Diplomová práca

2021 **Bc. Peter Piliar**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA MEDZINÁRODNÝCH VZŤAHOV

TESTOVANIE EXEKUČNEJ VÝKONNOSTI PARALELNEJ APLIKÁCIE NA CLUSTRI

Diplomová práca

Študijný program: Informačný manažment
Študijný odbor: Ekonómia a manažment
Školiace pracovisko: Katedra aplikovanej informatiky FHI
Vedúci záverečnej práce: Ing. Mgr. Peter Schmidt, PhD.

Bratislava 2021 **Bc. Peter Piliar**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Piliar
Študijný program: informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor: ekonómia a manažment
Typ záverečnej práce: Inžinierska záverečná práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Testovanie exekučnej výkonnosti paralelnej aplikácie na clustri.

Anotácia: Záverečná práca opisuje postup vytvorenia clustera z aspoň 4 PC. Inštalácia a konfigurácia vybranej distribúcie OS. Vytvorenie paralelnej aplikácie na demonštrovanie výkonu. Praktická práca vyžadujúca prehľad v oblasti OS, hardvéru a DSD.

Vedúci: Ing. Mgr. Peter Schmidt, PhD.
Katedra: KAI FHI - Katedra aplikovanej informatiky FHI
Vedúci katedry: Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 28.10.2019

Dátum schválenia: 28.10.2019

Ing. Mgr. Peter Schmidt, PhD.
vedúci katedry

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: .....

(podpis študenta)

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcemu tejto záverečnej práce Ing. Mgr. Petrovi Schmidtovi, PhD. za nasmerovanie, pripomienky a rady, ktoré viedli k vytvoreniu tejto práce.

ABSTRAKT

PILIAR, Peter: *Testovanie exekučnej výkonnosti paralelnej aplikácie na clustri.* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Vedúci záverečnej práce: Ing. Mgr. Peter Schmidt, PhD. Bratislava: FHI, 2021, 56s.

Cieľom záverečnej práce je vytvorenie počítačového klastra, ktorý sa skladá zo štyroch počítačov. Práca podrobne popíše postup pri tvorbe klastra pre vysoko výkonné počítanie a pre spracovanie veľkých dát. Práca je rozdelená do troch kapitol. Prvá kapitola priblíži súčasný stav v problematike. V ďalšej kapitole zadefinuje cieľ práce a metodiky skúmania. Záverečná kapitola obsahuje postup pri tvorbe virtuálneho klastra a zhrnie výsledky získané spustením paralelných úloh na klastri. Výsledkom je porovnanie výkonu klastra.

Kľúčové slová: počítačový klaster, big data, vysoko výkonné počítanie, slurm

ABSTRACT

PILIAR, Peter: Testing the execution performance of a parallel application on a cluster. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. Advisor: Ing. Mgr. Peter Schmidt, PhD. Bratislava: FHI EU, 2021, 56p.

The goal of the master thesis is to create a computer cluster, which consists of four computers. The work describes the procedure for creating a high-performance cluster and big data processing cluster. The work is divided into three chapters. The first chapter describes the current state of the issue. In the next chapter we define the goal of the work and research methodology. The final chapter contains the procedure for creating a virtual cluster and summarizes the results obtained by running parallel tasks on the cluster. The result is a comparison of cluster performance.

Key words: computer cluster, big data, high performance computing, slurm

Obsah

Úvod

V dnešnej online dobe vytvára ľudstvo denne obrovské množstvo dát, ktoré je potrebné bezpečne ukladať a spracovať. Táto úloha je často pre jeden počítač nesplniteľná, alebo finančne náročná pretože by sme potrebovali špeciálny procesor, ktorý by dokázal spracovať obrovské množstvo inštrukcií za sekundu. Spojenie viacerých počítačov do jedného celku, teda klastra, ponúka výkon, ktorý už je schopný takéto úlohy riešiť. Použitie počítačového klastra nám zabezpečí vysokú dostupnosť našich serverov, redundanciu dát, možnosť jednoduchej škálovateľnosti.

Na vytvorenie výpočtového klastra sa používa dnes bežne dostupná technika, väčšinou ide o spojenie procesorov a grafických kariet. Vďaka spojeniu týchto výkonných výpočtových procesorov môžu dnešné superpočítače vykonávať až 537 x 10^{15} operácií za sekundu. Takýto výkon je možné použiť na výpočet náročných úloh z rôznych oblastí ako napríklad kvantovej mechaniky, predpovede počasia, výskumu klímy, prieskumu ropy a plynu, molekulárneho modelovania a fyzikálnych simulácií (simulácia začiatku vesmíru, aerodynamiky lietadiel a kozmických lodí, detonácie jadrových zbraní). Superpočítače boli použité aj pri výskume látok, ktoré by mohli potenciálne zastaviť šírenie COVID-19. [1] Využitie počítačových klastrov je na riešenie zložitých vedeckých problémov a pre vývoj spoločnosti významným prvkom, preto je táto téma pre autora zaujímavá.

Práca v prvej kapitole priblíži aktuálny stav v problematike počítačových klastrov. Druhá kapitola zadefinuje cieľ práce a metodiku skúmania a na záver v tretej kapitole popíše postup pri vytváraní počítačového klastra a demonštruje jeho výpočtový výkon.

1 Súčasný stav problematiky doma a v zahraničí

Táto kapitola čitateľovi priblíži problematiku počítačových klastrov v súčasnej dobe. Popíšeme typy počítačových klastrov, ktoré sa dnes bežne používajú v praxi. Pozrieme sa na hardvérové vybavenie klastrov a tiež na softvér, ktorý sa používa na spravovanie a plánovanie úloh v klastroch. Predstavíme Linux ako najčastejšie používaný operačný systém v počítačových klastroch. A na záver sa pozrieme na možnosti vytvorenia virtuálnej verzie počítačového klastra.

1.1 Počítačový klaster

Počítačový klaster je skupina voľne alebo tesne prepojených počítačov, ktoré spolu spolupracujú na vykonávaní úlohy, takže sa z mnohých hľadísk dajú považovať za jeden systém. Klastre sú používané na zvýšenie rýchlosti alebo spoľahlivosti s väčšou efektivitou akú by mohol poskytnúť jeden počítač, ale pritom sú lacnejšie ako jediný počítač s porovnateľnou rýchlosťou alebo spoľahlivosťou.

Klastre slúžia k paralelným výpočtom zložitých výpočtových úloh (napr. faktorizácia, t.j. rozklad na činitele, generovanie prvočísel, simulácia vývoja počasia, analýza veľkého množstva dát, atď.), alebo sa používajú na zaistenie vysokej dostupnosti určitej služby (napr. databázy, SMS centra, atď.). Používajú sa buď špecializované viacprocesorové stroje prepojené cez sieť alebo obyčajné počítače triedy PC. Úlohy, určené pre urýchlenie výpočtu pomocou výpočtového klastra musia byť navrhnuté špeciálne pre tento účel.

Jednotlivé počítače v klastri sa nazývajú uzly (node) a zvyčajne sú prepojené pomocou rýchlych lokálnych sietí LAN. Každý uzol má vlastnú systémovú pamäť, vlastný operačný systém, databázovú inštanciu a aplikačný software. Medzi sebou zdieľajú len spoločné dáta pomocou zdieľaného diskového subsystému.

Počítačové klastre vznikli v dôsledku konvergencie mnohých výpočtových trendov vrátane dostupnosti lacných mikroprocesorov, vysokorýchlostných sietí a softvéru pre vysoko výkonné distribuované výpočty. Majú širokú škálu uplatniteľnosti a nasadenia, od klastrov malých firiem s niekoľkými uzlami až po niektoré z najrýchlejších superpočítačov na svete, ako napríklad IBM Summit. [2] Pred príchodom klastrov boli použité sálové počítače odolné voči

poruchám jednej jednotky s modulárnou redundanciou; ale nižšie počiatočné náklady na klastre a vyššia rýchlosť sieťovej infraštruktúry podporili rozvoj klastrov.

Klastre nám umožňujú jednoduché zväčšovanie systémového výkonu. Redundantné softwarové a hardwarové komponenty zvyšujú dostupnosť systému pre používateľov a jeho odolnosť voči výpadkom. Tieto charakteristiky sú spoločné pre všetky klastrované systémy. Jednotlivé systémy sa podľa ich potrieb líšia svojou architektúrou a implementáciou.

Klastrovanie bolo na začiatku takmer výlučne synonymom potreby poskytovať škálovateľné riešenia pre rastúci podnik. Napríklad v roku 1997 bol postavený klaster „Deep Blue“ zoskupením počítačov IMB RS/6000, ktorý poskytol dostatočný výpočtový výkon aby ako prvý počítač v histórii porazil vtedajšieho šachového veľmajstra Garyho Kasparova.[3]

V dnešnej dobe je pri použití, (resp. voľbe) klastra škálovateľnosť samozrejmosťou a medzi hlavné prínosy klastra patria vysoká dostupnosť a rýchle odpovede (vyvažovanie záťaže).

Vysoká dostupnosť: je potrebná pre základné systémy a ich webové front end servery, ktoré musia byť neustále v prevádzke. To je miesto, kde je klastrovanie vynikajúcim riešením. Klastrovanie poskytuje transparentné zálohovanie, prepnutie po zlyhaní a poskytuje redundanciu systémov, periférií a dát. To všetko je dôležité pre udržanie funkčných informačných systémov 24x7x365.

Integrácia servera: Rozsiahle webové obchodné prostredie tiež znamená, že podniky už nemôžu očakávať, že jediný počítač vyrieši všetky ich potreby na spracovanie dát, aj keď sú na trhu dostupné vysoko-výkonné servery, ktoré by zvládli spracovať celú spoločnosť. Mnoho aplikácií ale vyžaduje, aby boli prostriedky servera vylepšené rôznymi spôsobmi, tak aby sa dosiahli rôzne účely. Spravidla existujú tri triedy pracovných záťaží, ktoré vyžadujú inú dynamiku servera:

- Servery dátových transakcií riadia základné obchodné procesy
- Webové aplikačné servery spravujú dojem koncového používateľa
- Servery zariadení spravujú špecifické sieťové funkcie.

Všetky aplikácie, ktoré bežia v týchto rôznych triedach pracovných záťaží (a ktoré sú preto pravdepodobne implementované na samostatných serveroch), musia navzájom pristupovať, zdieľať a aktualizovať svoje údaje v reálnom čase.

Je tiež pravda, že funkcie, ktoré sú tradične priamo pripojené k jednotlivým serverom, ako sú sieťové pripojenia a úložiská, sú nasadené centrálne, aby poskytovali efektívnejšie využitie, ľahšiu škálovateľnosť a väčšiu flexibilitu.

Klastrovanie elegantne rieši problém integrácie týchto troch typov serverov spolu s centralizovanými sieťovými a úložnými prostriedkami. Ďalšou výhodou je umožnenie informačnému systému spravovať tieto samostatné servery, akoby boli jedným.

1.2 Paralelné výpočty

Paralelné výpočty (ang. parallel computing) je v informatike označenie pre výpočty, ktoré sú riešené súbežne (paralelne). Paralelizácia je využívaná pre zvýšenie výpočtového výkonu v situácii, keď nie je možné použiť rýchlejší počítač (vyššiu frekvenciu procesora) alebo pre zjednodušenie použitého algoritmu alebo pre lepšie využitie elektrickej energie. Paralelné výpočty sa môžu realizovať pomocou viacprocesorových systémov alebo spustením úlohy na počítačovom klastri, kde sú jednotlivé počítače prepojené pomocou siete. Paralelizácia funguje na princípe rozdeľovania zložitejších úloh na jednoduchšie, ktoré sú spracovávané súbežne a je považovaná za náročnejšiu formu programovania ako sekvenčné programovanie. Paralelné programovanie prináša niekoľko nových typov potenciálnych softvérových chýb, z ktorých najčastejšou je súbeh (race condition), ku ktorej môže dôjsť v dôsledku toho, že aplikácia predpokladá, že určité operácie prebehnú v určitom poradí. Komunikácia a synchronizácia medzi rôznymi čiastkovými úlohami sú zvyčajne jednou z najväčších prekážok dosiahnutia optimálneho výkonu paralelného programu. [4]

1.3 Hardvér pre výpočty

CPU a GPU majú veľa spoločného. Oba sú kritické výpočtové moduly. Oba sú mikroprocesory na báze kremíka. A oba spracúvajú údaje. Ale CPU a GPU majú odlišnú architektúru a sú postavené na rôzne účely.

CPU je vhodný pre širokú škálu pracovných záťaží, najmä pre tie, pre ktoré je dôležitá latencia alebo výkon na jadre. CPU je výkonný procesor, ktorý zameriava svoj menší počet jadier na jednotlivé úlohy a na rýchle vykonávanie úloh. Vďaka tomu je jedinečne vybavený pre rôzne úlohy ako sú sériové výpočty a správa databázy.

Vďaka svojmu jednoúčelovému dizajnu sú jadrá GPU oveľa menšie ako jadrá pre CPU, takže GPU majú tisíce jadier, zatiaľ čo CPU max. 32. Vďaka až 5 000 jadrám, ktoré sú k dispozícii pre jednu úlohu, sa tento dizajn hodí na masívne paralelné spracovanie .

V minulosti sa paralelné počítanie realizovalo s veľkým počtom procesorov, typu x86, ktoré boli veľmi drahé a ťažko sa programovali. Dnes GPU ako jednoúčelový procesor ponúka oveľa väčšiu výpočtovú hustotu a využíva sa pri mnohých úlohách matematickej akcelerácie. Používanie GPU v dátovom centre začalo s internými aplikáciami vďaka jazyku vyvinutému spoločnosťou Nvidia s názvom CUDA. CUDA umožňuje vývojárom softvéru a softvérovým inžinierom používať grafickú jednotku na účely všeobecného spracovania tak ako CPU. Je to softvérová vrstva, ktorá poskytuje priamy prístup k sade virtuálnych inštrukcií GPU a je navrhnutá na prácu s programovacími jazykmi ako C, C ++ a Fortran.

Keď sa zlepšil výkon GPU a procesory sa ukázali ako životaschopné aj pre iné ako hráčske úlohy, začali k nim pribúdať aj rôzne aplikácie. Na trh sa dostali desktopové aplikácie, ako napríklad Adobe Premier, ale aj serverové aplikácie vrátane databáz SQL. GPU je ideálne vhodný na urýchlenie spracovania dotazov SQL, pretože SQL vykonáva rovnakú operáciu - zvyčajne vyhľadávanie - v každom riadku množiny. GPU môže tento proces paralelizovať priradením riadku údajov k jednému jadru.

Spoločnosti Brytlyt, SQream Technologies, MapD, Kinetica, PG-Strom a Blazegraph ponúkajú vo svojich databázach analytiku akcelerovanú pomocou GPU. Spoločnosť Microsoft nepodporuje akceleráciu GPU na serveri SQL Server.

