VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

INTEGRACE VÝPOČETNÍCH ZDROJŮ AMAZON COMPUTE CLOUD DO SYSTÉMU K-DISPATCH

INTEGRATION OF AMAZON COMPUTATIONAL CLOUD INTO K-DISPATCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK KUPČO**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. JIŘÍ JAROŠ, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144965

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Kupčo Patrik**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Integrace výpočetních zdrojů Amazon Compute Cloud do systému k-Dispatch**
Kategorie:	Uživatelská rozhraní
Akademický rok:	2022/23

Zadání:

1. Seznamte se s výpočetními prostředky Amazon AWS EC2 a jejich použitím pro vědecké výpočty.
2. Seznamte se se systémem k-Dispatch pro automatizované spouštění a monitorování výpočetních úloh.
3. Navrhněte a implementujte modul pro komunikaci s Amazon Compute Cloud (zadání úlohy, monitorování stavu úlohy, přenos souborů) a integrujte ho do systému k-Dispatch.
4. Navrhněte a implementujte prostředek pro vytváření alokací na Amazon Cloud pro administrátory systému k-Dispatch, např. v podobě webového rozhraní.
5. Otestujte a zdokumentujte vytvořené řešení.
6. Zhodnoťte dosažené výsledky a diskutujte přínos práce pro další vývoj balíku k-Dispatch.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Jaroš Jiří, doc. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	31.7.2023
Datum schválení:	31.10.2022

Abstrakt

Cieľom tejto práce je zoznámiť sa zo službami platformy Amazon Web Services (AWS), v hlavnom rade so službou EC2, ktorá poskytuje výpočtové prostriedky prostredníctvom cloudu. Tak isto zoznámiť sa z aktuálnym riešením systému k-Dispatch pre automatizované spúšťanie a monitorovanie výpočtových úloh, a následne navrhnúť a vyvinúť prostriedky pre integráciu služby EC2 do systému k-Dispatch. Vytvorené riešenie ponúka nástroj, pomocou ktorého je možné vytvoriť konfiguráciu pre EC2 HPC zhluk, ktorý je možné pomocou nástroju aj nasadiť. Ďalej bolo vytvorené komunikačné rozhranie, slúžiace na komunikáciu zo spomínaným zhlukom.

Abstract

The main goal of this thesis is to get introduced to the Amazon Web Services (AWS) platform, especially the EC2 compute capacity service, next goal is to get introduced to the current solution of the k-Dispatch system for automatic launch and monitoring of computing jobs, and in the end to propose and develop utilities for the integration of the EC2 service into the k-Dispatch system. The developed solution offers a tool that can be used to create a configuration for an EC2 HPC cluster, which can also be deployed using the tool. Furthermore, a communication interface has been created to communicate with the said cluster.

Kľúčové slová

k-Dispatch, AWS, Amazon Web Services, EC2, REST API

Keywords

k-Dispatch, AWS, Amazon Web Services, EC2, REST API

Citácia

KUPČO, Patrik. *Integrace výpočetních zdrojů Amazon Compute Cloud do systému k-Dispatch.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jiří Jaroš, Ph.D.

Integrace výpočetních zdrojů Amazon Compute Cloud do systému k-Dispatch

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího Jaroše Ph.D. Další informace mi poskytla pani Ing. Marta Jaroš. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Patrik Kupčo
31. júla 2023

Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce doc. Ing. Jiřímu Jarošovi Ph.D. za jeho čas a a mnoho dôležitých rád pri vedení tejto práce. Ďalej by som chcel poďakovať Ing. Marte Jaroš, ktorá je vývojárkou systému k-Dispatch, ktorá mi tiež v mnohých momentoch pomohla s otázkami ohľadom funkcionality tohto systému. V neposlednej rade patrí moja vďaka mojej rodine a blízkym, ktorý pri mne stáli po celú túto cestu.

Obsah

1 Úvod **4**

2 High Performance Computing (HPC) **6**
2.1 Moderné HPC zariadenia 6
2.2 HPC zhluky a ich architektúra 7
2.3 Plánovače úloh 7

3 Cloudové služby a Amazon Web Services **9**
3.1 Cloud Computing 9
3.2 Modely cloudových služieb 9
3.3 Amazon Web Services (AWS) 10
3.4 Amazon Elastic Compute Cloud (EC2) 11
3.5 Ďalšie služby platformy AWS 14
3.6 AWS ParallelCluster 15

4 k-Dispatch **19**
4.1 Architektúra systému k-Dispatch 19
4.2 Technológie používané v k-Dispatch 20
4.3 Popis práce s k-Dispatch 22
4.4 Spúšťanie k-Dispatch 24

5 Návrh integrácie HPC zhluku na platforme AWS do k-Dispatch **26**
5.1 Technický pohľad na funkcionalitu systému k-Dispatch 26
5.2 Reprezentácia zhlukov v k-Dispatch 28
5.3 Návrh implementácie komunikačného rozhrania pre AWS zhluk 29
5.4 Návrh integrácie AWS zhluku do databázy systému k-Dispatch 29

6 Implementácia **31**
6.1 Príprava testovacieho HPC zhluku na platforme AWS 31
6.2 Implementácia komunikačného rozhrania pre AWS zhluk do k-Dispatch . . 38
6.3 Integrácia AWS HPC zhluku do databázy 40
6.4 Implementácia nástroju na správu AWS HPC zhlukov 41

7 Testovanie **42**
7.1 Test rýchlosti zdieľaných úložísk 42
7.2 Porovnanie výkonu inštancii Intel a AMD 43

8 Záver **46**

Literatúra **48**

Zoznam obrázkov

2.1 Pomer top500 MPP a zhlukových superpočítačov 6
2.2 Ukážka HPC architektúry [21]. 8

3.1 Ukážka jednoduchej EC2 architektúry vo verejnom subnete danej VPC [14]. 12
3.2 Tvorba mnohých inštancii z jedného AMI obrazu [12]. 13
3.3 Model zdieľanej zodpovednosti [13]. 13
3.4 Ukážka VPC s verejnou a súkromnou podsieťou [28]. 15

4.1 Prihlasovacia obrazovka k-Dispatch 23
4.2 Dashboard k-Dispatch 23

5.1 Ukážka plánu v stave *NEW* 26
5.2 Ukážka plánu v stave *RUNNING* 27
5.3 Ukážka plánu v stave *FINALIZED* 27
5.4 Skrátená verzia UML reprezentácie remote_machine.py 28
5.5 Skrátená verzia ER diagramu databázy k-Dispatch 30

Kapitola 1

Úvod

High-Performance Computing (HPC) je termín, ktorý popisuje zhluky viacerých procesorov, ktoré paralelne spolupracujú na obrovských viacrozmerných výpočtoch, v rýchlostiach oveľa väčších, ako keby mali byť spracované individuálne. Pomocou tohto termínu nazývame takzvané HPC zhluky, čo sú zhluky niekoľkých počítačov, alebo serverov, ktoré sa spoločne podieľajú na výpočtoch. Takéto zhluky boli v minulosti vysoko náročné na prevádzkovanie z dôvodu vysokej ceny, alebo aj komplikovanej údržby vzhľadom na fakt, že za prevádzku zodpovedal samotný užívateľ.

Dnes však žijeme v internetovej dobe. Takmer každý počítač na svete je pripojený na internet, a vďaka tomuto faktu je dnes možné správou hardvéru poveriť tretiu stranu, ktorá poskytuje prenájom výpočtových prostriedkov po internete. Tomuto sa hovorí cloud computing, a na trhu sa dnes pohybuje mnoho firiem, ktoré ponúkajú práve takéto služby.

Táto práca sa zameriava na integráciu výpočtových zdrojov Amazon Elastic Compute Cloud (EC2) do existujúceho systému k-Dispatch. Systém k-Dispatch je v súčasnosti využívaný ako systém riadenia pracovných postupov, ktorý je zameraný na modelovanie ultrazvukových procedúr pomocou sady nástrojov k-Wave.

Prvá časť práce sa venuje bližšie k platforme Amazon Web Services (AWS), pričom sa špecificky zameriava na službu EC2 a služby, ktoré sú s ňou blízko zviazané. Ďalej sa práca zameriava na samotný systém k-Dispatch. V nej budú zhrnuté technológie, ktoré systém využíva, a bude poskytnutý prehľad architektúry a funkcionality tohto systému.

Hlavným cieľom práce je navrhnúť a vytvoriť implementáciu HPC zhluku na platforme AWS do existujúceho systému k-Dispatch. Toto umožní systému k-Dispatch využívať výpočtové zdroje, ktoré sú dodávané prostredníctvom vedúcej cloudovej platformy, poskytujúcu širokú škálovateľnosť pre záťažové požiadavky systému dostupnú v dátových centier po celom svete.

Prvým krokom bude návrh a implementácia rozšírení do systému k-Dispatch, ktoré umožnia jeho integráciu s AWS. To zahŕňa návrh a implementáciu komunikačného modulu do systému k-Dispatch, ktorý bude schopný komunikovať s HPC zhlukom vytvoreným v službe EC2.