Grafické procesory (GPU) sa už mnoho rokov používajú na zobrazovanie obrázkov a vykresľovanie pohybu na displejoch počítačov. Grafické procesory GPU sú dnes dosť výkonné na to, aby zvládali viac ako len pohyb obrázkov na obrazovke. Referenčné hodnoty zamerané na aritmetiku s pohyblivou rádovou čiarkou, ktoré sa najčastejšie používajú v týchto inžinierskych výpočtoch, ukazujú, že GPU môžu vykonávať takéto výpočty oveľa rýchlejšie

ako tradičné jednotky centrálneho spracovania (CPU) používané na dnešných pracovných staniciach - niekedy až 20-krát rýchlejšie, v závislosti od výpočtu.

GPU si tiež našli domov v HPC, kde veľa úloh, ako sú simulácie, finančné modelovanie a 3D vykresľovanie, funguje dobre aj v paralelnom prostredí. Podľa spoločnosti Intersect 360, zaoberajúcej sa prieskumom trhu, ktorá sleduje trh HPC, 34 z 50 najpopulárnejších aplikačných balíkov HPC ponúka podporu GPU vrátane všetkých 15 najlepších aplikácií HPC. Patria sem chemické aplikácie GROMACS, Gaussian a VASP, ANSYS a OpenFOAM pre dynamiku tekutín, Simulia Abaqus pre štrukturálnu analýzu a WRF pre modelovanie počasia / prostredia.

Dnes najrýchlejší superpočítač od Fujitsu Fugoku používa výlučne špeciálne navrhnuté CPU. Stále však v liste Top 500 superpočítačov dominuje kombinácia CPUs a GPUs. [2]

1.4 Vysoko výkonné počítanie (HPC)

Moderné vedecké a priemyselné problémy napr. v medicíne (genetika, „in silico“ dizajn liečiv), vo fyzike (meteorologické a klimatické modely, jadrová a časticová fyzika), v chémii (vlastnosti atómov a molekúl, korelácia medzi štruktúrou a reaktivitou) ale aj v ekonomike (risk investícii, rast akcií) vyžadujú vysokú výpočtovú výkonnosť. Práve na riešenie takýchto úloh sa používa High performance computing (HPC) alebo vysoko výkonné počítanie (VVP).

Najznámejšou implementáciou VVP je superpočítač. Za superpočítače sú považované systémy, ktoré svojím výkonom vysoko prevyšujú bežné, osobné počítače. Na meranie výpočtového výkonu sa používa jednotka výpočtového výkonu, takzvaný Flop (počet operácii s plávajúcou desatinnou čiarkou (floating point) za sekundu). Moderné superpočítače dosahujú výkony na úrovni terraFlopov (10^{12}) až petaFlopov (10^{15}). Agregátny výpočtový výkon je ovplyvnený aj ďalšími parametrami, akými sú oneskorenie a priepustnosť dátových sietí, prostredníctvom ktorých procesory alebo uzly superpočítača komunikujú, oneskorenie a priepustnosť komunikácie s úložiskom dát a pod. Všetky tieto parametre determinujú škálovateľnosť, čiže efektivitu tzv. paralelných výpočtov využívajúcich viac uzlov, procesorov resp. jadier súčasne. [5]

Trendom v modernom VVP je už spomenuté paralelné počítanie. V súčasnosti už jasne dominuje viacuzlová, viacprocesorová resp. viacjadrová architektúra superpočítačov, pričom jednotlivé výpočtové jednotky spolu komunikujú a ako celok poskytujú agregovaný výkon. Ich

vzájomne prepojenie je realizované vysoko rýchlostnou sieťou, ako je napr. Infiniband alebo 10Gb/s ethernet, resp. inou dedikovanou sieťou. Architektúry takéhoto typu je možné (aspoň teoreticky) škálovať do obrovských rozmerov, pričom limitujúcimi faktormi sú spotreba elektrickej energie, chladenie a priestorové možnosti miesta inštalácie.

Vysokovýkonné počítanie umožňuje vedeckým pracovníkom získavať nové poznatky, podporuje inovácie a pomáha pri dosahovaní prelomových objavov. Vďaka efektívnemu spracovaniu komplexných dát pomáha HPC zvyšovať naše chápanie zložitých vedeckých problémov a vytvárať tak inovácie zvyšujúce konkurencieschopnosť podnikov, efektivitu verejnej správy a úroveň vedy a výskumu.

S vývojom technológií ako internet vecí (IoT), umelá inteligencia (AI) a 3D zobrazovanie exponenciálne rastie veľkosť a množstvo dát, s ktorými musia organizácie pracovať. Schopnosť spracovávať údaje v reálnom čase je na mnohé účely, ako napríklad vysielanie živých športových udalostí, sledovanie rozvíjajúcej sa búrky, testovanie nových produktov alebo analýzu trendov na sklade.

1.5 Spracovanie Big data (veľkých dát)

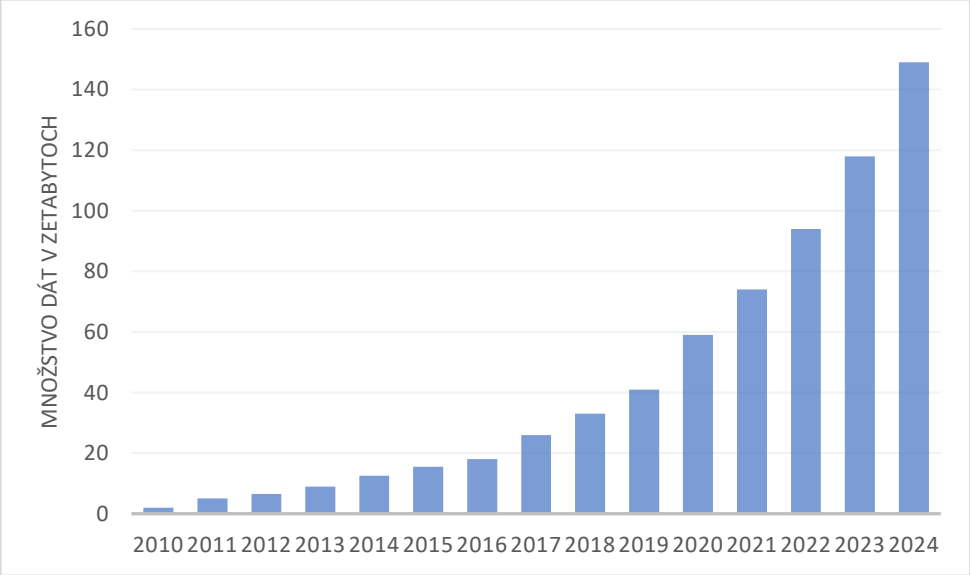
Big data (slovensky veľké dáta, ale často sa v slovenskej literatúre používa anglický názov big data) sú oblasťou, ktorá sa zaoberá spôsobmi analýzy, systematického získavania informácií alebo spracovávania súborov údajov, ktoré sú príliš veľké alebo zložité na to, aby ich bolo možné zvládnuť tradičným aplikačným softvérom na spracovanie údajov. Z technického hľadiska sa Big data spracovávajú práve v počítačových klastroch, ktoré poskytujú potrebný dátový prietok a výpočtový výkon.

Medzi veľké výzvy v oblasti analýzy údajov patrí zber údajov, ukladanie údajov, analýza údajov, vyhľadávanie, zdieľanie, prenos, vizualizácia, dopytovanie, aktualizácia, ochrana údajov a zdroj údajov. Big data boli pôvodne spojené s tromi kľúčovými konceptmi: objemom, rozmanitosťou a rýchlosťou. [6] Big data preto často zahŕňajú dáta s veľkosťami, ktoré presahujú kapacitu tradičného softvéru na spracovanie v prijateľnom čase a hodnote.

Súčasné používanie pojmu big data zvykne odkazovať na použitie prediktívnej analýzy, analýzy správania používateľa alebo určitých ďalších pokročilých metód analýzy údajov, ktoré extrahujú hodnotu z veľkých údajov a zriedka na konkrétnu veľkosť súboru údajov. Vedci,

riadiaci pracovníci, lekári, marketingoví analytici a vládni analytici sa pravidelne stretávajú s problémami s rozsiahlymi súbormi údajov v oblastiach, ako sú internetové vyhľadávanie, fintech, analytika v zdravotníctve, geografické informačné systémy, mestská informatika a obchodná informatika.

Veľkosť a počet dostupných súborov údajov rýchlo vzrástol, pretože údaje sa zhromažďujú prostredníctvom zariadení, ako sú mobilné zariadenia, lacné a početné zariadenia IoT snímajúce informácie, antény (diaľkový prieskum), softvérové protokoly, fotoaparáty, mikrofóny, rádiofrekvenčná identifikácia (RFID) čítačky a bezdrôtové siete senzorov. [7] Technologická kapacita pre ukladanie informácií na obyvateľa sa od osemdesiatych rokov zhruba každých 40 mesiacov zdvojnásobila. [8] Na základe predpovede správy IDC sa predpovedalo, že globálny objem dát exponenciálne vzrastie zo 4,4 zettabytov na 44 zettabytov v rokoch 2013 až 2020. Do roku 2025 IDC predpovedá 163 zettabytov dát. [9] Tento rast je zobrazený na obrázku č.1.



Obrázok 1: Rast množstva uložených dát [31]

Relačné systémy na správu databáz a desktopové štatistické softvérové balíčky používané na vizualizáciu údajov majú často ťažkosti so spracovaním a analýzou veľkých dát. Spracovanie a analýza veľkých dát môže vyžadovať „masívne paralelný softvér bežiaci na desiatkach, stovkách alebo dokonca tisícoch serverov“. [10] To, čo sa kvalifikuje ako „veľké dáta“, sa líši v závislosti od schopností osôb, ktoré ich analyzujú, a ich nástrojov. Vďaka rozširujúcim sa

schopnostiam sa big data stávajú pohyblivým cieľom. „Pre niektoré organizácie môže prvé stretnutie so stovkami gigabajtov dát vyvolať potrebu prehodnotiť možnosti správy údajov. U iných môže trvať desať alebo stovky terabajtov, kým sa veľkosť dát stane významným faktorom.“ [11]

Big Data sú najčastejšie charakterizované ako 3V a to:

- Objem (Volume) - množstvo vygenerovaných a uložených údajov. Podľa veľkosti údajov môžeme určiť ich hodnotu a potenciálny prehľad, ktorý z nich získame a to, či sa dajú považovať za big data alebo nie. Veľkosť pri big data sa zvyčajne pohybuje v terabajtoch až petabajtoch.
- Rôznorodosť (Variety) - typ a povaha údajov. Predchádzajúce technológie ako RDBMS boli schopné efektívne spracovávať štruktúrované údaje. Avšak zmena typu a povahy zo štruktúrovanej na pološtruktúrovanú alebo neštruktúrovanú spochybnila existujúce nástroje a technológie. Technológie veľkých dát sa vyvinuli s hlavným zámerom zachytiť, uložiť a spracovať pološtruktúrované a neštruktúrované (rozmanité) údaje generované vysokou rýchlosťou a obrovskou veľkosťou.
- Rýchlosť (Velocity) - rýchlosť akou sa údaje generujú a spracúvajú, aby splnili požiadavky a výzvy, ktoré stoja na ceste rastu a rozvoja. Big data sú často k dispozícii v reálnom čase. V porovnaní s malými dátami sa veľké dáta vytvárajú kontinuálnejšie. Dva druhy rýchlosti súvisiace s veľkými dátami sú frekvencia generovania a frekvencia manipulácie, zaznamenávania a publikovania.

1.5.1 Nástroje Big Data

Jedným z hlavných priekopníkov v odvetví veľkých dát je Google. V začiatkoch Google, bola celá databáza webu uložená na desiatich štvorgigových diskoch (dokopy teda 40 GB, čo je asi desatina kapacity bežných, dnes predávaných diskov do PC). Web sa však začal prudko rozrastať a vyhľadávanie začalo narážať na technické limity vtedajších databáz. Bolo potrebné vymyslieť úplne novú technológiu, ktorá by umožnila rýchle a za prijateľnú cenu spracovať takéto dátové objemy. V roku 2004 Google publikoval vedeckú prácu opisujúcu metódu nazvanej MapReduce a ďalšiu v roku 2006 o spôsobe ukladania veľkých dát nazvanom BigTable.

Základnou vlastnosťou týchto dvoch inovácií bolo, že umožňovali masívne paralelné spracovanie údajov. To znamená, že údaje sa rozdelia medzi množstvo počítačov (tisíce až desaťtisíce) a každý počítač spracuje len malú časť údajov a výsledky sa potom zosumarizujú. Toto je to tajomstvo, vďaka ktorému Google dokáže naraz obslúžiť taký veľký počet užívateľov, pričom zakaždým musí zároveň expresne spracovať obrovský objem dát.

Samozrejme big data, to nie je dnes len Google. Okrem vyhľadávania na internete a spracovania údajov zo sociálnych sietí existuje množstvo rôznych ďalších aplikácií spracovania veľkých dát. Napríklad marketingové aplikácie na segmentáciu zákazníkov, analýzu zákazníckeho správania a cielenú reklamu, ktoré kombinujú dáta z predaja a zákazníckych databáz alebo aplikácie na optimalizáciu business procesov, ktoré kombinujú dáta z rôznych zariadení, snímanie RFID čipov, počítačových logov a ďalšie. [12]

1.5.2 Lucene

Nástroj Lucene beží na licencii Apache a ponúka možnosť vysoko výkonnostného indexovania. Dokáže indexovať až 150 GB za hodinu s minimálnou záťažou na RAM pamäť. Medzi jeho možnosti patrí i vyhľadávanie podľa priority, najlepšie výsledky sú vždy prvé. Umožňuje používanie vyhľadávania fráz, s pomocou divokej karty, podobnosti, rozsahu dát, prehľadávanie polí, či viacnásobné indexové prehľadávanie so zlúčenými výsledkami. Algoritmus dokáže brať do úvahy i preklepy a zapracuje ich do výsledkov. [13]

1.5.3 Apache Solr

Apache Solr je nadstavba Lucene, ktorá dokáže fungovať ako fulltextový vyhľadávací server, pri vysokej záťaži. Umožňuje dynamické klastrovanie, indexovanie v skutočnom čase i vyhľadávanie podľa geolokačných údajov. Tento systém má prívetivé rozhranie, je odolný voči chybám, dokáže balansovať záťaž na server, má centralizovanú konfiguráciu, umožňuje monitorovanie, škálovanie a má sadu pluginov, ktoré dokážu indexovať i PDF, či Word súbory, rozoznávať jazyk, parsovať dokumenty a mnoho ďalšieho. Tento vyhľadávací server je populárny medzi stránkami s obrovskou záťažou, medzi ktoré patria i eBay, Netflix, Disney, MTV Networks, Adobe a iné. Existuje ešte nadstavba nad Solr, ktorá namiesto HTTP využíva JSON. Nazýva sa Elasticsearch a dokáže pracovať až na 100 zariadeniach zároveň. Dokáže pracovať naprieč niekoľkými indexmi, ktoré sú rozdelené na menšie časti a rozdistribuované do

vyhľadávacích uzlov. Vďaka tomu je rýchly a nenáročný na pamäť. Túto možnosť využívajú obrovské siete ako napríklad LinkedIn, či Wikipédia. [13]

1.5.4 Hadoop

Hadoop je doposiaľ najrozšírenejší opensource systém pre ukladanie veľkých dát. Umožňuje obrovský rozsah škálovateľnosti a poskytuje obrovské množstvo úložného priestoru pre akýkoľvek typ dát. Používanie big data klastrov s technológiou Hadoop je viac ako dvadsaťkrát lacnejšie oproti používaniu relačných databáz. Už pri množstve dát 10 TB dokáže táto technológia ušetriť 200 až 400 tisíc eur pri vývoji. Tieto klastre teda jednoznačne šetria zdroje, avšak na dolovanie dát a niektoré typy analýz môžu byť neefektívne. Ukladanie dát v relačných databázach je drahé ale práca s BI nástrojmi je potom oveľa jednoduchšia. Z tohto dôvodu Hadoop nie je náhradou existujúcich relačných skladov, ale ponúka alternatívu na ukladanie veľkého množstva dát. Na trhu sú viaceré komerčné riešenia, ktoré stavajú na systéme Hadoop a ponúkajú ho v upravenej forme, ktorá je viac používateľsky prívetivá. Medzi tieto riešenia patria produkty od Hortonworks, Rmap, Cloudera, IBM a ďalších. [13]

1.6 Linux ako OS pre klastre

Linux je skupina operačných systémov UNIXového typu, ktoré sú založené na linuxovom jadre. Jadro Linuxu je open source a tvorí základ operačných systémov, ktoré ho rozširujú ďalšími knižnicami a nástrojmi s rôznou funkcionalitou. Mnohé používané knižnice sú vyvíjané pod projektom GNU a spolu s Linux jadrom sa označujú ako GNU/Linux.