Ďalším dôležitým bodom bude preskúmanie nástrojov ponúkaných AWS, ktoré nám umožnia efektívne spravovať a konfigurovať HPC zhluky prostredníctvom služby EC2. Na základe týchto nástrojov bude vytvorený návrh a implementácia nástroju, ktorého účelom bude tvoriť, používať, upravovať a mazať HPC zhluky rôznych konfigurácii z jedného miesta.

S touto integráciou by sa systém k-Dispatch stal univerzálnou platformou, ktorá by umožnila vedeckým pracovníkom a výskumníkom rýchly a jednoduchý prístup k modelovaciemu nástroju k-Wave prostredníctvom výpočtových zdrojov platformy AWS. Verím,

že táto práca prinesie pozitívne výsledky a posunie organizáciu na novú úroveň v oblasti výpočtového výkonu a vedeckého objavovania.

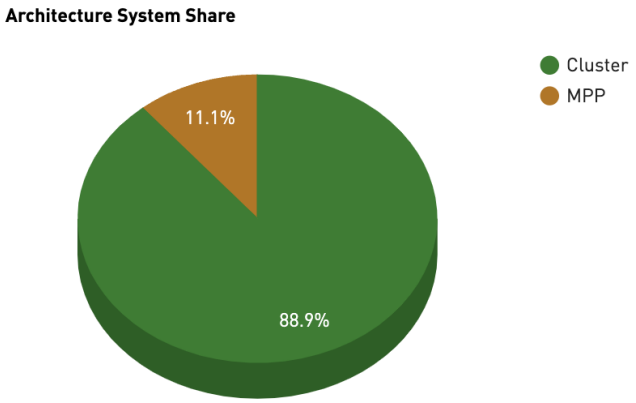
Kapitola 2

High Performance Computing (HPC)

2.1 Moderné HPC zariadenia

V skorej histórii počítačov predstavovali vrchol výkonu počítače, ktorých výpočtovú kapacitu poháňal jeden procesor, alebo neskôr niekoľko, ktoré však stále zdieľali jednu pamäť [23]. Zdieľaná pamäť je obmedzujúcim faktorom pri paralelnom prístupe viacerých procesorov kvôli ťažkostiam s koordináciou prístupu. Tento fakt robil škálovatelnosť, schopnosť systému zvládať rastúce množstvo práce, problematickou.

Moderné HPC zariadenia zo zdieľanou pamäťou, sú dnes podľa webu TOP500[1], ktorý udržiava zoznam najvýkonnejších superpočítačov na svete, dominantne dostupné v dvoch hlavných prevedeniach: Massively parallel procesory (MPP) a clustre (zhluky). Pomer ich zastúpenia v TOP500 je ukázaný na obrázku 2.1.



Obr. 2.1: Pomer top500 MPP a zhlukových superpočítačov

Hlavný rozdiel týchto dvoch prístupov spočíva v tom, že pokiaľ MPP zariadenia fungujú ako jedno zariadenie, obsahujúce mnoho procesorových jednotiek s tesným prepojením, zhluky fungujú na princípe, v ktorom sa systém skladá z viacerých nezávislých jednotiek, každá operujúc so svojím nezávislým hardvérom [25]. Táto práce sa bude ďalej zaoberať iba riešením prostredníctvom HPC zhlukov.

[1]TOP500 https://www.top500.org/

2.2 HPC zhluky a ich architektúra

HPC zhluk je skupina vzájomne prepojených vysokovýkonných počítačov, označovaných ako uzly (nodes), ktoré spoločne pracujú na vykonávaní zložitých výpočtov. Ich využitie je významné hlavne pri výpočtoch vyžadujúce paralelné spracovávanie, čo podporuje výskum v oblastiach ako bioinformatika, umelá inteligencia alebo v rôznych fyzikálnych odvetviach.

Architektúra takýchto zhlukov tradične pozostáva z nasledujúcich častí [21]:

- **Head node (Hlavný uzol)** – Hlavným bodom prístupu pre užívateľov do HPC zhluku, poskytuje prostredie na prácu so vstupmi/výstupmi výpočtov, organizuje plánovanie úloh na výpočtových uzloch.
- **Compute nodes (Výpotčové uzly)** – Vykonávajú výpočtové úlohy. Jeden uzol môže súčasne pracovať na viacerých úlohách, alebo viacej uzlov môže pracovať na jednej úlohe.
- **Accelerator nodes (Urýchlovacie uzly)** – Podobné ako výpočtové uzly, s tým rozdielom, že obsahujú hardvér určený na urýchlenie špecifických výpočtov (napríklad grafické karty, alebo iný špecializovaný hardvér).
- **Storage system (Úložiskový systém)** – Úložisko pre daný HPC zhluk, väčšinou pozostáva z úložiska na všeobecné použitie, kde sa nachádza napríklad softvér potrebný pre výpočty, a paralelného súborového systému, na ktorý sa väčšinou pripája viacero užívateľov naraz, ktorý slúži na ukladanie dát z aktuálne vykonávanej práce [30].
- **Network (Sieť)** – Sieť, v ktorej uzly operujú, pri paralelizovanej pracovnej záťaže je dôležitá nízka latencia, pri vysokej môže sieť tvoriť obmedzujúci faktor v rýchlosti výpočtov.
- **Software** – Softvér v HPC zhlukoch slúži ako na vykonávanie práce, tak aj na jej rozdelenie pomedzi uzly, prostredníctvom plánovačov úloh (popísané v 2.3).

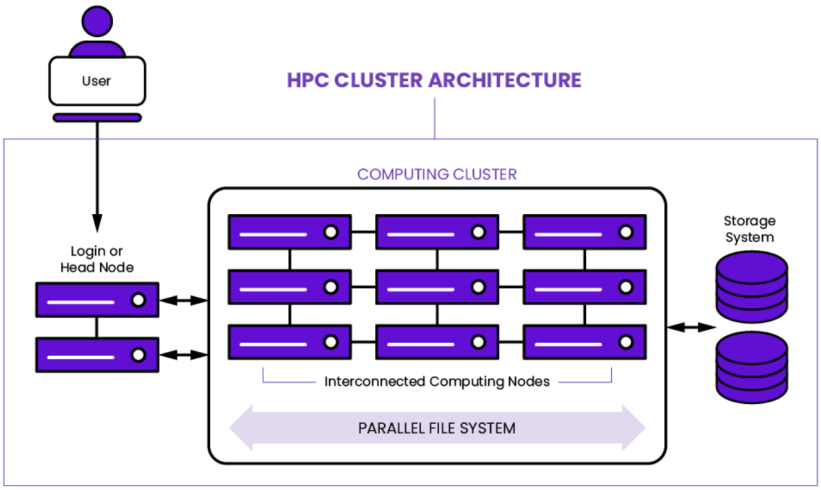
Grafická ukážka architektúry je zobrazená na obrázku 2.2.

2.3 Plánovače úloh

Jednou z najdôležitejších súčastí HPC zhlukov je plánovač úloh. Plánovač úloh je softvér, ktorý riadi vykonávanie a správu úloh zadávaných na vypracovanie v rámci konkrétneho zhluku. Užívateľ skrz plánovač zadáva zhluku úlohy a plánovač pre ne zabezpečuje priradenie výpočtových zdrojov. Zároveň užívateľovi poskytuje rozhranie, prostredníctvom ktorého je schopný dohliadať na stav prebiehajúcich úloh. V tejto sekcii budú popísané vybrané plánovače úloh, ktoré sú relevantné v kontexte tejto práce.

Slurm

Slurm je open-source softvér slúžiaci na manažment zhlukov, popri čom slúži aj ako plánovač úloh. Slurm zoraďuje výpočtové uzly v rámci zhlukov do takzvaných partícií, ktoré môžu byť považované za fronty úloh. Je schopný prijímať úlohy, v ktorých môže byť špecifikovaný konkrétny počet uzlov, jadier, pamäte alebo iných požiadavok na ich prevedenie. V tomto



Obr. 2.2: Ukážka HPC architektúry [21].

kontexte sa Slurm stará aj o to, či sú aktuálne tieto prostriedky dostupné. V prípade, že nie, vkladá úlohy do particií, skadiaľ sa eventuálne dostávajú do behu, keď sú pre nich potrebné prostriedky uvoľnené.

Kapitola 3

Cloudové služby a Amazon Web Services

Táto kapitola ponúka náhľad do služieb, ktoré poskytujú výpočtové prostriedky cez internet, často označovanými ako cloudové služby. Ďalej ponúkne pohľad do modelov použitia týchto služieb, ktoré sú v súčasnosti široko dostupné. Na záver bude pohľad na platformu Amazon Web Services, ktorá je lídrom v oblasti poskytovania cloudových služieb, a bližší pohľad na jej služby v kontexte poskytovania výpočtových prostriedkov po internete.

3.1 Cloud Computing

Pojem Cloud Computing (v preklade výpočty v oblakoch) sa prvý krát začal vyskytovať v roku 1996 vo firme Compaq Computer. Pod týmto pojmom boli označené technológie, ktoré by boli dostupné prostredníctvom siete počítačov, či už sa to týka biznis-aplikácií alebo aj bežných spotrebiteľských užívateľov [31].

Táto vízia je v dnešnej dobe realitou. Cloud Computing v dnešnej dobe chápeme, ako službu dostupnú na požiadanie, ktorá sa zaoberá dodaním výpočtových zdrojov prostredníctvom internetu, napríklad vo forme aplikácií, serverov, úložiska dát a ďalších. Tieto zdroje sú spravované na diaľku poskytovateľom cloudových služieb, ktorý ich tradične poskytuje za istý poplatok.

Hlavnými výhodami takýchto platforiem sú:

- **Zníženie nákladov** - Užívateľ nie je nútený zaobstarať si vlastný hardvér, čím šetrí na nákladoch za počiatočný nákup daného hardvéru a ďalej aj na nákladoch za administráciu týchto zariadení.
- **Flexibilita** - Užívateľ si výpočtové prostriedky v princípe len prenajíma, a tým pádom sa potencionálne vylepšenia alebo rozšírenia prostriedkov v budúcnosti stávajú jednoduchšími a menej nákladnými.
- **Dostupnosť** - Služby sú dostupné prostredníctvom internetu z akejkoľvek lokácie.

3.2 Modely cloudových služieb

Model cloudových služieb je možné deliť podľa rozsahu funkcionality, ktorú poskytujú. V dnešnej dobe identifikujeme tri hlavné modely.

Software-as-a-Service

Software-as-a-Service (SaaS) je model, pri ktorom poskytovateľ služby softvéru poskytuje svoju službu užívateľovi prostredníctvom internetu. Užívateľ samotný službu iba využíva, všetko ostatné od správy aplikácie po správu cloudovej infraštruktúry zabezpečuje poskytovateľ služby. Takéto aplikácie sú väčšinou dostupné prostredníctvom internetového prehliadača [32].

Výhodami tohto prístupu sú zjednodušenie nasadenia, vzhľadom na odstránenie potreby pre inštaláciu a správu samotného softvéru. Príkladmi tohto modelu sú služby ako Google Workspace, Dropbox alebo Cisco WebEx [32].

Platform-as-a-Service

Pri modeli Platform-as-a-Service (PaaS) užívateľ získava prístup k platforme, ktorá slúži ako framework na postavenie aplikácie prispôsobenej svojím potrebám, pokiaľ všetká správa hardvéru a platformy samotnej je v rukách dodávateľa. Platforma zákazníkovi taktiež poskytuje služby, ktoré majú užívateľovi asistovať s vývojom, testovaním a nasadením aplikácie [32].

Výhodami tohto prístupu sú jednoduchý a cenovo efektívny vývoj a nasadenie aplikácií, vysoká dostupnosť a výrazná redukcia v nutnosti programovania vlastného kódu. Príkladmi takejto platformy sú Google App Engine, Heroku alebo OpenShift [32].

Infrastructure-as-a-Service

Pri modeli Infrastructure-as-a-Service (IaaS) poskytovateľ služby poskytuje infraštruktúru vypočetných prostriedkov, čím sa chápu prostriedky ako uložisko, servery alebo siete. Užívateľ v tomto prípade spravuje prvky ako operačný systém, middleware a všetky dáta a aplikácie beziace na danej infraštruktúre [32].

Výhodami tohto prístupu sú vysoká škálovateľnosť výpočtových prostriedkov podľa potrieb, vysoká flexibilita samotného modelu a jednoduché nasadenie samotných prostriedkov. Príkladmi tohto modelu sú DigitalOcean, Linode, alebo Amazon Web Services [32].

Amazon Web Services (AWS) je cloudová platforma poskytujúca vyše 200 služieb [7] s rôznorodou funkcionalitou, ktorá beží na serveroch firmy Amazon v 30 geografických regiónoch, čo platforme umožňuje ponúkať svoje služby v 245 krajinách a územiach [4]. Vďaka tomu je dnes AWS najväčšou platformou svojho druhu, a považuje sa za jednu z najbezpečnejších, najinovatívnejších a najosvedčenejších cloudových platforiem na svete [7].

Služby na platforme AWS sú spoplatňované modelom pay-as-you-go. V praxi to znamená, že užívateľ platí len za tie prostriedky, ktoré v skutočnosti využije. Toto môže užívateľovi výrazne znížiť počiatočné náklady, vzhľadom na fakt, že užívateľovi zaniká potreba nákupu vlastného hardvéru. Tak isto tento model umožňuje užívateľovi podľa potreby upravovať množstvo prostriedkov, ktoré využíva, čím je možné ušetriť ďalšie financie.

3.3 Amazon Web Services (AWS)

S jej celkovým počtom služieb (200+) poskytuje AWS obrovskú škálu možností využitia podľa potrieb užívateľa, či už sa jedná o malý start-up, alebo popredné vládne agentúry [7]. Tieto služby sa delia podľa využitia do niekoľkých podkategórií. V nasledujúcej sekcii budú popísané služby, ktoré sú relevantné pre problematiku riešenú v tejto práci.

3.4 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (EC2) je služba platformy Amazon Web Services (AWS), ktorá poskytuje užívateľom virtuálne servery (označované ako inštancie) pre beh aplikácií v rámci infraštruktúry AWS. Ukážka jednoduchej architektúry jednej inštancie je na obrázku 3.1. Služba je vysoko škálovateľná, čo umožňuje užívateľom prispôsobovať využívané prostriedky podľa ich momentálnych výkonnostných potrieb, a to buď ich rozšírením v prípadoch vysokého zaťaženia, alebo ich redukciou v prípadoch, keď je záťaž na nižšej úrovni [14].

Inštančné typy

EC2 inštancie sú poskytované v rôznych hardvérových konfiguráciách. Tieto konfigurácie sa nazývajú inštančné typy. Inštančné typy spadajú do rôznych kategórií podľa ich konkrétnej špecializácie. Z pohľadu užívateľa je dôležité vybrať si zo správnej kategórie na základe jeho požiadavok. Medzi jednotlivé kategórie inštančných typov patria:

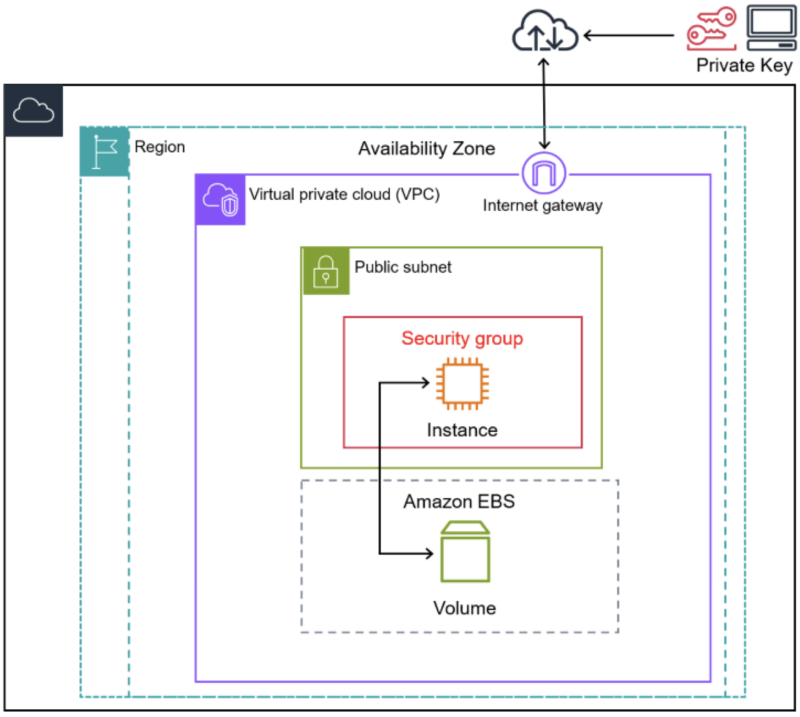
- **General Purpose** – Inštancie na všeobecné použtie, vyvážený pomer medzi pamäťou, procesorom a sieťovou kapacitou.
- **Compute Optimized** – Zamerané hlavne na čistú výpočtovú kapacitu.
- **Memory Optimized** – Zamerané na výpočty, ktoré sú namáhavé na pamäť.
- **Accelerated Computing** – Inštancie ponúkajú na urýchlenie výpočtov grafické karty, alebo rôzne proprietárne hardvérové akcelerátory. Tie sú výhodné na graficky namáhavé výpočty, alebo výpočty zaoberajúce sa trénovanom umelej inteligencie [11].
- **Storage optimized** – Zamerané na prácu vyžadujúcu vysoký počet vstupno-výstupných operácií nad veľkými súbormi dát na lokálnom úložisku.
- **HPC Optimized** – Účelovo postavené na beh HPC výpočtov, ideálne pre aplikácie, ktoré požadujú vysoko výkonne procesory.

Inštančné typy sa môžu líšiť v hardvérovej konfigurácii po mnohých stránkach. EC2 ponúka procesory od firiem AMD a Intel, k dispozícii sú však aj procesory AWS Graviton, ktoré bežia na architektúre ARM [17], alebo Mac inštancie s natívnou podporou pre MacOS [1]. Ďalšie prostriedky ako napríklad RAM, úložný priestor a sieťová kapacita sú taktiež ponúkané v rôznych konfiguráciách, v závislosti od konkrétneho inštančného typu.