Medzi najpopulárnejšie linuxové distribúcie patria Debian, Fedora, Ubuntu a tiež aj komerčné riešenia Red Hat Enterprise Linux a SUSE Linux Enterprise Server. [14] Desktopové distribúcie pridávajú systém okien X11 alebo Wayland a grafické používateľské rozhranie ako napríklad GNOME alebo KDE Plasma. Serverové distribúcie tieto grafické prvky neobsahujú. Vďaka voľnej dostupnosti zdrojového kódu Linuxu, môže ktokoľvek vytvoriť distribúciu operačného systému prakticky na akýkoľvek účel.

Pôvodne bol Linux vyvinutý ako operačný systém pre osobné počítače na Intel x86 architektúre, ale od vtedy sa rozšíril na viac platforiem ako akýkoľvek iný operačný systém. [15] Kvôli dominancii systému Android v smartfónoch, ktorý je tiež založený na linuxovom jadre má systém Linux najväčší počet inštalácií zo všetkých univerzálnych operačných

systémov. [16] Aj keď je používaný iba na okolo 2.3 percentách stolových počítačov [17], je ale najpoužívanejším operačným systémom na serveroch (približne 74.2% z 10 milióna webových stránok beží na Linuxe). [18] Linux vedie aj v sálových počítačoch a je jediným operačným systémom používaným v superpočítačoch TOP500 (od 2017 eliminoval všetkých konkurentov).

1.7 Softvérové vybavenie klastra

V tejto podkapitole predstavíme softvérové riešenia, ktoré sú navrhnuté pre klastrové počítanie. Ide o knižnice ktoré poskytujú funkcionalitu potrebnú na spracovanie veľkých dát, paralelné počítanie, správu a plánovanie úloh a manažment výpočtových zdrojov.

1.7.1 MPI

Rozhranie na prenos správ (MPI – message passing interface) je štandard na prenos správ, ktorý navrhla skupina výskumníkov z akademickej obce a priemyslu tak, aby fungoval na širokej škále architektúr pre paralelné výpočty. Norma definuje syntax a sémantiku jadra knižničných rutín užitočných pre širokú škálu používateľov, ktorí píšu prenosné programy na posielanie správ v jazykoch C, C ++ a Fortran. Existuje niekoľko dobre otestovaných a efektívnych implementácií MPI, z ktorých mnohé sú open-source alebo verejné. Tieto podporili rozvoj paralelného softvérového priemyslu a podporili vývoj prenosných a škálovateľných rozsiahlych paralelných aplikácií.

Špecifikácia MPI definuje knižnicu podprogramov, ktoré obsahujú komunikačné funkcie na prenos údajov medzi procesormi, funkcie vykonávajúce kolektívne operácie nad nejakou množinou procesorov, a mnohé iné funkcie, ktoré sa zaoberajú prenosom správ a dynamickým vytváraním nových procesov.

MPI je komunikačný protokol na programovanie paralelných počítačov. Podporovaná je komunikácia typu point-to-point a aj kolektívna komunikácia. MPI „je rozhranie pre programovanie aplikácií na odovzdávanie správ spolu s protokolom a sémantickými špecifikáciami toho, ako sa musia jeho funkcie správať pri akejkoľvek implementácii.“ [20] Cieľom MPI je vysoký výkon, škálovateľnosť a prenosnosť. MPI zostáva dominantným modelom používaným v súčasnosti vo vysoko výkonných počítačoch. [21]

Medzi funkcie knižnice MPI patria okrem iného operácie odosielania / prijímania typu point-to-point, výber medzi karteziánskou alebo grafickou topológiou logického procesu, výmena údajov medzi dvojicami procesov (operácie odosielania a prijímania), kombinovanie čiastočných výsledkov výpočtov (zhromažďovanie a znižovanie počtu operácií), synchronizácia uzlov, ako aj získavanie informácií týkajúcich sa siete, ako je počet procesov vo výpočtovej relácii, aktuálna identita procesora, na ktorú je proces mapovaný, susedné procesy prístupné v logickej topológií atď.

Väčšina implementácií MPI pozostáva zo špecifickej sady rutín priamo vyvolaných z jazykov C, C ++, Fortran (t. j. API) a z ľubovoľného jazyka schopného prepojiť sa s takýmito knižnicami, vrátane C #, Java alebo Python. Výhodou MPI oproti starším knižniciam na prenos správ je prenositeľnosť (pretože MPI je implementovaná takmer pre každú architektúru distribuovanej pamäte) a rýchlosť (pretože každá implementácia je v zásade optimalizovaná pre hardvér, na ktorom beží).

V súčasnosti má tento štandard niekoľko verzií: verzia 1.3 (bežne označovaná skratkou MPI-1), ktorá kladie dôraz na odovzdávanie správ a má statické runtime prostredie, MPI-2.2 (MPI-2) obsahuje nové funkcie, ako sú paralelné I/O operácie, dynamické riadenie procesov a operácie vzdialenej pamäte a MPI-3.1 (MPI-3), ktorá zahŕňa rozšírenia kolektívnych operácií s neblokujúcimi verziami a rozšírenia jednostranných operácií. [22]

1.7.2 OpenMPI

Open MPI je open source implementácia rozhrania posielania správ, ktorá je vyvíjaná a udržiavaná konzorciom akademických, výskumných a priemyselných partnerov. Open MPI je preto schopný kombinovať odborné znalosti, technológie a zdroje z celej komunity pracujúcej s vysokovýkonnými počítačmi s cieľom vybudovať najlepšiu dostupnú knižnicu MPI. Open MPI ponúka viaceré výhody pre dodávateľov systémov a softvéru, vývojárov aplikácií a výskumných pracovníkov v oblasti informatiky.

Medzi funkcie implementované v Open MPI patria:

- Plná zhoda s normami MPI-3.1
- Bezpečnosť vlákna a súbežnosť
- Dynamické vytváranie procesov
- Sieťová a procesná odolnosť voči chybám
- Podpora heterogenity siete
- Jedna knižnica podporuje všetky siete
- Podporovaných je mnoho plánovačov úloh
- Mnoho podporovaných OS (32 a 64 bitov)
- Vysoký výkon na všetkých platformách
- Prenosný a udržiavateľný kód
- Dizajn založený na komponentoch, zdokumentované API
- Licencia open source založená na licencii BSD. [23]

1.7.3 MapReduce

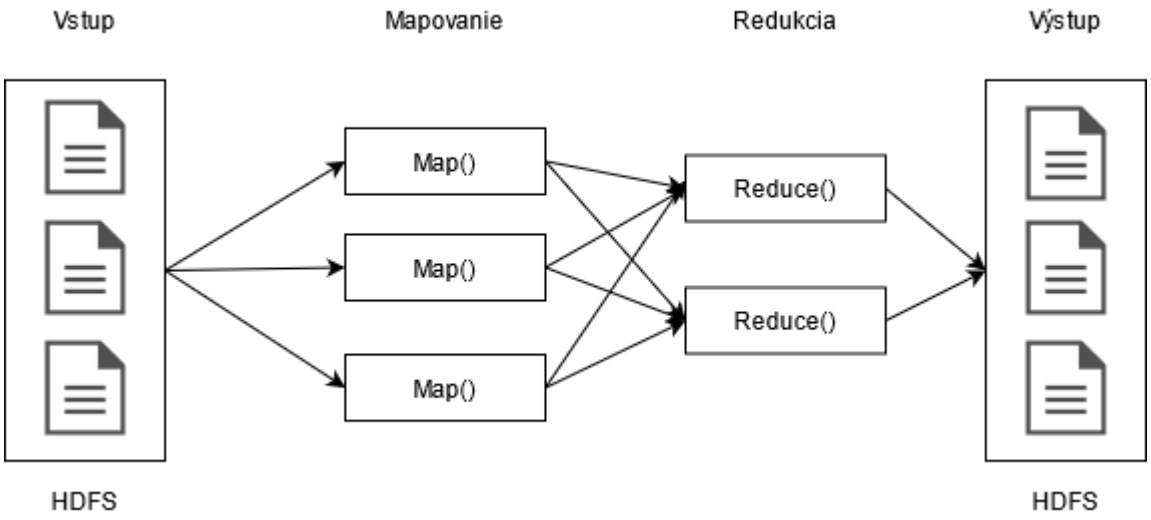
MapReduce je programovací model a súvisiaca implementácia na spracovanie a generovanie súborov veľkých dát pomocou paralelného distribuovaného algoritmu v klastri. [24]

Program MapReduce sa skladá z mapovacej procedúry, ktorá vykonáva filtrovanie a triedenie (napríklad triedenie študentov podľa mena do radov, jeden rad pre každé meno), a z metódy redukcie, ktorá vykonáva súhrnnú operáciu (napríklad spočítanie počtu študentov v každej rade, čím sa získa frekvencia mien). Celý „MapReduce systém“ (tiež nazývaný „infraštruktúra“ alebo „framework“) organizuje spracovanie dát prostredníctvom riadenia distribuovaných serverov, paralelného spúšťania úloh, správy všetkej komunikácie a dátových prenosov medzi rôznymi časťami systému, zabezpečením redundantnosti a odolnosti voči chybám.

Tento model je špecializáciou stratégie split-apply-combine na analýzu dát. Je inšpirovaný funkciami mapovania a redukcie, bežne používaných vo funkčnom programovaní, aj keď ich účel v rámci MapReduce nie je rovnaký. Kľúčovým prínosom rámca MapReduce nie je samotná funkcia mapovania a redukcie (ktoré existujú aj v štandarde MPI),

ale škálovateľnosť a odolnosť proti chybám. Implementácia MapReduce s jedným vláknom ako taká zvyčajne nie je rýchlejšia ako tradičná implementácia (bez MapReduce) a akékoľvek zisky sa zvyčajne prejavia iba pri implementácii viacerých vlákien na viacprocesorovom hardvéri.

Knižnice MapReduce boli napísané v mnohých programovacích jazykoch s rôznymi úrovňami optimalizácie. Populárna implementácia open-source, ktorá podporuje distribuované zmiešavanie dát, je súčasťou Apache Hadoop. Názov MapReduce pôvodne odkazoval na patentovanú technológiu Google, ale odvtedy sa zovšeobecnil. Od roku 2014 však už Google nepoužíva MapReduce ako svoj primárny model spracovania veľkých dát. [25]



Obrázok 2: Systém Mapreduce, vlastné spracovanie

Framework (alebo systém) MapReduce sa zvyčajne skladá z troch operácií (alebo krokov):

- Mapovanie (map): každý pracovný uzol použije funkciu mapovania na lokálne údaje a výstup zapíše do dočasného úložiska. Hlavný uzol zaisťuje, že sa spracuje iba jedna kópia redundantných vstupných údajov.
- Zmiešanie (shuffle): pracovné uzly redistribuujú údaje na základe výstupných kľúčov (produkovaných mapovacou funkciou), takže všetky údaje patriace k jednému kľúču sa nachádzajú v rovnakom pracovnom uzle.
- Redukcia (reduce): pracovné uzly paralelne spracujú každú skupinu výstupných údajov pre jednotlivé kľúče.

1.7.4 MapReduce-MPI

Knižnica MapReduce-MPI (MR-MPI) je softvér s otvoreným zdrojovým kódom, ktorý implementuje operáciu MapReduce popularizovanú spoločnosťou Google s využitím štandardného odovzdávania správ MPI.

Knižnica je navrhnutá na paralelné vykonávanie na platformách distribuovanej pamäte, ale bude fungovať aj na jednom procesore. Na zostavenie a spustenie nie je potrebný žiadny ďalší softvér, s výnimkou prepojenia s knižnicou MPI, pre paralelné vykonávanie.

Knižnica MR-MPI je napísaná v jazyku C++ a je možné ju používať z vysoko-úrovňových jazykov ako C++, C, Fortran. Zahrnutý je aj Python wrapper, takže programy MapReduce je možné písať v Pythone, vrátane callback metód map() a reduce(). Cieľom knižnice MR-MPI je poskytnúť používateľom jednoduché a prenosné rozhranie na vytváranie vlastných programov MapReduce, ktoré je potom možné spustiť na ľubovoľnom počítači alebo veľkom paralelnom klastri pomocou MPI. Distribúcia obsahuje niekoľko príkladov jednoduchých programov, ktoré ilustrujú použitie MR-MPI.

Knižnica používa MPI na medzi-procesorovú komunikáciu. To umožňuje presnú kontrolu nad pamäťou pridelenou počas rozsiahlej MapReduce úlohy.

Každý vytvorený objekt MapReduce alokuje stránky pamäte pre procesor, kde veľkosť stránky určuje užívateľ. Typické operácie MapReduce je možné vykonať iba na niekoľkých takýchto stránkach. Ak sa množina spracovávaných údajov zmestí na jednu stránku, potom knižnica vykonáva svoje operácie priamo v pamäti. Ak však množina údajov presahuje veľkosť stránky, procesory podľa potreby zapisujú do dočasných súborov na disku a následne z nich čítajú. To umožňuje spracovanie súborov údajov, ktoré sú väčšie, ako celková agregovaná pamäť všetkých procesorov.