Amazon Machine Image (AMI)

Softvérová výbava inštancií je zabezpečovaná v podobe obrazov nazývaných Amazon Machine Image (AMI). Tieto obrazy poskytujú šablónu, ktorá obsahuje softvérovú konfiguráciu. Tá v sebe zahŕňa operačný systém, nainštalované aplikácie, konfigurácie, alebo ďalšie častí. Z AMI obrazu je možné vytvoriť neobmedzené množstvo inštancii, ako je ukázané na obrázku 3.2.

AWS ponúka mnoho preddefinovaných obrazov, čo umožňuje rýchle rozbehnutie novej inštancie. AMI je však možné vygenerovať z hociktorej aktuálne bežiacej inštancie EC2, čo umožňuje ľahkú tvorbu východzích softvérových konfigurácii pre užívateľov. Vďaka tomuto existuje celá rada AMI obrazov vytvorených komunitou na rozličné prípady použitia [12].



Obr. 3.1: Ukážka jednoduchej EC2 architektúry vo verejnom subnete danej VPC [14].

Zabezpečenie

Hlavnou prioritou služby pôsobiacej v "oblaku", akou je aj EC2 musí byť bezpečnosť. U AWS to nie je inak, keďže ich služby sú využívané v mnohých kritických oblastiach rôznych priemyslov.

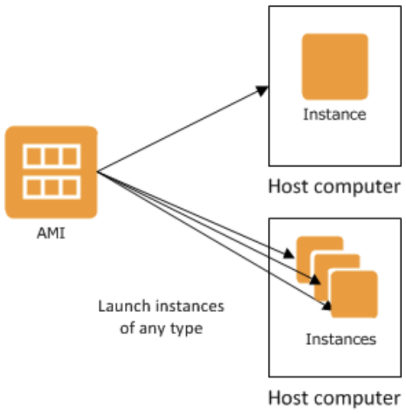
Pri ohľade na bezpečnosť je potrebné vziať do úvahy takzvaný Shared Resposibility Model[1] (v preklade model zdieľanej zodpovednosti). Tento model delí zodpovednosť za bezpečnosť na dve časti: Security of the Cloud (zabezpečnie cloudu) a Security in the Cloud (zabezpečnie v cloude) [13]. Detaily tohto modelu sú popísané na obrázku 3.3.

Za zabezpečenie cloudu sa v tomto modeli zodpovedá AWS samotné. Spočíva v zabezpečení infraštruktúry Cloudu, ktorá spočíva z hardvéru, softvéru, siete a zariadenia, v ktorých ostatné spomínané častí infraštruktúry sídlia [13].

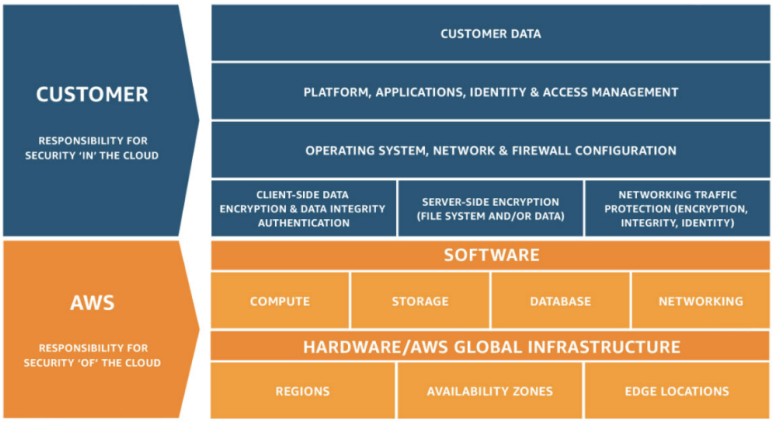
Za zabezpečenie v cloude zodpovedá používateľ. To znamená, že používateľ musí aktívne prevziať úlohu zabezpečenia vlastných zdrojov v prostredí cloudu. Táto úloha z pohľadu služby EC2 pozostáva z nasledujúcich zodpovedností [9]:

- Kontrolovanie sieťového prístupu k užívateľským inštanciám (prostredníctvom VPC, alebo bezpečnostných skupín, viz. obrázok 3.1).
- Správa poverení používaných na pripájanie ku inštanciám.
- Správa operačného systému, a softvéru nainštalovanom na inštanciách.
- Konfigurovanie IAM rolí pripojených ku inštanciám a povolení, ktoré sú k rolám priradené.

[1]Shared Responsibility model https://aws.amazon.com/compliance/shared-responsibility-model/



Obr. 3.2: Tvorba mnohých inštancii z jedného AMI obrazu [12].



Obr. 3.3: Model zdieľanej zodpovednosti [13].

Bezpečnostné skupiny fungujú ako virtuálny firewall, ktoré na základe im priradených pravidiel obmedzujú prichádzajúcu a odchádzajúcu sieťovú komunikáciu. Jedna inštancia môže byť naraz vo viacerých skupinách. Ak užívateľ pre konkrétnu inštanciu nezvolí žiadnu skupinu, inštancii sa priradí predvolená skupina danej VPC, v ktorej sa inštancia nachádza [10].

Cena služby EC2

Cena používania EC2 sa odvíja od dvoch hlavných faktorov. Prvým z nich je inštančný typ, z ktorého je inštancia vytvorená a druhým faktorom je nákupný model, ktorý bol zvolený na zakúpenie inštancii. Medzi tieto modely spoplatnenia patria:

- **On-Demand** – Užívateľ za používané prostriedky, platí fixnú cenu buď za každú využitú hodinu alebo minútu, v závislosti od konkrétneho typu inštancie, ktorá beží.
- **Spot** – Umožňuje užívateľovi využívať EC2 prostriedky, ktoré aktuálne nie sú využívané až za 90% zľavu. Služba EC2 si v tomto prípade vyhradzuje možnosť zobrať si prostriedky späť s dvojminútovým varovaním.

- **Savings Plan** – Užívateľ si dopredu na dobu jedného alebo troch rokov rezervuje istú výpočtovú kapacitu, pri výmene za znížené ceny.
- **Dedicated Host** – Užívateľ si rezervuje konkrétny fyzický server na jeho využitie. Toto môže užívateľovi vyhovovať, ak potrebuje mať na serveri softvér, ktorého licencia je limitovaná na jeden stroj.

3.5 Ďalšie služby platformy AWS

Amazon Elastic Block Store (EBS)

Amazon Elastic Block Store (EBS) poskytuje službe EC2 perzistentné blokové úložisko pre dáta [2]. Perzistentné v tomto zmysle znamená, že daný EBS zväzok môže existovať aj mimo životnú dobu jemu priradenej EC2 inštancie, a je ho možné pripojiť aj na iné inštancie. Blokové úložisko znamená, že každý blok sa správa ako samostatné zariadenie pre ukladanie dát (napr. pevný disk).

EBS zväzky môžu byť prevádzkované pomocou rôznych zariadení podľa potreby využitia. AWS poskytuje rôzne typy EBS zväzkov bežiacich na pevných diskoch a aj SSD diskoch. Tieto typy sa môžu líšiť v rýchlostiach, cenách a rôznych ďalších parametroch.

Jednotlivé zväzky sú taktiež veľmi flexibilné - majú schopnosť dynamicky zvýšiť svoju kapacitu, prispôsobiť svoju rýchlosť a zmeniť typ zväzku. Cena za túto službu sa odvíja od viacerých parametrov, ako napríklad celkové využitie zväzkov v gigabytoch, počet vstupno-výstupných operácií za mesiac a ďalších [2].

Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) je úložisková služba na ukladanie objektov, ktorá poskytuje špičkovú škálovateľnosť, dostupnosť dát, bezpečnosť a výkon [15].

Objekt v službe S3 reprezentuje kombináciu súboru a s ním spojených metadát. S3 taktiež umožňuje verzovanie jednotlivých objektov, vďaka čomu je možné udržiavať aj staršie verzie rovnakého objektu.

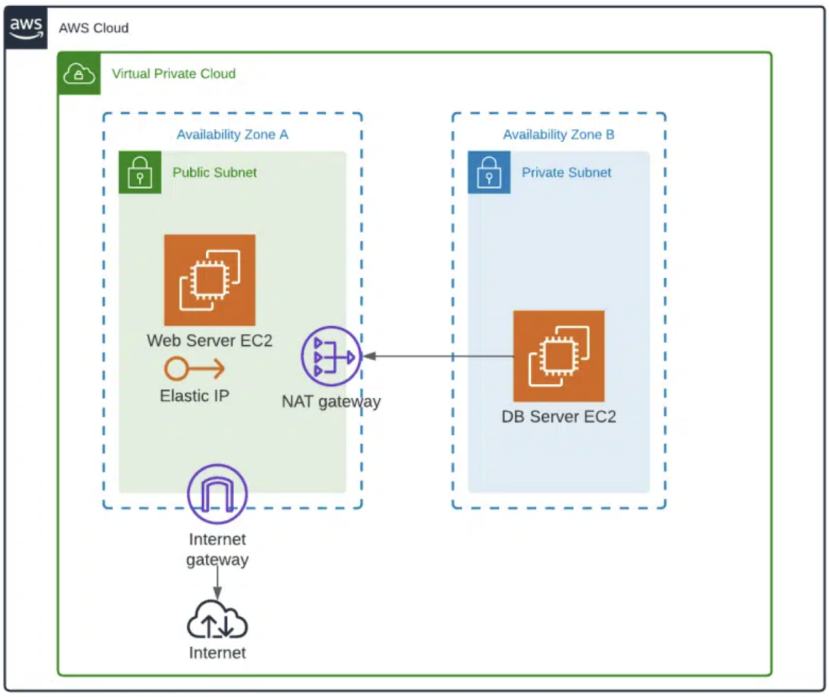
Každý objekt je unikátne označovaný pomocou kľúča a je ukladaný do kontajneru nazývaného bucket. Tieto buckety musia mať unikátny názov medzi všetkými ostatnými bucketmi vrámci jednotlivých AWS partícií. V čase tvorby tohto dokumentu sa AWS skladá z troch partícií: aws (Štandardné regióny), aws-cn (Čínske regióny) a aws-us-gov (buckety vlády USA) [15].

Objekt môže byť unikátne identifikovaný pomocou svojho kľúča a bucketu v ktorom je (prípadne ešte pomocou ID verzie, ak je verzovanie v buckete zapnuté) [15].

Cena služby sa odvíja od celkovej veľkosti uložených dát v gigabytoch a triedy uloženia. Triedy uloženia slúžia na archiváciu objektov, ktoré nevyžadujú okamžitý prístup (v milisekundách). Toto môže znížiť finančné náklady na archiváciu niektorých súborov vo výmene za predĺžený čas prístupu (od 1 minúty až po 12 hodín).

Amazon Virtual Private Cloud (VPC)

Služba Amazon Virtual Private Cloud (Amazon VPC) poskytuje funkcionalitu, ktorá umožňuje tvorbu virtuálnych sietí v rámci cloudového priestoru platformy AWS. Tieto virtuálne siete svojimi schopnosťami emulujú tvorbu sieti v skutočnom živote [16].



Obr. 3.4: Ukážka VPC s verejnou a súkromnou podsieťou [28].

Novy VPC cloud sa rozprestiera naprieč celý región, v ktorom bol vytvorený. V rámci cloudu je možné tvoriť podsiete. Podsieť je rozsah IP adries, v rámci ktorého je možné tvoriť rôzne AWS zdroje, ako napríklad EC2 inštancie. Jedna podsieť môže sídliť v iba jednej zóne dostupnosti konkrétneho regiónu [28]. Podsiete je možno deliť na verejné a súkromné. Ukážka VPC s oboma druhmi podsieti je na obrázku 3.4.

Verejné podsiete sú tie, ktoré majú priame pripojenie k internetu, sprostredkované prostredníctvom internetovej brány. Tieto brány umožňujú zariadeniam v rámci podsiete prístup na internet pod podmienkou, že zariadeniu bola pridelená verejná IP adresa, čím sa zariadenie stáva prístupným online. Táto prístupnosť sa ale vzťahuje aj na iné zariadenia na internete, čím sa tieto zariadenia stávajú zraniteľnými voči vonkajším útokom

Súkromné siete nemajú priamy prístup na internet. Namiesto toho sa zariadenia v rámci súkromnej podsiete pripájajú na internet prostredníctvom brány NAT. Táto brána NAT, ktorá sa nachádza vo verejnej podsieti, umožňuje internetovú komunikáciu prostredníctvom verejnej IP adresy brány NAT. Tieto zariadenia však nemôžu prijímať prichádzajúcu externú komunikáciu, keďže nemajú nastavenú verejnú IP adresu. Brána NAT iba maskuje súkromné adresy IP zariadení, ktoré ju používajú, a umožňuje len odchádzajúce spojenia, čím zvyšuje bezpečnosť súkromnej siete.

3.6 AWS ParallelCluster

Narozdiel od vyššie spomínaných služieb, AWS ParallelCluster je nástroj, ktorý slúži na zjednodušenie práce pri nasadzovaní a správe HPC zhlukov. Automaticky nastavuje potrebné výpočtové prostriedky, plánovač a zdieľaný súborový systém [18].

Tento nástroj je možné používať viacerými spôsobmi. Prvým je jednoduché grafické rozhranie (GUI) prevedené ako webová aplikácia, ktorá slúži ako nástenka (dashboard) na vytváranie, monitoring a správu jednotlivých zhlukov. Toto rozhranie bolo pridané do AWS ParallelCluster vo verzii 3.5.0 [6]. Druhým spôsobom je vo forme aplikácie v príkazovom riadku (CLI), vytvorenom v programovacom jazyku Python, kde sa jednotlivé konfigurácie zlukov nahrávajú skrz konfiguračný súbor. Ďalším spôsobom je použitie ParallelCluster API rozhrania, ktoré umožňuje programatický prístup k tomuto nástroju.

Aplikácia je dodávaná v dvoch verziách: verzii 2 a verzii 3. Medzi týmito verziami sú veľmi zásadné rozdiely. Medzi ne patria [24]:

- **GUI webová aplikácia** – Keďže UI bolo pridané až vo verzii 3.5.0, verzia 2 podporuje čisto len prácu s CLI aplikáciou.
- **ParallelCluster API** – Vo verzii 3 bolo pridané na prácu API, čo umožňuje jednoduchší programatický prístup ku ParallelCluster
- **Syntax príkazového riadku** – CLI aplíkácia používa rozdielne príkazy vo verzii 2 a 3, vo verzii 3 bližšie reflektujú možné volania skrz API.
- **Konfiguračné súbory** – Konfiguračné súbory pre CLI aplikáciu majú výrazne rozdielny syntax, a tým pádom nie sú medzi sebou bez úprav kompatibilné.
- **Podpora plánovačov úloh** – Verzia 2 podporuje plánovače úloh: SGE, Slurm, Torque (PBS) a AWS Batch. Vo verzii 3 podpora pre SGE a Torque bola úplne stiahnutá

```
[cluster default]
fsx_settings = fsx
...

[fsx fsx]
shared_dir = /shared-fsx
fsx_fs_id = fsx_fs_id
```

Výpis 3.1: Ukážka konfigurácie vo v2

```
...
SharedStorage:
  - Name: fsx
    MountDir: /shared-fsx
    StorageType: FsxLustre
    FsxLustreSettings:
      FileSystemId: fsx_fs_id
```

Výpis 3.2: Ukážka konfigurácie vo v3

Funkcionalita AWS ParallelCluster

Hlavnou úlohu AWS ParallelCluster je zjednodušenie vytvárania infraštruktúry výpočtových zhlukov. Toto dosahuje vďaka použitiu AWS služby CloudFormation. Táto služba poskytuje jazyk na modelovanie infraštruktúry v cloudovom prostredí. Zbierka zdrojov modelovaných ako jedna jednotka sa nazýva stack. Zdroje takéhoto stacku sú definované vo vnútri CloudFormation šablóny. AWS ParallelCluster vytvára infraštruktúru HPC zhlukov reprezentovanú jedným CloudFormation stackom.

Vo väčšine prípadov jednotlivé CLI príkazy v AWS ParallelCluster priamo korešpondujú na jednotlivé AWS CloudFormation stack príkazy, ako napríklad *create update* alebo *delete* [18].

Jednotlivé uzly v týchto zhlukoch sú reprezentované pomocou inštancií bežiacich vrámci služby Amazon Elastic Compute Cloud (EC2). Na vrchu sa nachádza takzvaný "head nodeälebo teda hlavný uzol.

Hlavný uzol je reprezentovaný EC2 inštanciou, ktorej hlavnou úlohou je prijímať jednotlivé úlohy, ktoré majú byť v danom zhluku vyhodnotené. Plánovač bežiaci na hlavnom uzle takýto dotaz spracuje a na základe potrieb práce sa vytvoria výpočtové uzly takisto reprezentované inštanciami EC2, ktoré túto prácu vyhodnocujú.

Konfigurácie jednotlivých HPC zhlukov sú definované prostredníctvom konfiguračných súborov. V týchto súboroch sú definované nastavenia, ako napríklad použitý plánovač úloh, nastavenia zdieľaných úložísk medzi uzlami v zhlukoch, ale aj napríklad inštančné typy jednotlivých uzlov (ako hlavných, tak i výpočtových).

```
Region: eu-central-1
Image:
  Os: ubuntu2004
HeadNode:
  InstanceType: t2.micro
  Networking:
    SubnetId: subnet-xxxxxxxxxxxxxxxx
  Ssh:
    KeyName: EC2_key_pair
Scheduling:
  Scheduler: slurm
  SlurmQueues:
  - Name: queue1
    ComputeResources:
    - Name: t2micro
      Instances:
      - InstanceType: t2.micro
      MinCount: 0
      MaxCount: 10
    Networking:
      SubnetIds:
      - subnet-xxxxxxxxxxxxxxxx
```

Výpis 3.3: Jednoduchá konfigurácie zhluku vygenerovaná v CLI aplikací

Analýza metód prístupu k AWS ParallelCluster

Nástroj ParallelCluster poskytuje pre užívateľa rozličné prostriedky k prístupu, umožňujúce variabilné metódy použitia v závislosti na požiadavkách užívateľa. V tejto sekcii budú bližšie popísané tie najrelevantnejšie, a bude nad nimi vykonaná analýza ich prípadov využitia a limitácií.