Táto knižnica neposkytuje žiadnu odolnosť proti chybám. Súčasné implementácie MPI neumožňujú ľahkú detekciu nefunkčného procesora. Rovnako ako väčšina programov MPI, operácia MapReduce buď zamrzne alebo zlyhá, v prípade ak procesor z nejakého dôvodu zlyhá. [26]

1.7.5 Hadoop

Softvérová knižnica Apache Hadoop je framework, ktorý umožňuje distribuované spracovanie veľkých súborov údajov v klastroch počítačov pomocou jednoduchých programovacích modelov. Je navrhnutý tak, aby sa mohol rozšíriť z jednotlivých serverov na tisíce počítačov, z ktorých každý ponúka lokálne výpočty a úložisko. Namiesto spoliehania sa na hardvér pri zabezpečovaní vysokej dostupnosti je samotná knižnica navrhnutá na zisťovanie a zvládanie zlyhaní na aplikačnej vrstve, takže poskytuje vysoko dostupnú službu ako nadstavbu klastra počítačov, z ktorých každý môže byť náchylný na zlyhanie.

Projekt obsahuje tieto moduly:

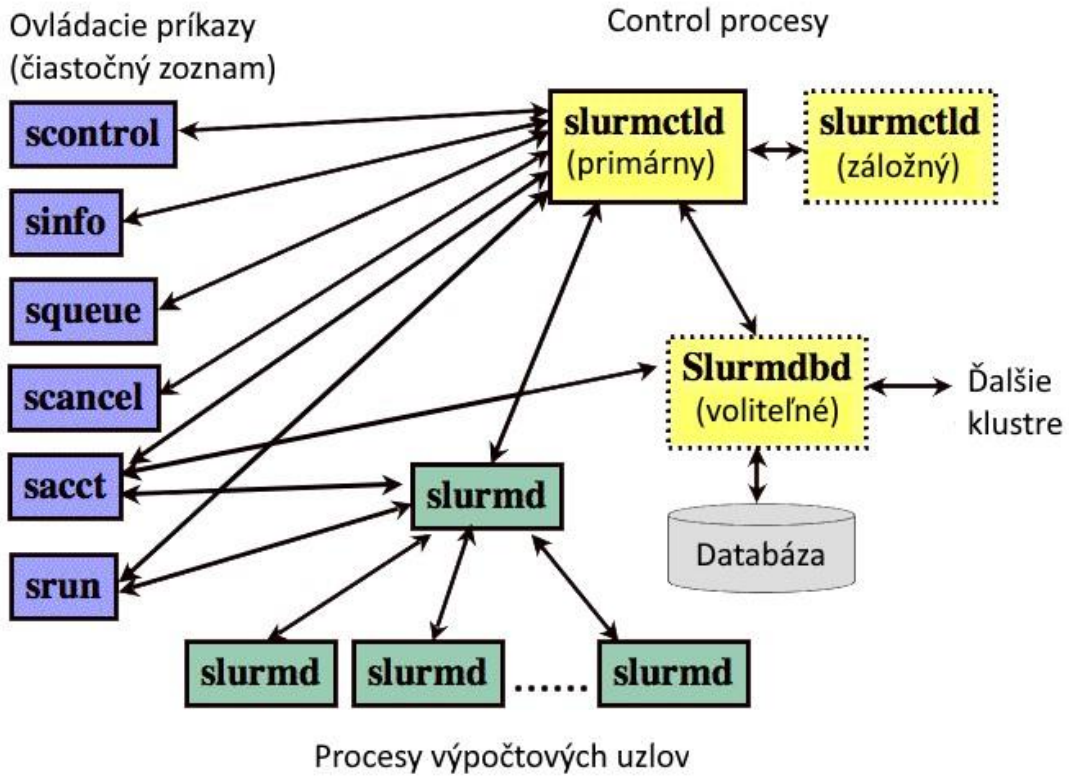
- Hadoop Common: Bežné pomocné programy, ktoré sú podporou pre ďalšie moduly Hadoop.
- Hadoop Distributed File System (HDFS): Distribuovaný súborový systém, ktorý poskytuje vysoko kapacitný prístup k dátam aplikácie.
- Hadoop YARN: Framework pre plánovanie úloh a správu klastrových zdrojov.
- Hadoop MapReduce: Systém založený na YARN pre paralelné spracovanie veľkých súborov údajov.

1.7.6 Slurm

Slurm je open-source, chybám odolný, vysoko škálovateľný systém správy a plánovania úloh pre veľké a malé klastre Linuxu. Slurm nevyžaduje pre svoju činnosť žiadne úpravy jadra a je relatívne samostatný. Ako manažér klastrového zaťaženia má Slurm tri kľúčové funkcie. Najskôr prideľuje používateľom na určitý čas, vyhradený alebo nevyhradený prístup k zdrojom (výpočtovým uzlom), aby mohli vykonávať prácu. Po druhé, poskytuje rámec pre začatie, vykonávanie a monitorovanie práce (zvyčajne paralelnej úlohy), ako je rozhranie Message Passing Interface (MPI), na množine pridelených uzlov. Napokon rozhoduje spory o zdroje riadením frontu čakajúcich prác.

Ako je znázornené na obrázku č.1, Slurm sa skladá z procesu slurmd bežiaceho na každom výpočtovom uzle a centrálneho procesu slurmctld bežiaceho na uzle správy (s voliteľnou zálohou voči zlyhaniu). Procesy slurmd poskytujú hierarchickú komunikáciu odolnú voči poruchám. Medzi užívateľské príkazy patria: sacct, salloc, sattach, sbatch, sbcast, scancel,

scontrol, sinfo, sprio, squeue, srun, sshare, sstat, strigger a sview. Všetky príkazy môžu byť spustené kdekoľvek v klastri.



Obrázok 3: Slurm procesy [27]

Medzi entity spravované týmito procesmi Slurm, patria uzly (výpočtové zdroje v Slurme), oddiely (zoskupujú uzly do logických množín), úlohy (alokácie zdrojov pridelených používateľovi na stanovený čas) a pracovné kroky (súbory krokov v rámci úlohy). Za oddiely je možné považovať fronty úloh, z ktorých každá obsahuje rôzne obmedzenia, ako napríklad limit veľkosti úlohy, limit času úlohy, používatelia, ktorí ju môžu používať, atď. Akonáhle je úlohe priradená skupina uzlov, užívateľ je schopný zahájiť paralelnú prácu vo forme krokov úlohy v akejkoľvek konfigurácii v rámci tejto skupiny. Môže napríklad spustiť jeden krok úlohy, ktorý využíva všetky uzly pridelené k úlohe, alebo niekoľko krokov úlohy môže nezávisle využívať len časť pridelených uzlov.

Linuxové stránky s manuálom *man* existujú pre všetky procesy, príkazy a funkcie aplikačného programového rozhrania Slurm. Voľba príkazu --help tiež poskytuje krátke zhrnutie možností. Uvedieme niektoré Slurm príkazy a ich význam.

- *sacct* sa používa na hlásenie informácií o úlohe alebo aktuálnom kroku úlohy a o aktívnych alebo dokončených úlohách.
- *salloc* sa používa na pridelenie zdrojov pre prácu v reálnom čase. Spravidla sa to používa na alokáciu zdrojov a vytvorenie shellu. Shell sa potom použije na vykonanie príkazu *srun* na spustenie paralelných úloh.
- *sattach* sa používa na pripojenie štandardného vstupu, výstupu a chýb k aktuálne spustenej úlohe alebo kroku úlohy. Je možné sa k úlohám pripájať a odpájať viackrát.
- *sbatch* sa používa na odoslanie skriptu úlohy na neskoršie vykonanie. Skript bude zvyčajne obsahovať jeden alebo viac príkazov srun na spustenie paralelných úloh.
- *sbcast* sa používa na prenos súboru z lokálneho disku na disk v uzloch pridelených úlohe. To možno použiť na efektívne využitie výpočtových uzlov bez diskov alebo na zlepšenie výkonu v porovnaní so zdieľaným súborovým systémom.
- *scancel* sa používa na zrušenie čakajúcej alebo prebiehajúcej úlohy alebo kroku úlohy. Môže sa tiež použiť na odoslanie ľubovoľného signálu všetkým procesom spojeným s bežiacou úlohou alebo krokom úlohy.
- *scontrol* je administratívny nástroj používaný na prezeranie alebo úpravu stavu Slurmu.
- *sinfo* hlási stav oddielov a uzlov spravovaných programom Slurm. Má širokú škálu možností filtrovania, triedenia a formátovania.
- *sprio* sa používa na zobrazenie detailného pohľadu na komponenty ovplyvňujúce prioritu úlohy.
- *squeue* nahlási stav úloh alebo krokov úlohy. Má širokú škálu možností filtrovania, triedenia a formátovania. V predvolenom nastavení hlási spustené úlohy v prioritnom poradí a potom čakajúce úlohy v prioritnom poradí.
- *srun* sa používa na odoslanie úlohy na vykonanie alebo na začatie krokov úlohy v reálnom čase. *srun* má širokú škálu možností na špecifikáciu požiadaviek na

zdroje, vrátane: minimálneho a maximálneho počtu uzlov, počtu procesorov, konkrétnych uzlov, ktoré sa majú alebo nemajú používať, a špecifických charakteristík uzlov (koľko pamäte, miesta na disku, určité požadované funkcie atď.) . Úloha môže obsahovať viac krokov úlohy vykonávaných postupne alebo paralelne na nezávislých alebo zdieľaných zdrojoch v rámci alokácie uzla úlohy.

- *sstat* slúži na získanie informácií o zdrojoch využívaných bežiacou úlohou alebo krokom úlohy.
- *strigger* sa používa na nastavenie, získanie alebo zobrazenie spúšťačov udalostí. Spúšťače udalostí zahŕňajú napríklad pokles uzlov alebo úlohy blížiace sa k časovému limitu.
- *sview* je grafické užívateľské rozhranie na získanie a aktualizáciu informácií o stave úloh, oddielov a uzlov spravovaných softvérom Slurm. [27]

1.8 Paralelný program na otestovanie výkonu

Na demonštráciu výkonu vytvoreného klastra, potrebujeme program ktorý je možné spustiť paralelne a ktorý je možné upraviť tak aby sme paralelným vykonávaním prišli k výsledku rýchlejšie. Jedným z jednoduchších spôsobov paralelizácie je rozdelenie výpočtového cyklu medzi viaceré stroje. Pri tvorení programu použijeme metódu Monte Carlo na výpočet hodnoty pi π.

1.8.1 Odhad hodnoty π pomocou metódy Monte Carlo

Metódy Monte Carlo sú širokou triedou výpočtových algoritmov, ktoré sa pri získavaní výsledkov spoliehajú na opakované generovanie náhodných čísel. Základnou koncepciou je použitie náhodnosti na riešenie problémov. Často sa používajú pri fyzikálnych a matematických problémoch a sú najužitočnejšie, keď je ťažké alebo nemožné použiť iné prístupy. Jedným zo základných príkladov metódy Monte Carlo je odhad hodnoty Ludolfovho čísla π. Metóda je graficky zobrazená na obrázku č. 4.

Základom je štvorec, do ktorého je vpísaná štvrť kružnica. Hodnotu π je možné získať generovaním bodov v priestore štvorca tak, že výsledný pomer všetkých bodov a bodov vo vnútri kruhu nám dá hodnotu π. Matematicky:

Obsah štvrť kružnice: $S_1 = \frac{\pi r^2}{4}$

Obsah štvorca: $S_2 = r^2$

Pomer týchto obsahov: $\frac{S_1}{S_2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$

Potom $\pi = 4\frac{S_1}{S_2}$, ak N je počet bodov tak $\pi = 4\frac{N_{vnútri}}{N_{všetky}}$.

Pseudokód algoritmu:

1. Inicializujeme body_vnútri, body_všetky a i na 0.

2. Vytvoríme náhodné číslo x.

3. Vytvoríme náhodné číslo y.

4. Vypočítame d = x * x + y * y.

5. Ak d <= 1, zvýšime body_vnútri o 1.

6. Zvýšime body_všetky o 1.

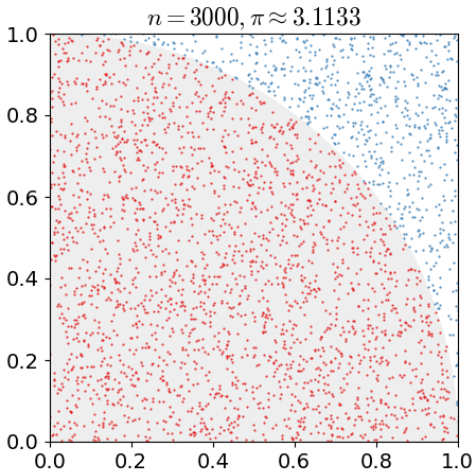
7. Zvýšime i o 1.

8. Ak je i < POČET_ITERÁCII, opakujeme od 2.

9. Vypočítame π = 4 * (body_vnútri / body_všetky).

10. Koniec.

Príslušný kód v jazyku C bude priložený v prílohe.



Obrázok 4: Monte Carlo π [32]

1.9 Virtuálny stroj (Virtual machine)

Virtuálny stroj je emulácia/virtualizácia počítačového systému. Virtuálne stroje poskytujú funkčnosť fyzického počítača na vykonávanie úloh. Jeden alebo viacero virtuálnych strojov typu „hosť“ (guest), môže byť emulovaných na jednom fyzickom stroji typu „hostiteľ“ (host). Každý virtuálny stroj má vlastný operačný systém a funguje nezávisle od ostatných virtuálnych strojov, aj keď všetky bežia na rovnakom hostiteľovi.

Technológia virtuálnych strojov sa používa v mnohých lokálnych prostrediach podnikov alebo v cloudových prostrediach. V poslednej dobe verejné cloudové služby používajú virtuálne stroje na poskytovanie zdrojov pre virtuálne aplikácie a to viacerým používateľom naraz, za účelom efektívnejšieho a flexibilnejšieho využitia výpočtovej techniky.

O správu virtuálnych strojov sa stará tzv. hypervízor (hypervisor) tiež nazývaný monitor virtuálneho stroja VMM (Virtual Machine Monitor), ktorý riadi tok inštrukcií medzi operačnými systémami virtuálnych strojov a fyzickým hardvérom hosťovského počítača ako je CPU, diskové úložisko, pamäť a sieťové karty. Pri každom virtuálnom stroji sú tieto prostriedky nezávislé od ostatných strojov. To znamená že sa javia ako oddelené počítače.

Hypervýzori delíme podľa typu virtualizácie na natívne a hosťované. Pri natívnej virtualizácii beží hypervízor priamo na hardvéri bez hostiteľského operačného systému, prípadne môže byť priamo vstavaný do firmvéru počítača. [28] Pri hosťovanej virtualizácii ale

hypervízor beží nad hostiteľským operačným systémom, a správa sa v podstate ako ostatné programy. Tento typ virtualizácie podporuje všetky najpoužívanejšie operačné systémy ako Windows, macOS, Linux.

Počítač ktorý bude použitý na vytvorenie virtuálneho klastra má operačný systém Windows 10. Pre systém Windows sú k dispozícii voľne dostupné virtualizačné programy VMware Workstation Player a Oracle VirtualBox. Autor má praktické skúsenosti s programom Oracle VirtualBox, preto bude práve tento použitý pri tvorbe virtuálnych strojov pre testovací kluster.

2 Cieľ práce a metodika skúmania

Cieľom práce je vytvoriť virtuálny počítačový klaster a demonštrovať jeho výkon. Práca sa zameria na dva typy klastrov, konkrétne klaster na vysoko výkonné počítanie (HPC) a klaster pre spracovanie veľkých dát. Demonštrovanie výpočtového výkonu HPC klastra bude realizované jednoduchým programom na odhad hodnoty π. Na otestovanie funkcionality klastra na spracovanie veľkých dát bude použitý program založený na metóde MapReduce, ktorý spočíta frekvenciu výskytu slov v datasete obsahujúcom 3200 kníh v anglickom jazyku. Na koniec práca porovná výsledky úloh spustených na klastri a vyvodí záver.

Čiastkovým cieľom je popísanie softvéru a hardvéru potrebného pre vytvorenie virtuálneho klastra. Tiež je opísaný podrobný postup krokov, ktoré je nutné vykonať na inštaláciu, konfiguráciu a spustenie úloh na klastri. Tento postup je možné s rovnakým výsledkom zopakovať aj na fyzických zariadeniach, ktoré sú prepojené sieťou.

Ďalej práca popíše kroky na spustenie úloh, sledovanie stavu úloh a zber výsledkov. A porovná výsledky úloh medzi jednoprocesorovou konfiguráciou a klastrovou konfiguráciou.

Pri realizácií práce sme analyzovali aktuálny trh so softvérovými nástrojmi, ktoré sa používajú pri tvorbe a správe klastrov. Existujú mnohé komerčné riešenia, ktoré ponúkajú firmám kompletné riešenia od zostavenia, cez inštaláciu až po udržiavanie. Veľa z nich stavia na myšlienkach alebo aj priamo na open source systémoch, a to konkrétne Slurm pre vysoko výkonné počítačové klastre a Hadoop pre spracovanie veľkých dát. Práca analyzuje a popíše vytvorenie klastra práve s použitím dvoch spomínaných voľne dostupných systémov. Na demonštráciu výkonu bude použitá jednoduchá implementácia Monte Carlo metódy, a pri spracovaní veľkých dát použijeme vzorový program na výpočet frekvencie slov. Tieto programy budú mať priamo v sebe zabudované meranie času potrebného na výpočet, teda trvania programu. Každú úlohu spustíme desaťkrát aby, sme sa vyhli možnej odchýlke, spôsobenej nerovnomerným zaťažením zariadenia počas vykonávania rôznych úloh. Na záver z nameraných hodnôt určíme geometrický priemer, ktorý je v tomto prípade lepší ukazovateľ, pretože jedna hodnota s neobvykle veľkou odchýlkou neovplyvní výsledok tak výrazne ako pri aritmetickom priemere. [29]

3 Výsledky práce a diskusia

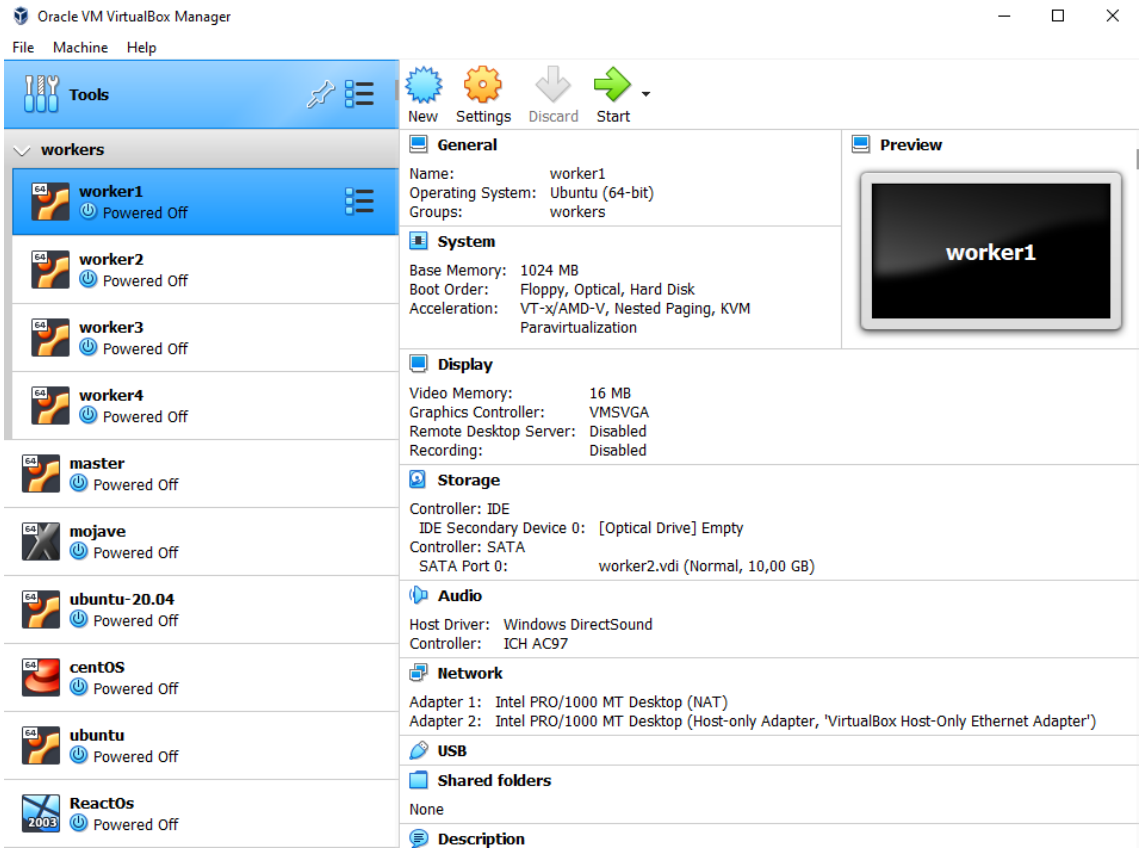
V tejto kapitole popíšeme podrobný postup pri inštalácií softvéru potrebného na vytvorenie virtuálneho klastra. V našom prípade nainštalujeme na klaster plánovací systém Slurm a popri ňom aj systém na spracovanie big data Hadoop, takže náš klaster bude schopný používať funkcionalitu oboch systémov. Ak ale použijeme len jeden postup buď z kapitoly 3.3 alebo 3.4, dopracujeme sa tiež k funkčnému klastru s vybraným systémom. V prípade ak by sme mali k dispozícií fyzický hardvér, vynecháme postup 3.1 a môžeme začať inštaláciu operačného systému priamo na fyzické zariadenia, tak že vytvoríme bootable USB zariadenie s obrazom Linuxu a ďalej pokračujeme podľa postupu 3.2.

Po nainštalovaní potrebného softvéru popíšeme postup na vytvorenie a spustenie úloh, ktoré budú bežať na klastri. Tieto úlohy spustíme, zozbierame výstup a čas trvania a na záver porovnáme trvanie výpočtu pri použití rôznych konfigurácii.

3.1 Inštalácia Oracle VM VirtualBox

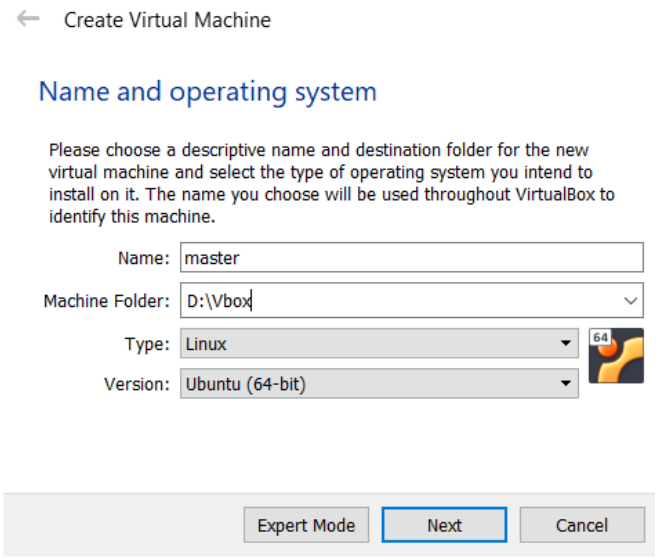
VirtualBox je bezplatná a open-source virtualizačná platforma, ktorá dokáže virtualizovať x86 a AMD64 / Intel64 architektúry . Môže byť nainštalovaná na existujúce operačné systémy Windows, Mac, Linux alebo Solaris. Obsahuje množstvo funkcií, ktoré nájdu vyžitie v malých vstavaných systémoch alebo stolových počítačoch až po dátové centrá a cloudové prostredie. Ponúka širokú hardvérovú podporu: hosťované multiprocesy (SMP), USB, hardvérová kompatibilita (IDE, SCSI, SATA, niekoľko virtuálnych sieťových kariet, zvukové karty, virtuálne sériové a paralelné porty, I / O APIC), plná podpora ACPI, rozlíšenie viacerých obrazoviek, vstavaná podpora iSCSI, sieťová podpora PXE. [8]

Program je voľne dostupný na stránke https://www.virtualbox.org. Stiahli sme najnovšiu verziu VirtualBox 6.1.18 pre hostiteľský systém Windows. Potom sme spustili inštalačný súbor, všetky nastavenia nechali na predvolených hodnotách a začali inštaláciu. Po nainštalovaní sa otvorí okno Oracle VM VirtualBox Manager ako na obrázku č. 5. Pre spustenie samotného virtuálneho stroja musí náš hosťovský počítač podporovať virtualizáciu. Niektoré modely majú túto funkciu od výroby vypnutú. V tomto prípade je potrebné virtualizáciu povoliť v BIOSe zariadenia.



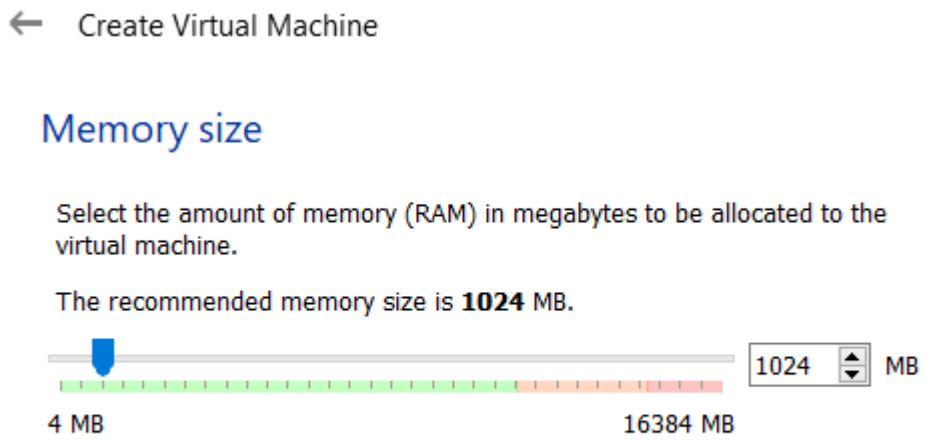
Obrázok 5: Prostredie Oracle VM VirtualBox

Pomocou tlačidla New sme vytvorili nový virtuálny stroj typu linux Ubuntu (64-bit).



Obrázok 6: Vytvorenie virtuálneho stroja

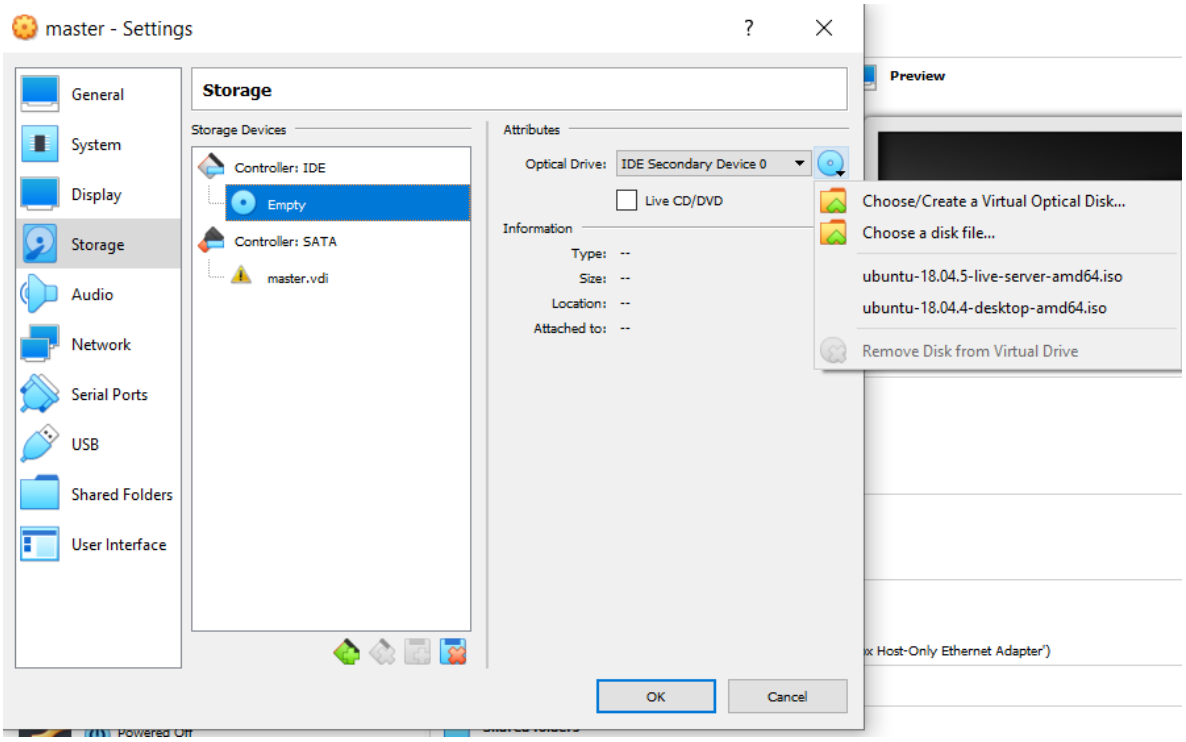
V ďalšom kroku sme priradili virtuálnemu stroju veľkosť pamäte RAM, ktorú mu náš hosťovský počítač (OS) ponúkne k dispozícií. Samozrejme nie je možné použiť 100% pamäte dostupnej na hosťovskom počítači, pretože samotný hosťovský OS potrebuje na vlastné fungovanie a na spustenie programu VirtualBox určitú časť pamäte. Táto časť je približne znázornená oranžovou a červenou farbou na posúvači. Pre náš prípad použitia sme vybrali predvolenú hodnotu 1024MB, ktorá stačí na pokrytie potrieb operačného systému Ubuntu server.