AWS ParallelCluster CLI

AWS ParallelCluster command line interface (CLI) je hlavnou metódou, ktorou je táto aplikácia distribuovaná k užívateľom. Jedná sa o balík vytvorený v programovacom jazyku Python, ktorý je bežiaci v príkazovom riadku na užívateľovom počítači. Túto aplikáciu je možné nainštalovať dvomi rôznymi spôsobmi.

- Ako Pythonovskú knižnicu, pomocou správcu balíkov *pip*

- Ako samostatnú aplikáciu zo stránky dodávateľa.

Táto aplikácia ponúka užívateľom nástroj na vytváranie, zobrazovanie, modifikáciu a odstraňovanie HPC zhlukov v rámci rôznych konfiguračných možností služieb AWS. Užívateľ interaguje s aplikáciou prostredníctvom sady príkazov priradených jednotlivým funkciám. Architektúry týchto HPC zhlukov sú reprezentované pomocou YAML súborov, ktoré umožňujú reprezentáciu konfigurácie vo forme zrozumiteľnej pre ľudské oko.

Jednou z hlavných funkcií tejto aplikácie je schopnosť generovať takéto konfiguračné súbory, z ktorých aplikácia vytvára nimi opísané zhluky. Jej princíp spočíva v kladení série otázok užívateľovi, z ktorých odpovedí aplikácia generuje konfiguračný súbor. Týka sa to však len nevyhnutných položiek pre vytvorenie zhluku. Pre dodatočnú konfiguráciu je potrebný manuálny zásah do konfiguračného súboru.

Z konfiguračného súboru je možné prostredníctvom aplikácie vygenerovať konečný zhluk, ktorý potom užívateľ môže ďalej monitorovať, manipulovať alebo prípadne zmazať. Podstata fungovania tejto aplikácie ostáva podobná aj naprieč ostatnými prostriedkami pre prácu s ParallelCluster, ostatné sa však pokúšajú riešiť nedostatky tejto aplikácie.

Medzi nedostatky aplikácie patria pomerne neintuitívna interaktivita so samotnou aplikáciou pri manipulácii so zhlukmi, kde sa všetky operácie vykonávajú prostredníctvom značného počtu príkazov (približne 25). Ďalej, programatický prístup k aplikácií prostredníctvom príkazov nie je príliš priateľský pre programátorov. Amazon však poskytuje aj iné metódy prístupu k ParallelCluster, ktoré tieto problémy riešia.

AWS ParallelCluster API

Rozhranie na programovanie aplikácií (API) je softvérové rozhranie, ktorý umožňuje komunikáciu dvoch aplikácií vrámci pevne navrhnutej špecifikácie na konkrétne správy (požiadavky). Prostredníctvom takéhoto rozhrania môže jedna aplikácia odoslať požiadavku druhej aplikácii, ktorá ju spracuje a vráti odpoveď späť opäť cez rozhranie bez nutnosti odhaľovať viac informácií na obidvoch stranách, ako je nevyhnutné [27]. Týmto spôsobom API umožňujú efektívnu a bezpečnú komunikáciu naprieč rôznymi systémami

Nástroj ParallelCluster poskytuje svoju vlastnú implementáciu rozhrania na programovanie aplikácií nazývanou AWS ParallelCluster API. Táto aplikácia je distribuovaná v podobe šablóny pre AWS CloudFormation, ktorá slúži na jej zhotovenie vo vnútri platformy AWS. Využíva na to službu Amazon API Gateway, ktorá poskytuje vstupný bod do rozhrania, sprístupňujúc jeho jednotlivé funkcie, a AWS Lambda, ktorá riadi vyhodnotenie týchto funkcií bez potreby ďalších výpočtových prostriedkov, ako napríklad serveru.

Od verzie 3.5.0. je dostupná verzia rozhrania aj v podobe knižnice pre programovací jazyk Python, kde sa dodáva ako pribalené vrámci CLI aplikácie [5]. Táto verzia rozhrania poskytuje vývojárom sadu funkcií totožnú s tou poskytovanou prostredníctvom AWS ParallelCluster API, bez potreby viazať sa jej funkcionalitou na ďalšie služby AWS platformy.

Kapitola 4

k-Dispatch

V tejto kapitole bude popísaný systém k-Dispatch, jeho funkcionalita a prínosy, ktoré ponúka v práci s modelovaním ultrazvukových procedúr v oblastí biomedicíny.

k-Dispatch je systém, ktorého účelom je zjednodušiť komplexnú prácu spočívajúcu vo vyhodnocovaní fyzikálnych modelov, ktoré kvôli ich výpočtovej komplexnosti je potrebné vyhodnocovať prostredníctvom High-performance computing (HPC) zhlukov, ktorých obsluha je pre bežného užívateľa vysoko komplexná.

Program je postavený na akustickom nástroji k-Wave, ktorý modeluje biomedicínske ultrazvukové postupy. Takéto postupy, označované ako liečebné plány, sú nahrávané do systému k-Dispatch, ktorý ich v prvom rade dekóduje, následne optimalizuje parametre vykonávania toku práce, odosiela úlohy na vzdialené výpočtové zariadenia, monitoruje ich priebeh a zaznamenáva spotrebované hodiny jadier [29].

Kedže systém k-Dispatch sa používa na prácu s modelmi vytvorenými na medicínske účely, je tým pádom považovaný za medicínske zariadenie, a teda sa na jeho používanie vyžadujú prísne požiadavky na kvalitu a manažment hrozieb. Kvôli tomuto systém funguje na princípe preddefinovaných šablón pre jednotlivé procedúry, ktoré sú vyhodnocované certifikovanými binárnymi súbormi, ktoré môžu byť do systému pridané len autorizovaným personálom [29].

Bežní používatelia na základe týchto šablón vytvárajú jednotlivé procedúry a odosielajú ich do k-Dispatch systému. Ten dekóduje procedúru, zostavuje tok práce, vyberá vhodné zdroje a optimalizuje výpočtové zdroje pre úlohy s cieľom minimalizovať čas a náklady [29]. k-Dispatch monitoruje úlohy na anomálie (napríklad zamrznuté úlohy) a umožňuje sa z nich v záujme procedúry zotaviť.

Po vyhodnotení liečebného plánu k-Dispatch umožňuje užívateľovi stiahnuť výsledky do svojho zariadenia a vyčistí za sebou miesto na zhluku, kde výpočet prebiehal. Vďaka tomuto procesu sú užívatelia ochránení pred komplexnou prácou s HPC zhlukom, ako je napríklad odosielanie úloh, politiky fronty a výber zdrojov. Automatické procesy eliminujú potrebu, aby používatelia nastavovali počet uzlov a jadier, volili medzi výpočtami na CPU alebo GPU, alebo odhadli čas výpočtu [29].

4.1 Architektúra systému k-Dispatch

Architektúra systému sa delí na 3 základné moduly: Webový server, jadro, a databázu. Tieto 3 moduly sú bližšie popísané v tejto sekcii:

Webový server

Webový server slúži ako grafické rozhranie, pomocou ktorého je užívateľ schopný pracovať zo systémom k-Dispatch. Toto rozhranie je postavené v programovacom jazyku Python, špecificky vo webovom frameworku Flask.

Jeho hlavnou úlohou je poskytnúť užívateľovi rozhranie, skrz ktoré je schopný nahrávať nové liečebné plány do systému, sledovať ich aktuálny stav a v prípade korektného vypočítania stiahnuť výsledky do svojho zariadenia.

Po nahraní liečebného plánu do systému sa súbor reprezentujúci plán uloží na lokálne úložisko serveru, a tak isto sa vytvára v databáze nový záznam o pláne. Na základe týchto záznamov potom ďalej jadro jednotlivé plány zozbieráva, a prevádza nad nimi požadované operácie. Počas tohoto procesu úlohou webového serveru zostáva získavať stav plánov v databáze, a informovať o nich užívateľa.

Po dokončení vyhodnocovania plánu je užívateľ prostredníctvom webového serveru informovaný o tom, že sa tak udialo, a ďalej webový server výsledok tohto behu umožňuje stiahnuť v podobe súboru užívateľovi do jeho zariadenia, uvoľní miesto na lokálnom disku serveru a v databáze upraví záznam patriaci plánu do správneho stavu.

Jadro

Ďalším modulom systému k-Dispatch je jadro. Tento modul je podobne ako aj webový server vytvorený v programovacom jazyku Python, a slúži v podstate ako back-end celého systému.

Toto jadro sa dá ďalej rozdeliť na tri hlavné podmoduly: Daemon, Monitor a Dispatch & Transfer [29].