Obrázok 7: Priradenie prostriedkov virtuálnemu stroju

Potom sme vytvorili virtuálny disk a použili predvolené hodnoty, to znamená typ disku VDI (VirtualBox Disk Image), ktorý bude dynamicky alokovaný (na začiatku nebude zaberať celú alokovanú veľkosť ale len časť ktorá je aktuálne využitá). Ďalej sme zvolili požadovanú cestu kam sa tento virtuálny disk vytvorí a predvolenú veľkosť 10GB, ktorá je v našom prípade dostačujúca. V liste virtuálnych strojov nám pribudne náš novo vytvorený stroj.

Posledným krokom je vloženie inštalačného obrazu operačného systému, ktorý chceme na virtuálny stroj nainštalovať. Použili sme distribúciu OS Ubuntu, konkrétne Ubuntu server vo verzii 20.04.2 LTS, voľne dostupný na stránke https://ubuntu.com/download/server. Stiahnutý obraz, sme pridali v nastaveniach pod záložkou Storage do voľného IDE controlleru. Zvolili sme obraz disku ubuntu-18.04.5-live-server-amd64.iso, ako na obrázku č. 8 a následne môžeme spustiť virtuálny stroj.



Obrázok 8: Vloženie inštalačného obrazu

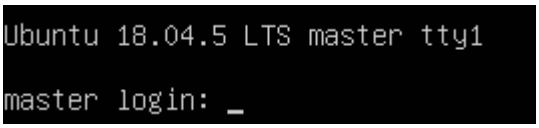
3.2 Inštalácia OS Ubuntu

Po spustení virtuálneho stroja sa dostaneme na obrazovku s výberom jazyka. Keďže sa jedná o distribúciu pre server, neobsahuje GUI grafické používateľské prostredie. Preto na navigáciu v menu budeme používať šípky a enter na klávesnici.

1. Zvolíme predvolený jazyk English stačením klávesy Enter.
2. Predvolené hodnoty zvolíme aj pre nastavenie rozloženia klávesnice.
3. Pri sieťových adaptéroch zvolíme predvolený adaptér enp0s3.
4. Proxy nevypĺňame a posunieme sa Enterom na ďalšiu obrazovku.
5. Ubuntu mirror tiež necháme na predvolenej hodnote a prejdeme na dalšiu obrazovku.
6. Pri nastavení využitia disku zvolíme pole Use an entire disk a Use LVM group.
7. Potvrdíme enterom a znova potvrdíme naformátovanie disku zvolením hodnoty Continue.
8. Pomenujeme server ako „master“, a tiež vyplníme „master“ do všetkých ďalších políčok.

9. Zvolíme enterom možnosť Install OpenSSH server.
10. Na ďalšej obrazovke nevyberieme nič, len sa presunieme ďalej klávesou enter.
11. Začne sa inštalácia operačného systému, ktorá trvá niekoľko minút.
12. Po skončení zvolíme možnosť reboot, na reštartovanie systému do novo nainštalovaného operačného systému.

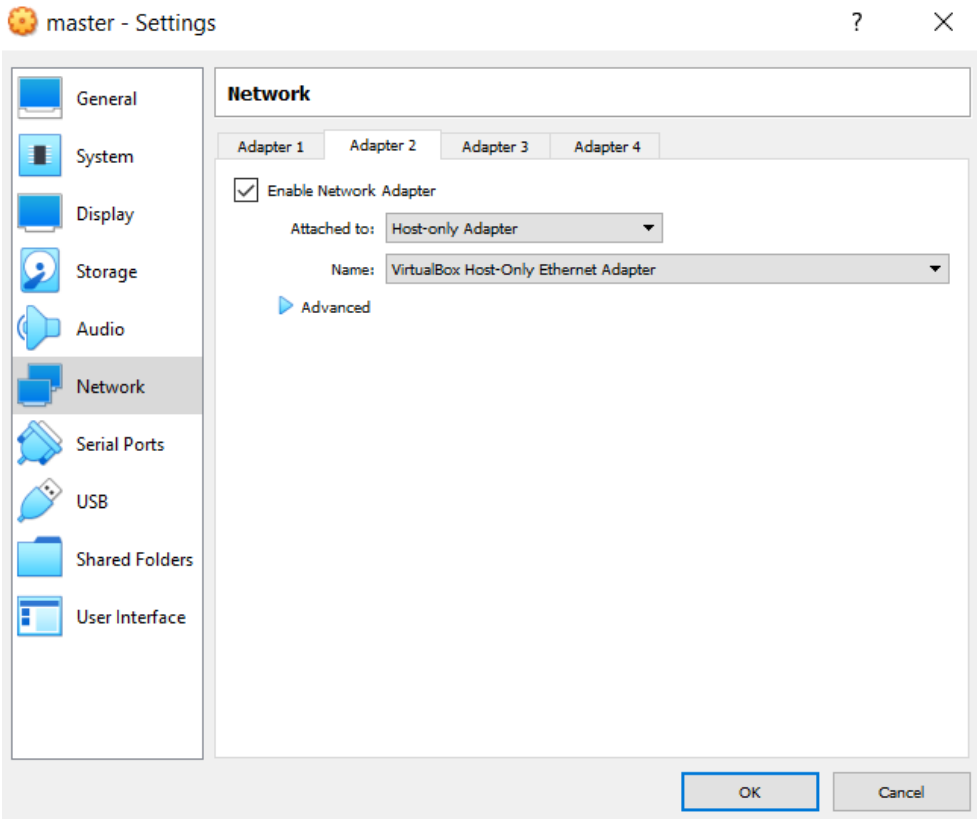
Po štarte operačného systému sa dostaneme do terminálu OS kde zadáme login a password „master“.



Obrázok 9: Konzola virtuálneho stroja

Tento postup môžeme zopakovať na vytvorenie virtuálnych strojov pre worker uzly. Je ale efektívnejšie a jednoduchšie duplikovať už vytvorený virtuálny stroj priamo v nástroji VirtualBox, ktorý túto možnosť ponúka. Tiež je ale výhodné najprv nainštalovať všetky knižnice a programy, potrebné na beh výpočtového klastra, ktoré sa zduplikujú v nových virtuálnych strojoch. Tým uľahčíme a skrátime konfiguráciu nových virtuálnych strojov a tiež sa vyhneme možným chybám pri opakovaných inštaláciách.

Na zabezpečenie komunikácie vnútri klastra je ešte potrebné pridať do virtuálneho stroja druhú sieťovú kartu typu Host-only adapter. Túto sme pridali v menu Network na obrazovke nastavenia, tak ako je zobrazené na obrázku č. 10.

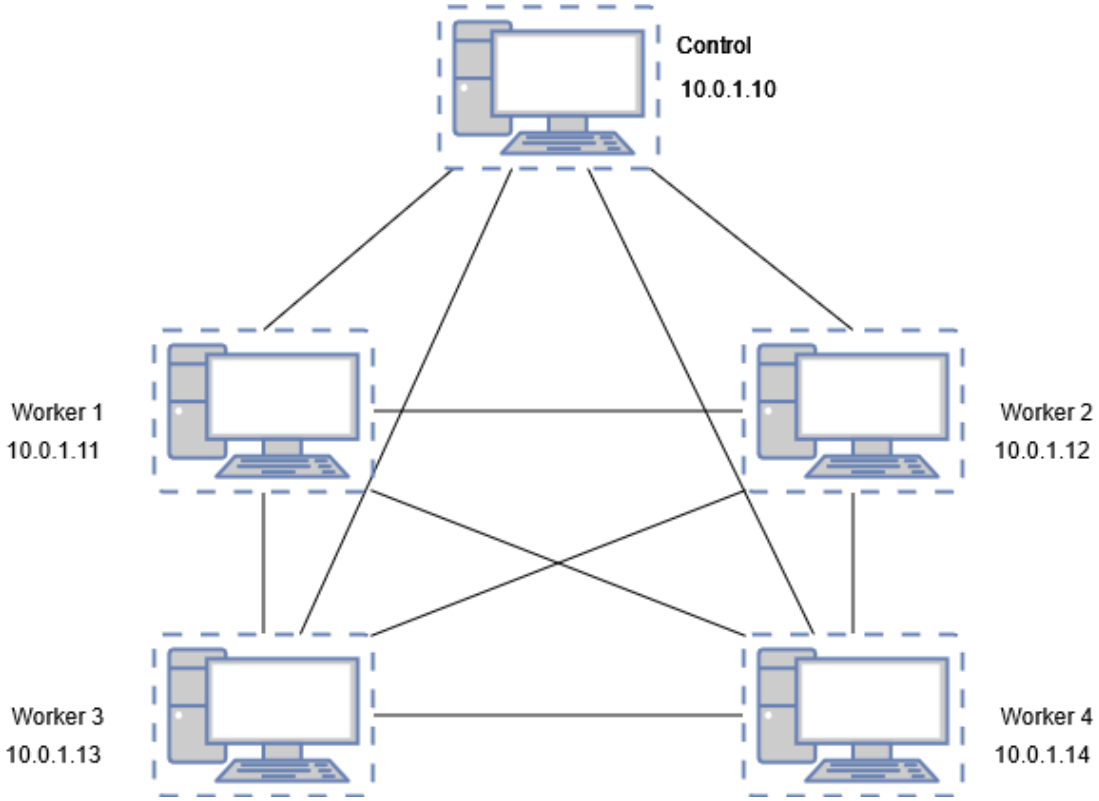


Obrázok 10: Pridanie sieťového adaptéra

3.3 Inštalácia SlurmWorkload manager

Na demonštrovanie výkonu paralelnej aplikácie sme vytvorili testovací klaster z 5 PC. V praxi by sme mohli použiť 5 počítačov prepojených sieťou. Vzhľadom na aktuálnu pandemickú situáciu sme nepoužili reálne počítače ale využili sme virtualizáciu. Virtualizácia nám umožňuje vytvoriť na jednom počítači (host) celú našu klaster architektúru, ktorá je zobrazená na obrázku č.11.

Klaster sa skladá z 5 virtuálnych strojov s 1 vCPU, 1GB RAM a 10GB SSD. Tieto stroje sú vzájomne prepojené sieťou a každému je pridelená statická IP adresa.



Obrázok 11: Schéma virtuálneho klastra

1. Vytvorenie používateľov pre slurm a munge

Po inštalácii OS potrebujeme vytvoriť používateľov a skupiny munge and slurm na všetkých počítačoch. GID a UID (skupinové a užívateľské ID) sa musia zhodovať pre munge a slurm na všetkých strojoch.

```
sudo adduser -u 1111 munge --disabled-password --gecos ""
sudo adduser -u 1121 slurm --disabled-password --gecos ""
```

MUNGE (MUNGE Uid 'N' Gid Emporium) je autentifikačná služba na vytváranie a overovanie prihlasovacích údajov. Je navrhnutá tak, aby bola vysoko škálovateľná na použitie v prostredí klastra HPC. Umožňuje procesu autentifikovať UID a GID iného lokálneho alebo vzdialeného procesu v rámci skupiny hostiteľov, ktorí majú spoločných používateľov a skupiny. Títo hostitelia tvoria oblasť bezpečnosti, ktorá je definovaná zdieľaným kryptografickým kľúčom. Klienti v tejto oblasti zabezpečenia môžu vytvárať a overovať poverenia bez použitia oprávnení root, vyhradených portov alebo metód špecifických pre platformu.

2. Inštalácia NFS (zdieľaného úložiska)

Aby SLURM fungoval správne, musí byť na všetkých počítačoch v klastri miesto s úložiskom s rovnakými súbormi, ktoré sú potrebné pre vykonanie úlohy. Všetky počítače v klastri musia byť schopné čítať a zapisovať do tohto adresára. Jedným zo spôsobov, ako to zabezpečiť, je NFS, aj keď existujú ďalšie možnosti, ako napríklad OCFS2. My sme využili práve NFS.

Podľa zvolenej architektúry máme jeden hlavný control (hostiteľský ukladací priestor a radič SLURM) a ďalšie pracovné uzly nazvané worker. Meno používateľa / názov skupiny pre hlavný účet správcu na všetkých počítačoch je master: master. Pre administratívne účty na všetkých serveroch sme použili rovnaké používateľské meno a skupinu.

Do nášho virtuálneho stroja sme nainštalovali NFS server pomocou nasledujúceho príkazu.

```
sudo apt install nfs-kernel-server -y
```

Potom sme vytvorili priečinok storage a pridelili práva používateľovi a skupine master.

```
sudo mkdir /storage

sudo chown admin:admin /storage
```

Ďalej sme pridali do súboru /etc/exports riadok:

```
/storage *(rw,sync,no_root_squash)
```

Znak * je pre adresy IP alebo názvy hostí. V tomto prípade povoľujeme čokoľvek, ale je možné obmedziť prístup na vlastné adresy IP / názvy hostí v klastri.

Potom sme spustili službu NFS:

```
sudo systemctl start nfs-kernel-server.service
```

Služba sa spustí automaticky po reštartovaní zariadenia.