Daemon je najjednoduchší z týchto modulov, jeho jedinou prácou je konfigurácia logovania, a spúšťanie monitoru v pravidelných časových intervaloch. Slúži však hlavne ako vstupný bod pre celú funkcionalitu jadra, a umožňuje registrovanie k-Dispatch ako služby v operačnom systéme [29].

Monitor modul má za úlohu pri svojej invokácii preskúmať databázu ohľadom aktuálnych stavov jednotlivých liečebných plánov. Na základe týchto stavov sa nad plánmi volajú rôzne funkcie z modulu Dispatch & Transfer.

Dispatch & Transfer je hlavným modulom celého jadra. Tento modul zjednocuje prístup k rôznym výpočtovým zdrojom a ich plánovačom [29].

4.2 Technológie používané v k-Dispatch

Vo vývoji systému k-Dispatch bolo použitých viacero technológií, ktoré sú pre túto prácu relevantné. V tejto sekcii budú bližšie popísané.

SSH

SSH (Secure Shell) je protokol, ktorý umožňuje zabezpečené prihlasovanie, prenos súbrov a ďalšie typy zabezpečených dátových komunikácií bežiacich na nezabezpečenej sieti. Na zabezpečenie komunikácie protokol SSH používa kryptografickú autentizáciu, šifrovanie relácie a ochranu integrity [33].

Bežným použitím protokolu SSH je v dnešnej dobe je možnosť ovládania a úpravy vzdialeného zariadenia zabezpečene po internete. Príkladom takéhoto využitia v systéme k-Dispatch je komunikácia systému so vzdialenými zhlukmi, na ktorých pomocou tohto

protokolu je možné zadávať úlohy do plánovačov, a taktiež kontrolovať ich aktuálny stav, prípadne zotaviť z neočakávaného stavu.

SSH sa v systéme k-Dispatch taktiež používa aj na nahrávanie súborov zo systému na zhluk. Toto je dosiahnuté pomocou nástroju rsync, ktorý používa práve protokol SSH. Tento nástroj slúži na synchronizáciu súborov medzi lokálnym a vzdialeným zariadením. V praxi to znamená, že užívateľ je schopný prekopírovať súbory z konkrétneho umiestnenia z lokálneho zariadenia na vzdialené bez toho, aby bolo potrebné kopírovať súbory, ktoré už na vzdialenom zariadení existujú.

Fabric

Fabric je Pythonovská knižnica, ktorá slúži na zjednodušenie používania SSH v Pythonovských programoch. Je postavená na knižnici Paramiko, ktorá poskytuje SSH funkcionalitu ako správu kľúčov, alebo SSH relácie, a na knižnici Invoke, ktorá poskytuje vykonávanie príkazov v shelli a iné operácie [26]. Z pohľadu k-Dispatch Fabric umožňuje používanie protokolu SSH priamo z jeho modulov pre prácu so zhlukmi.

Docker

Docker je open-source platforma, ktorá slúži na vytváranie virtuálnych prostredí bežiacich nad operačným systémom. Dosahuje to pomocou spúšťania softvéru v takzvaných kontajneroch. Tieto fungujú podobne ako napríklad virtuálne stroje, kde sa taktiež vytvára prostredie bežiace nad operačným systémom (OS), ktoré s ním zdieľa výpočtové prostriedky. Rozdiel spočíva v tom, že narozdiel od virtuálnych strojov, v ktorých je potrebné spúšťať celý OS aj s hypervízorom, v kontajneroch je zabalený len základ OS potrebný na spúšťanie procesov a závislosti potrebné pre softvér, ktorý bude v kontajneri spustený [19].

Jednotlivé kontajneri je možné vytvárať pomocou Docker image-ov. Tieto slúžia ako šablóny, v ktorých je popísané, čo sa má v kontajneri zostaviť. Vďaka image-om sú jednotlivé zostavy vysoko prenosné, keďže pomocou jedného image-u je možné zostrojiť rovnaké prostredie v kontajneri na akomkoľvek stroji. Tieto image sa vytvárajú pomocou Dockerfile-ov. Toto sú textové súbory, v ktorých sú jednotlive príkazy, ktoré Docker vykoná pri zostavení image-u. Tieto príkazy sú obdobné tým, ktoré by užívateľ zadával do príkazového riadku.

Docker-Compose

V bežnom používaní Dockeru často nastáva situácia, kde je potrebné vytvoriť viaceré prostredia, ktoré spolu tvoria jeden systém, ako napríklad v k-Dispatch medzi webovým serverom, jadrom a databázou. Na uľahčenie takejto práce slúži nástroj docker-compose. Tento nástroj užívateľovi umožňuje napísať YAML súbor, v ktorom sú definované služby, ktoré majú v systéme fungovať. Následne je Docker z takýchto súborov schopný systém zostaviť, spustiť, a zastaviť ako celok pomocou jedného príkazu.

Conda

Conda je open-source správca balíčkov a prostredí pre jazyk Python. Pomocou tohto nástroju je možné vytvárať si v počítači jednotlivé vývojárske prostredia, ktoré môžu v sebe obsahovať rôzne špecifické verzie ako balíčkov, tak aj samotného jazyku Python. Conda umožňuje jednoduché prepínanie medzi jednotlivými prostrediami, ale aj iné operácie, ako napríklad zdieľanie prostredí pomocou konfiguračných YAML súborov.

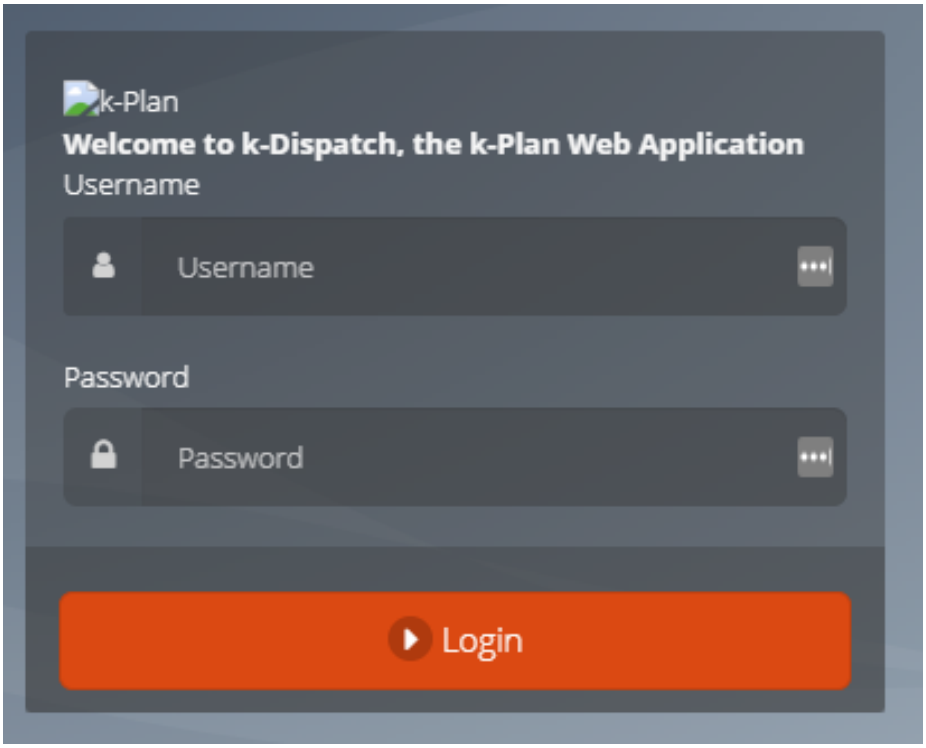
Pôvodne Conda bola súčasťou Pythonovskej distribúcie Anaconda, ktorá v sebe obsahuje radu balíčkov na vedecké výpočty v oblastiach, ako sú dátová veda, alebo umelá inteligencia. Časom sa Conda ukázala ako samostatne užitočný nástroj, a tak bola odštiepená do samostatného open-source nástroju [20], ktorý sa dnes používa ako správca balíčkov nie len s jazykom Python, ale aj jazykmi ako R, Ruby, C/C++ a ďalšími [8].