3. Nastavenie nfs klientov.

Ďalej nastavíme klientov NFS na všetkých pracovných serveroch:

```
sudo apt install nfs-common -y
sudo mkdir /storage
sudo chown master:master /storage
sudo mount master:/storage /storage
```

Aby sa jednotka po reštartovaní pracovných uzlov pripojila, pridali sme nasledujúci riadok do súboru fstab (/etc/fstab):

```
master:/storage /storage nfs auto,timeo=14,intr 0 0
```

4. Inštalácia munge na control uzle.

```
sudo apt-get install libmunge-dev libmunge2 munge -y
sudo systemctl enable munge
sudo systemctl start munge
```

5. Skopírovanie munge kľúč do /storage.

```
sudo cp /etc/munge/munge.key /storage/
sudo chown munge /storage/munge.key
sudo chmod 400 /storage/munge.key
```

6. Inštalácia munge na worker uzloch.

```
sudo apt-get install libmunge-dev libmunge2 munge
sudo cp /storage/munge.key /etc/munge/munge.key
sudo systemctl enable munge
sudo systemctl start munge
```

7. Inštalácia prerekvizít pre slurm

```
sudo apt-get install git gcc make ruby ruby-dev libpam0g-dev
libmariadb-client-lgpl-dev libmysqlclient-dev mariadb-server build-
essential libssl-dev -y
sudo gem install fpm
```

8. Vytvorenie inštalačného súboru slurm zo zdrojového kódu

Stiahli sme zdrojový kód SLURM voľne dostupný na stránke https://www.schedmd.com. Ten sme potom rozbalili a nainštalovali pomocou príkazu make.

```
cd /storage
wget https://download.schedmd.com/slurm/slurm-20.11.5.tar.bz2
tar xvjf slurm-20.11.5.tar.bz2
cd slurm-20.11.5
./configure --prefix=/tmp/slurm-build --sysconfdir=/etc/slurm --
enable-pam --with-pam_dir=/lib/x86_64-linux-gnu/security/ --without-
shared-libslurm
make
make contrib
make install
cd ..
```

9. Zabalenie a inštalácia slurm

```
sudo fpm -s dir -t deb -v 1.0 -n slurm-19.05.2 --prefix=/usr -C
/tmp/slurm-build .
sudo dpkg -i slurm-19.05.2_1.0_amd64.deb
```

10. Vytvorenie potrebných priečinkov.

```
sudo mkdir -p /etc/slurm /etc/slurm/prolog.d /etc/slurm/epilog.d
/var/spool/slurm/ctld /var/spool/slurm/d /var/log/slurm
sudo chown slurm /var/spool/slurm/ctld /var/spool/slurm/d
/var/log/slurm
```

11. Skopírovanie konfiguračného súboru conf.

```
sudo cp /storage/slurmdbd.conf /etc/slurm/
```

12. Spustenie slurm služby:

```
sudo systemctl daemon-reload
sudo systemctl enable slurmdbd
sudo systemctl start slurmdbd
sudo systemctl enable slurmctld
sudo systemctl start slurmctld
```

13. Inštalácia Slurm na worker uzloch

```
cd /storage
sudo dpkg -i slurm-19.05.2_1.0_amd64.deb
sudo cp /storage/ubuntu-slurm/slurmd.service /etc/systemd/system/
sudo systemctl enable slurmd
sudo systemctl start slurmd
```

14. Nakopírovanie konfiguračných súborov do všetkých worker uzlov.

```
sudo cp /storage/ubuntu-slurm/cgroup* /etc/slurm/
sudo cp /storage/slurm.conf /etc/slurm/
sudo cp /storage/gres.conf /etc/slurm/
```

15. Vytvorenie potrebných priečinkov:

```
sudo mkdir -p /var/spool/slurm/d
sudo chown slurm /var/spool/slurm/d
```

Po tomto nakonfigurovaní sme reštartovali všetky stroje.

3.3.1 OpenMPI

Inštalácia OpenMPI pre podporu spúšťania programov s posielaním správ.

```
sudo apt install openmpi-bin libopenmpi-dev
```

3.3.2 Mapreduce-MPI

Stiahli sme zdrojový kód knižnice mapreduce-MPI zo stránky https://cs.sandia.gov/~sjplimp/download.html. Rozbalili sme stiahnutý súbor a skompilovali zdrojový kód, tak aby bol spustiteľný na našom stroji. Tiež sme skompilovali ukážkové programy v priečinku examples, pričom program *wordfreq.cpp* na spočítanie výskytu jednotlivých slov v texte sme použili na ukážku výkonu klastra.

```
gunzip mapreduce.tar.gz
tar xvf mapreduce.tar
cd mapreduce-7Apr14/src
make mpicc
cd ../examples
make -f Makefile.mpicc
```

Slurm ponúka množstvo ďalších funkcií, ktoré sme pre zachovanie jednoduchosti nevyužili. Napríklad nastavenie kvót na množstvo použiteľného úložiska pre používateľov,

freeIPA alebo LDAP pre správu prístupu pre používateľov. Slurm je tiež možné nakonfigurovať tak, aby zhromažďoval účtovné informácie o každej vykonanej úlohe a krokoch tejto úlohy. Tieto podrobné informácie je možné ukladať do sql databázy. Tieto a ďalšie administratívne prvky je možné zobraziť pomocou grafického používateľského rozhrania, ktoré sa spúšťa príkazom sview.

3.4 Inštalácia Hadoop

Framework Hadoop je implementovaný v jazyku Java, preto je potrebné mať na všetkých virtuálnych strojoch nainštalovanú vývojovú sadu Java (Java Development Kit) JDK. Použili sme voľne dostupnú verziu OpenJDK vo verzii 8, ktorej funkcie Hadoop plne podporuje.

```
sudo apt install openjdk-8-jdk
```

Hadoop tiež pre svoje fungovanie potrebuje bezheslový ssh prístup z master uzla do všetkých worker uzlov. Preto sme vygenerovali a rozdistribuovali ssh master kľúč do všetkých worker uzlov.

Na master uzle (nfs):

```
ssh-keygen -b 4096
cp ~/.ssh/id_rsa.pub /storage/master.pub
```

Na ostatných worker uzloch:

```
cat /storage/master.pub >> ~/.ssh/authorized_keys
```

Ďalej sme stiahli a rozbalili binárne súbory samotného Hadoop frameworku. Potom sme nastavili cestu k týmto rozbaleným súborom, aby sme ich mohli volať priamo príkazmi v konzole.

```
cd

wget http://apache.cs.utah.edu/hadoop/common/current/hadoop-
3.3.0.tar.gz

tar -xzf hadoop-3.3.0.tar.gz

mv hadoop-3.3.0 hadoop

export HADOOP_HOME=/home/master/hadoop
export PATH=${PATH}:${HADOOP_HOME}/bin:${HADOOP_HOME}/sbin
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

Posledným krokom je nakonfigurovanie frameworku Hadoop pre našu implementáciu klastra. Tento proces zahŕňa nastavenie sieťovej adresy master uzla, nastavenie lokálnych súborov, ktoré budú použité pre distribuovaný súborový systém HDFS, nastavenie systému spravovanie úloh YARN a zadefinovanie dostupných worker uzlov. Všetky potrebné hodnoty sa nastavujú pomocou xml konfiguračných súborov, ktoré sa nachádzajú v priečinku ~/hadoop/etc/hadoop/. V tejto práci sa nebudeme venovať jednotlivým súborom a ich významu, ale ak by čitateľ narazil na problém, v oficiálnej dokumentácii pre Hadoop[1] sú tieto súbory popísané. Po nakonfigurovaní všetkých potrebných nastavení, sme spustili na master uzle HDFS systém, a vytvorili domovský priečinok.

```
hdfs namenode -format
hdfs dfs -mkdir -p /home/master
```

Na spustenie a zastavenie všetkých súčastí Hadoop frameworku sme použili nasledujúce príkazy:

```
start-all.sh

stop-all.sh
```

A pomocou príkazov pre YARN sme zobrazili list dostupných uzlov a list bežiacich aplikácii.

```
yarn node -list
yarn application -list
```

[1] https://hadoop.apache.org/docs/r3.2.2/hadoop-project-dist/hadoop-common/ClusterSetup.html

3.5 Otestovanie funkčnosti a výkonnosti klastra.

Na otestovanie funkčnosti, to znamená či sú všetky uzly klastra dostupné a schopné medzi sebou komunikovať, sme použili vstavané monitorovacie funkcie, ktoré ponúka slurm aj Hadoop. Prvým krokom je spustenie všetkých virtuálnych strojov a to konkrétne control a worker1-4.

Funkčnosť systému slurm vieme otestovať pomocou príkazu *sinfo* na ktoromkoľvek uzle klastra, ktorý zobrazí informácie o aktuálnom stave klastra. V prípade že sú všetky uzly spustené, dokážu medzi sebou komunikovať a na každom worker uzle je spustený slurmd proces, je výstup príkazu *sinfo* nasledovný.

```
master@master:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE  NODELIST
debug*       up   infinite      4   idle  worker[1-4]
```

Systému Hadoop ponúka viacero monitorovacích možností, a to buď priamo príkazmi alebo cez webové používateľské rozhranie. Pomocou príkazov môžeme zistiť stav oboch kľúčových častí systému Hadoop a to HDFS a YARN. Na control uzle sme po spustení celého systému Hadoop, použili príkazy *hdfs dfsadmin* a *yarn node* s nasledujúcim výstupom.

```
master@master:~$ start-all.sh
master@master:~$ hdfs dfsadmin -report
...
Live datanodes (4):
Name: 13.0.0.11:9866 (worker1)
...
master@master:~$ yarn node -list
Total Nodes:4
Node-Id        Node-State Node-Http-Address       Number-of-Containers
worker1:43571     RUNNING      worker1:8042                          0
worker2:38267     RUNNING      worker2:8042                          0
worker3:34705     RUNNING      worker3:8042                          0
worker4:34975     RUNNING      worker4:8042                          0
```

Tieto výpisy nám poskytujú prehľad o aktuálnom stave klustra a jeho uzloch. Ak sú všetky uzly nakonfigurované správne, mali by sa nachádzať vo výpise zo stavom RUNNING. V tomto stave je kluster pripravený na vykonávanie úloh a môžeme pokračovať priradením prvej úlohy na spracovanie.

3.5.1 Metóda Monte Carlo pi

Kód z kapitoly **Odhad hodnoty π** sme skompilovali pomocou mpicc kompilátora, ktorý je súčasťou OpenMPI. Na skompilovanie sme použili nasledovný príkaz.

```
mpicc -o pi.mpi pi.c
```

Skompilovaný program sme skopírovali do zdieľaného priečinku /storage. Potom sme vytvorili konfiguračný slurm skript pre našu úlohu. A úlohu sme príkazom *sbatch* zaradili do rady úloh, ktorej vykonanie slurm pri dostatočnom počte pracovných uzlov zabezpečí. Pomocou prepínača -N sme nastavili počet uzlov na ktorých má úloha bežať. Do slurm skriptu, ktorý sme nazvali run.sh, sme pridali príkaz na spustenie nami skompilovaného mpi programu na odhad hodnoty π, ktorý sme nazvali pi.mpi, ďalej sme definovali cestu kam sa uloží súbor s výstupom a zadali sme počet opakovaní úlohy. Úlohu zopakujeme desaťkrát aby sme pri porovnaní výkonnosti zobrali do úvahy aj náhodné chyby merania.

```
#!/bin/bash
#SBATCH --job-name=odhad-pi
#SBATCH --output=out/out-%j.txt
#SBATCH --array=0-9

mpirun pi.mpi
```

Súbor s C kódom metódy Monte Carlo je priložený v prílohe. Tento slurm skript sme teda spustili na našom virtuálnom klastri a porovnali čas vykonávania na jednom výpočtovom uzle a pri použití všetkých výpočtových uzlov. Príkaz `squeue` sme použili na zobrazenie úloh v rade s nasledujúcim výstupom.

```
master@master:~$ sbatch -N 1 run.sh
master@master:~$ squeue
JOBID PARTITION     NAME     USER ST      TIME  NODES NODELIST(REASON)
165_[4-9] debug odhad-pi   master PD      0:00      1 (Resources)
165_0     debug odhad-pi   master  R      0:04      1 worker1
165_1     debug odhad-pi   master  R      0:04      1 worker2
165_2     debug odhad-pi   master  R      0:04      1 worker3
165_3     debug odhad-pi   master  R      0:04      1 worker4

master@master:~$ sbatch -N 4 run.sh
master@master:~$ squeue
JOBID PARTITION     NAME     USER ST      TIME  NODES NODELIST(REASON)
175_[1-9] debug odhad-pi   master PD      0:00      4 (Resources)
175_0     debug odhad-pi   master  R      0:03      4 worker[1-4]
```

Po dokončení úloh sa v priečinku out/ nachádzajú súbory s výstupom, ktoré majú nasledovný obsah.

```
Počet uzlov= 1;    Čas=8.163481s;    PI=3.1415;
...
Počet uzlov= 4;    Čas=2.126565s;    PI=3.1412;
...
```

Prvých desať súborov je výstup úlohy na jednom uzle a ďalších desať je výstup úlohy na všetkých (4) uzloch. Dosiahnuté časy sme spriemerovali pomocou geometrického priemeru a porovnali v nasledujúcej tabuľke č.1.

Počet uzlov	Čas (s)		Počet uzlov	Čas (s)
1	8.163481		4	2.126565
1	8.388721		4	2.088794
1	9.092166		4	2.070926
1	8.321323		4	2.097465
1	8.301013		4	2.103377
1	8.297427		4	2.133001
1	8.336725		4	2.097465
1	8.250783		4	2.116232
1	8.333463		4	2.118499
1	8.180822		4	2.143753
Priemer	8.362995		Priemer	2.109504

Tabuľka 1: Čas výpočtu programu na odhad hodnoty Pi

Pomer medzi časom výpočtu na jednom uzle a na štyroch uzloch je 3.9644, čo zodpovedá štvornásobnému nárastu zdrojov ktoré má výpočet k dispozícii. Tento pomer sa môže takto blízko priblížiť vďaka samotnému algoritmu na odhad hodnoty π, ktorý rozdelí počet iterácií výpočtu rovnomerne medzi všetky uzly bez potreby medzi-výpočtovej komunikácie. To znamená že sa uzly nezaťažia réžiou synchronizácie medzi-výpočtov a dokážu tak využiť skoro celý svoj výkon na vykonávanie samotnej úlohy.

Z tohto porovnania môžeme vyvodiť záver, že paralelizácia má zmysel a v našom prípade prináša nárast výkonu, ktorý rastie lineárne s počtom uzlov.

3.5.2 Mapreduce

Pre otestovanie a ilustráciu funkčnosti metódy mapreduce na spracovanie big data je potrebný dataset s veľkým množstvom dát. Obe implementácie MapReduce-MPI a Hadoop obsahujú hotový program na výpočet výskytu slov v texte. Použili sme dataset dostupný na stránke kaggle.com[2]. Tento dataset obsahuje text z náučných kníh voľne vydávaných pod projektom Wikipédie ako Wikibooks. Knihy sú rozdelené podľa jazyka, pričom knihy v anglickom jazyku majú najväčšie zastúpenie a po extrahovaní textového obsahu kníh z csv súboru, dosahuje tento súbor veľkosť 693MB a je vhodný na demonštráciu metódy MapReduce.

[2] https://www.kaggle.com/dhruvildave/wikibooks-dataset

Tento súbor sme rozdelili na 4 rovnaké časti, a uložili do zdieľaného priečinku */storage/data/en/*.

3.5.3 Mapreduce MPI

V zdieľanom priečinku */storage/mapreduce-7Apr14/examples*, ktorý obsahuje skompilovaný kód knižnice mapreduce-mpi sme vytvorili slurm skript *run.sh* na spustenie úlohy.

```
#!/bin/bash
#SBATCH --job-name=mapreduce
#SBATCH --output=output/out-%j.txt
#SBATCH --array=0-9


mpirun --allow-run-as-root --mca oob_tcp_if_include enp0s8 --mca
btl_tcp_if_include enp0s8 -x NCCL_SOCKET_IFNAME=enp0s8 wordfreq
/storage/data/en/*
```

Úlohu sme potom spustili príkazom *sbatch* a skontrolovali jej vykonávanie pomocou príkazu *squeue*.

```
sbatch -N 1 run.sh
sbatch -N 4 run.sh
squeue
JOBID PARTITION     NAME     USER ST    TIME  NODES NODELIST(REASON)
239_[4-9] debug mapreduc   master PD    0:00      1 (Resources)
239_3     debug mapreduc   master  R    1:04      1 worker4
239_2     debug mapreduc   master  R    1:19      1 worker3
239_1     debug mapreduc   master  R    1:31      1 worker2
239_0     debug mapreduc   master  R    1:45      1 worker1
```

V priečinku *output/* sa nachádzajú súbory s výstupom, ktoré obsahujú 10 najpoužívanejších slov a čas trvania ich spočítania, pričom obsah týchto súborov je nasledovný.

```
...
4298819 the
2508404 of
2043447 to
1960357 and
1667624 a
1312993 in
1298812 is
735370 that
663243 for
623065 The
...
96759077 total words, 3962115 unique words
Time to process 4 files on 4 procs = 64.665 (secs)
...
Time to process 4 files on 1 procs = 259.332 (secs)
```

3.5.4 Hadoop

Spustenie úlohy na Hadoop klustri, sme vykonali týmito krokmi. Najprv sme presunuli súbory s knihami do HDFS súborového systému a potom sme spustili program na spočítanie výskytu slov. Program bol spustený na všetkých dostupných uzloch. Následne sme z výstupu vypísali 10 najčastejších slov, pričom zoradenie najpoužívanejších slov bolo rovnaké ako pri použití MapReduce-MPI.

```
hdfs dfs -mkdir books
hdfs dfs -put /storage/data/en/* books
yarn jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-
3.2.2.jar wordcount "books/*" output
hdfs dfs -cat output/part-r-00000 | sort -n -nk2 -r | head -n10
```

3.5.5 Porovnanie výkonnosti MapReduce

Po zozbieraní dát o trvaní výpočtu pri metóde spracovanie veľkých dát MapReduce, sme porovnali jednotlivé implementácie. Údaje o čase vykonávania úloh sú spracované v nasledujúcej tabuľke č.2.

MapReduce-MPI			MapReduce-MPI			Hadoop	
Počet uzlov	Čas (s)		Počet uzlov	Čas (s)		Počet uzlov	Čas (s)
1	259.332		4	64.665		4	136
1	233.302		4	67.1634		4	159
1	195.445		4	56.6064		4	128
1	195.401		4	62.8063		4	160
1	194.93		4	71.1387		4	156
1	213.257		4	71.9452		4	126
1	130.672		4	69.9462		4	162
1	-		4	72.4447		4	127
1	-		4	69.5584		4	147
1	-		4	66.6046		4	128
Priemer	199.4403		Priemer	67.1182		Priemer	142.1545

Tabuľka 2: Čas výpočtu programu MapReduce

Pri implementácií MapReduce-MPI, ktorá bežala na jednom uzle sa vyskytli chyby, kedy v troch prípadoch program padol a nedokázal sa korektne ukončiť, čo mohlo byť spôsobené nedostatočnou RAM pamäťou. Aj Hadoop mal problémy pri vykonávaní mapovacej funkcie a viacero pod-úloh zlyhalo, s čím sa ale Hadoop dokáže vysporiadať a dopracovať sa ku konečnému výsledku. V tomto prípade je tiež problém s nedostatočným hardvérom, pretože komerčné Hadoop riešenia ako napr. Cloudera, Hortonworks, MapR pracujú s RAM pamäťou s minimálnou veľkosťou 8GB pre worker uzol. [30]

V prípade knižnice MapReduce-MPI môžeme porovnať priemerný čas úlohy na jednom uzle a na štyroch uzloch. V prípade štvornásobného počtu uzlov je čas výpočtu 2.97 krát menší. Zväčšenie výkonu je teda zrejmé, aj keď nie tak efektívne ako v prípade jednoduchého odhadu hodnoty π, pretože samotný algoritmus metódy MapReduce je zložitejší a vyžaduje aj medzi-výpočtovú komunikáciu medzi procesmi. Pri použití štyroch uzlov tiež nedošlo k chybám a program sa ukončil správne vo všetkých prípadoch.

Časovo systém Hadoop zaostáva za implementáciou MapReduce-MPI približne 2x. Systém Hadoop je ale robustnejší a ponúka viacero funkcionalít ako napríklad redundanciu dát,

toleranciu chýb, podporuje viacero high-level jazykov, čo ale spôsobuje že nedosiahne taký výkon ako implementácia v jazyku C. Tieto funkcie sú však kľúčové v prípade veľkých dát (terabajty až petabajty), ktoré by už jednoduchšia knižnica MapReduce-MPI nezvládla.

Záver

Záverečná práca popisuje význam počítačových klastrov, hardvérové a softvérové vybavenie klastrov, typy klastrov podľa využitia a postup na vytvorenie klastra. Ukazuje postup pri vytváraní virtuálneho klastra, ktorý sa potom použije pri demonštrovaní jeho možného využitia na vysoko výkonné počítanie so správcom úloh Slurm alebo pri spracovaní veľkých dát systémom Hadoop.

Vytvorili sme funkčný virtuálny počítačový klaster, ktorý je možné použiť na vysoko výkonné počítanie ale aj spracovanie veľkých dát, čo sme overili spustením testovacích úloh. Z porovnania časov výpočtov môžeme vyvodiť záver, že pri správnom naformulovaní paralelnej úlohy, je možné niekoľko násobne zväčšiť výpočtový výkon pridaním ďalších výpočtových uzlov. Takýmto spôsobom je potom možné škálovať výkon klastra, kým nedosiahneme požadovaný výkon, vďaka ktorému sme schopný riešiť predtým neriešiteľné úlohy.

Ďalšou funkcionalitou klastra je spracovanie veľkých dát, pričom pri našom menšom testovacom datasete je jednoznačne rýchlejšia implementácia MapReduce-MPI v jazyku C. Systém Hadoop však ponúka robustnejšiu a priemyselne otestovanú implementáciu v jazyku Java, ktorej prednosti by sme ocenili pri oveľa väčšom množstve dát v datasete. Prináša výhody redundancie dát, toleranciu chýb a tiež je vývoj aplikácií vďaka podpore vysoko-úrovňových jazykov jednoduchší a dostupnejší pre širšie spektrum vývojárov.

Využitie počítačových klastrov je nepochybne dôležitým prvkom pri riešení rôznych vedeckých problémov. Tiež prináša mnohé prínosy pre biznis odvetvie, pričom mnohé firmy prechádzajú práve na cloudové riešenia, ktoré sú technicky zabezpečované práve počítačovými klastrami. Tieto medzi iným ponúkajú väčšiu dostupnosť služieb, ľahšiu škálovateľnosť a väčšiu bezpečnosť dát, častokrát s nižšími nákladmi ako pri prevádzkovaní vlastných serverov.

Zoznam použitej literatúry

[1] SALTMARSH, Abigail. 2020. *IBM's Summit—The Supercomputer Fighting Coronavirus*. MedicalExpo e-Magazine. [online]. [cit. 23.4.2021]. Dostupné na internete: <http://emag.medicalexpo.com/summit-the-supercomputer-fighting-coronavirus/>.

[2] *TOP500 Supercomuters*. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://www.top500.org/lists/top500/2020/11/>.

[3] WARWICK, Kevin. 2017. *A Brief History of Deep Blue, IBM's Chess Computer*. Mental Floss. [online]. [cit. 23.4.2021]. Dostupné na internete: <https://www.mentalfloss.com/article/503178/brief-history-deep-blue-ibms-chess-computer>.

[4] HENNESSY, John L. - PATTERSON, David A. - LARUS, James R. (1999). *Computer organization and design: the hardware/software interface (2. ed.).* San Francisco: Kaufmann. ISBN 978-1-55860-428-5.

[5] *O HPC*. [online]. [cit. 11.4.2021]. Dostupné na internete: <http://www.sivvp.sk/o-hpc>.

[6] *The 5 V's of big data*. Watson Health Perspectives. 2016. [online]. [cit. 23.4.2021] Dostupné na internete: <https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/>

[7] HELLERSTEIN, Joe. 2008. *Parallel Programming in the Age of Big Data*. Gigaom Blog. [online]. [cit. 22.4.2021]. Dostupné na internete: <http://gigaom.com/2008/11/09/mapreduce-leads-the-way-for-parallel-programming/>.

[8] HILBERT M. - LÓPEZ P. 2011. *The world's technological capacity to store, communicate, and compute information*. Science. 332. Dostupné na internete: <doi:10.1126/science.1200970>.

[9] REINSEL, David. - GANTZ, John. - RYDNING, John. 2017. *Data Age 2025: The Evolution of Data to Life-Critical*. seagate.com. Framingham. [online]. [cit. 22.4.2021] .Dostupné na internete: <https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf>.

[10] JACOBS, A. 2009. *The Pathologies of Big Data*. ACMQueue. [online]. [cit. 22.4.2021]. Dostupné na internete: <http://queue.acm.org/detail.cfm?id=1563874>.

[11] MAGOULAS, Roger. - LORICA, Ben. 2009. *Introduction to Big Data*. Release 2.0. Sebastopol CA: O'Reilly Media. [online]. [cit. 22.4.2021]. Dostupné na internete: <https://academics.uccs.edu/~ooluwada/courses/datamining/ExtraReading/BigData>

[12] KOSEČEK, M. 2015. *Čo je to big data?.* Denník N. [online]. Dostupné z : <https://dennikn.sk/blog/co-je-to-big-data/>

[13] VERNÍČEK, Marek. 2016. *Možnosti využitia Big Data pre Competitive Inteligence* : diplomová práca Praha : Vysoká škola ekonomická v Praze, 2016. 112s. [online]. Dostupné na internete: <http://theses.cz/id/50g8pe/>

[14] HOFFMAN, Chris. 2017. *10 of the Most Popular Linux Distributions Compared.* How-to-geek. [online]. [cit. 11.4.2021]. Dostupné na internete: <https://www.howtogeek.com/191207/10-of-the-most-popular-linux-distributions-compared/>

[15] LEVINE, Barry . 2013. "Linux' 22th Birthday Is Commemorated - Subtly - by Creator". Simpler Media Group, Inc. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://www.cmswire.com/cms/information-management/linux-22th-birthday-is-commemorated-subtly-by-creator-022244.php>.

[16] *Operating System Market Share Worldwide.* StatCounter Global Stats. [online]. [cit. 12.4.2021] Dostupné na internete: <https://gs.statcounter.com/os-market-share>.

[17] *Desktop Operating System Market Share*. Netmarketshare.com. [online]. [cit. 12.4.2021] Dostupné na internete: <https://netmarketshare.com/operating-system-market-share.aspx>.

[18] *Usage statistics of operating systems for websites*. W3Techs. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://w3techs.com/technologies/overview/operating_system>.

[19] LEVINE, Barry . 2013. "Linux' 22th Birthday Is Commemorated - Subtly - by Creator". Simpler Media Group, Inc. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://www.cmswire.com/cms/information-management/linux-22th-birthday-is-commemorated-subtly-by-creator-022244.php>.

[20] GROPP, William. - LUSK, Ewing. - SKJELLUM, Anthony. 1996. *A High-Performance, Portable Implementation of the MPI Message Passing Interface*. Parallel Computing. Dostupné na internete: <doi:10.1016/0167-8191(96)00024-5>.

[21] SUR, Sayantan. - KOOP, Matthew J. - PANDA, Dhabaleswar K. 2017. *MPI and communication---High-performance and scalable MPI over Infini Band with reduced memory usage.* High-performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-depth Performance Analysis. ACM. [online]. Dostupné na internete: <doi:10.1145/1188455.1188565. ISBN 978-0769527000. S2CID 818662>.

[22] *MPI: A Message-Passing Interface Standard Version 3.1*, Message Passing Interface Forum. 2015. [online] [cit. 12.4.2021] Dostupné na internete: <http://www.mpi-forum.org>.

[23] *A High Performance Message Passing Library*. OpenMPI. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://www.open-mpi.org/>.

[24] Dean, Jeffrey. – Ghemawat, Sanjay. 2004. *MapReduce: Simplified Data Processing on Large Clusters*. Google, Inc. [online]. [cit. 12.4.2021]. Dostupné na internete: <http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf>.

[25] SVERDLIK, Yevgeniy. 2014. *Google Dumps MapReduce in Favor of New Hyper-Scale Analytics System.* Data Center Knowledge. [online]. [cit. 12.4.2021]. Dostupné na internete: <http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/>.

[26] PLIMPTON, S. J. – DEVINE, K. D. 2011. *MapReduce in MPI for Large-Scale Graph Algorithms*, , Parallel Computing, 37, 610-632. [online]. [cit. 12.4.2021]. Dostupné na internete: <http://mapreduce.sandia.gov>

[27] *Quick Start User Guide*. Slurm workload manager. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://slurm.schedmd.com/overview.html>.

[28] HUDEC, Ladislav. 2014. *Bezpečnosť hardvérovej virtualizácie*. Ministerstvo financií Slovenskej republiky. [online]. [cit. 12.4.2021]. Dostupné na internete: <https://www.csirt.gov.sk/doc/MFSRVzdelavanie/01Vzdelavanie2014/10Prezentacie/Bezpecnost_hardverovej_virtualizacie.pdf>.

[29] FLEMING, Philip. – WALLACE, John. 1986. *How not to lie with statistics: the correct way to summarize benchmark results*. Commun. ACM 29, 3 (March 1986), 218–221. [online] [cit. 23.4.2021]. Dostupné na internete: <https://doi.org/10.1145/5666.5673>.

[30] *Hardware Requirements Guide.* Cloudera. [online]. [cit. 22.4.2021]. Dostupné na internete: <https://docs.cloudera.com/documentation/enterprise/release-notes/topics/hardware_requirements_guide.html>.

[31] Vlastné spracovanie podľa: *Data Age 2025*. Seagate. [online]. Dostupné na internete: <https://www.seagate.com/gb/en/our-story/data-age-2025/>.

[32] By nicoguaro - Own work, CC BY 3.0, Dostupné na internete: <https://commons.wikimedia.org/w/index.php?curid=14609430>.

Prílohy

Zdrojový kód programu na odhad hodnoty π z kapitoly 1.8.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

#define N 1E8
#define d 1E-8

int main (int argc, char* argv[])
        {
           int rank, size, error, i, result=0, sum=0;
           double pi=0.0, begin=0.0, end=0.0, x, y;

           error=MPI_Init (&argc, &argv);
           MPI_Comm_rank (MPI_COMM_WORLD, &rank);
           MPI_Comm_size (MPI_COMM_WORLD, &size);
           MPI_Barrier(MPI_COMM_WORLD);
           begin = MPI_Wtime();

           srand((int)time(0));

           for (i=rank; i<N; i+=size)
           {
              x=rand()/(RAND_MAX+1.0);
              y=rand()/(RAND_MAX+1.0);
              if(x*x+y*y<1.0)
                 result++;
           }
           // Pokračovanie na ďalšej strane
```

```
  // Pokračovanie
   MPI_Reduce(&result, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

   MPI_Barrier(MPI_COMM_WORLD);
   end = MPI_Wtime();

   if (rank==0)
   {
      pi=4*d*sum;
      printf("Počet uzlov=%2d;    Čas=%fs;    PI=%0.4f\n", size, end-begin, pi);
   }

   error=MPI_Finalize();

   return 0;
}
```