4.3 Popis práce s k-Dispatch

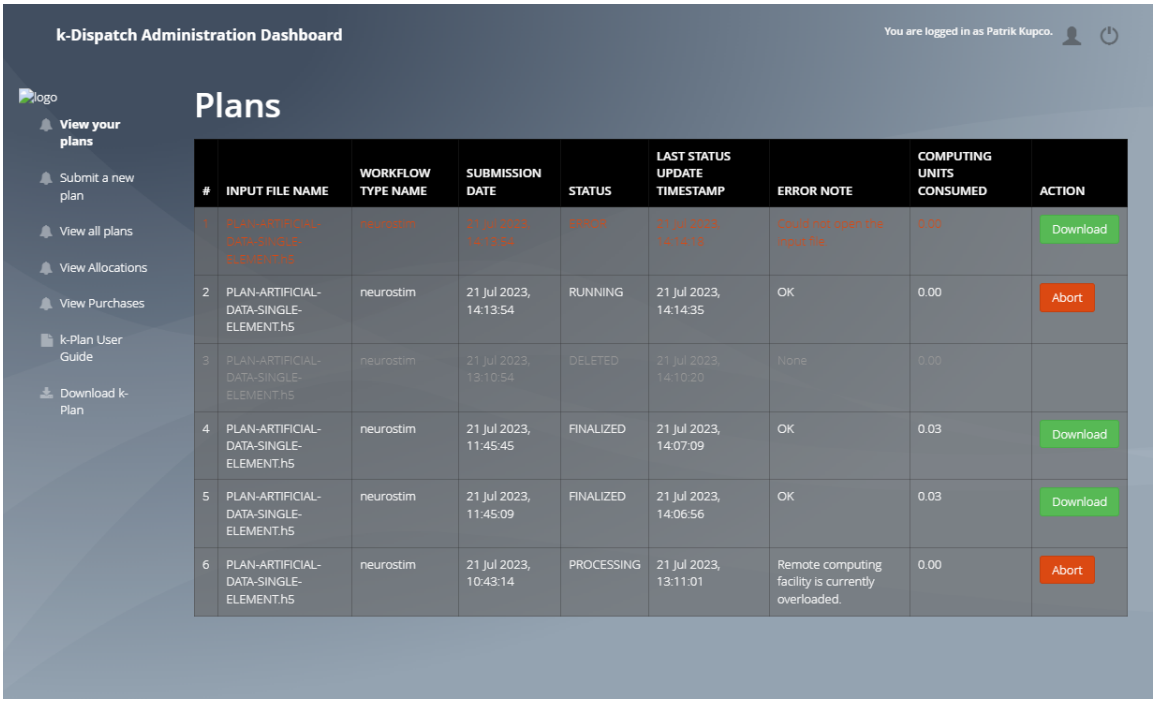
Hlavným rozhraním pre prácu zo systémom k-Dispatch je jeho webové rozhranie. Toto po spustení programu podľa návodu vyššie bude dostupné na doméne *localhost:8080*. Mimo vývojového prostredia bude toto webové rozhranie dostupný prostredníctvom interetu. Prvá vec, s ktorou sa tu užívateľ stretne je prihlasovacia obrazovka zobrazená v 4.1. Po zadaní svojho užívateľského mena a k nemu priradeného hesla je úživateľ do systému prihlásený.

Po prihlásení sa užívateľ dostáva do hlavného dashboardu systému zobrazeného v 4.2. Tu si môže prehliadnuť všetky liečebné plány, ktoré boli zadané na vypracovanie. Po ľavej strane sa nachádza menu s možnosťami na prezeranie svojich alokácií a svojich nákupov. Najdôležitejšou položkou tu je možnosť vložiť do systému nový liečebný plán. Touto cestou užívateľ do systému nahrá nový plán, ktorý sa tým pádom nahráva na úložisko serveru, a vytvára sa o ňom nový záznam v databáze

Po zadaní plánu do systému je o tomto pláne vytvorený záznam v databáze, ktoré v pravidelných intervaloch jadrom systému k-Dispatch zbiera na ďalšie spracovávanie. Tento sofistikovaný proces je pred užívateľom schovaný, ten je schopný progres spracovávanie sledovať skrz webové rozhranie. Po dokončení vyhodnocovania sú súbory z výsledku spracovávania zozbierané zo vzdialeného zhluku, zabalené do zipového archívu, a stiahnuté na úložisko systému k-Dispatch. Užívateľovi je umožnené tieto výsledky stiahnuť prostredníctvom webového rozhrania na jeho ďalší rozbor.



Obr. 4.1: Prihlasovacia obrazovka k-Dispatch



Obr. 4.2: Dashboard k-Dispatch

4.4 Spúšťanie k-Dispatch

Zdrojové kódy na zostavenie systému k-Dispatch mi boli poskytnuté mojím vedúcim práce, pánom docentom Jiřím Jarošem. Systém ako celok sa dá rozdeliť na tri služby, ktoré treba spustiť osobitne: Webový server, jadro, a databáza.

Na spustenie týchto častí sa ponúkajú dve metódy: Pomocou nástroju Docker, alebo lokálne rozbehnutie jednotlivých služieb.

k-Dispatch na Dockeri

V zdrojovom kóde systému k-Dispatch sa nachádza zložka *docker*. V tejto zložke sa nachádzajú všetky súbory potrebné pre spustenie k-Dispatch v dockerizovanom prostredí. Najdôležitejší je docker-compose.yml, ktorý použije rovnomenný nástroj na prípadné vytvorenie image-ov z pribalených Dockerfile-ov, a taktiež aj ich spustenie v kontajneroch.

Pre vytvorenie image-ov z príslušných Dockerfile-ov sa použije príkaz *docker-compose build*. Na spustenie samotných kontajnerov z image-ov sa použije príkaz *docker-compose up* Po spustení všetkých kontajnerov je systém spojazdnený.

Za zmienku stojí ešte spomenúť, že aj keď sa v zložke nachádzajú tri Dockerfile-y, spúštajú sa až štyri služby. To kvôli tomu, že v docker-compose je ešte definovaná služba *db*, ktorá bude slúžiť ako PostgreSQL databáza. Pre ňu sa vytvára kontajner z už pred-vytvoreného image-u *sameersbn/postgresql*

Spúšťanie systému v rámci platformy Docker výrazne uľahčuje spúšťanie programu pre nového užívateľa, avšak prináša aj isté nevýhody. Z pohľadu vývoja je proces úpravy kódu a následného čakania na zastavenie kontajnerov, nové zostavenie image-ov a ich spustenie v kontajneroch zdĺhavý proces.

k-Dispatch spustený z lokálnych zdrojov

V prípade, že užívateľ plánuje robiť v zdrojovom kóde časté úpravy, a chce ich následne otestovať v najkratšej možnej dobe, väčší zmysel dáva spúšťanie tohto systému na užívateľovom stroji lokálne. Keďže systém beží v jazyku Python, tieto zmeny sa stanú prevedenými po uložení zdrojového súboru (webový server musí byť spustený s argumentom –debug, aby sa zmeny po uložení automaticky načítali). Pri používaní Dockeru by bolo potrebné na prevedenie zmien potrebné previesť zdĺhavý proces zastavenia všetkých kontajnerov, vytvorenie nových image-ov, a z nich opätovného spustenia nových kontajnerov.

Pre zabezpečenie správneho vývojárskeho prostredia sa výrazne doporučuje použiť správcu balíčkov Conda. V zdrojovom kóde sa nachádzajú dva YAML súbory obsahujúce všetky závislosti potrebné pre beh jednotlivých súčastí systému. Prvým je súbor *DSM-py35.yml*, z ktorého je možné vytvoriť prostredie bežiace na verzii jazyku Python 3.5.5, a druhým je *DSM-py38.yml* z ktorého sa vytvára prostredie na verzii Python 3.8.13.

Popis lokálneho spúšťania

V tomto popise sa predpokladá, že na stroji, kde sa tento systém bude spúšťať už je nainštalovaná Conda, DBMS PostgreSQL a užívateľ ma vybrané, ktorú verziu prostredia bude chcieť použiť. V nasledujúcich krokoch bude popísané, ako postupovať pri vytváraní prostredia z priloženého YAML súboru, a taktiež aj ako spustiť jednotlivé časti systému

1. Vytvoriť Conda prostredie

```
conda create -n <nazov_prostredia> --file <cesta_k_YAML_suboru>
```

2. Aktivovať Conda prostredie

```
conda activate <nazov_prostredia>
```

3. Prepnúť sa do zložky so zdrojovými kódmi
4. (Nepovinné) V prípade potreby testovacích dát v databáze spustiť inštalačný skript

```
python install_db.py DATA_DOCKER_TEST
```

5. Spustenie jadra

```
python daemon.py
```

6. Spustenie Webserveru

```
exec gunicorn -c gunicorn_config.py -b 0.0.0.0:8000 wsgi:app
```

Kapitola 5

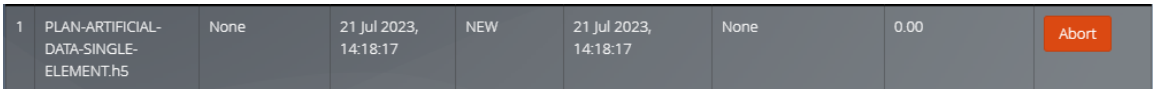
Návrh integrácie HPC zhluku na platforme AWS do k-Dispatch

Prvým kľúčovým aspektom tejto práce je návrh implementácie komunikačného modulu, ktorý zabezpečí požadovanú úroveň interakcie medzi systémom k-Dispatch a HPC zhlukom umiestneným na serveroch AWS. Tento proces je stredobodom nasledujúcej kapitoly,v ktorej bude priblížená funkcionalita systému k-Dispatch z pohľadu interakcie s aktuálne používanými riešeniami HPC systémov, implementácia nástrojov na komunikáciu s nimi, a na záver návrh podobného nástroju, ktorý bude komunikovať so zhlukmi prevádzkovanými na platforme AWS.

5.1 Technický pohľad na funkcionalitu systému k-Dispatch

Proces spracovávania liečebného plánu v systéme k-Dispatch začína nahraním konkrétneho plánu do systému pomocou jeho webového rozhrania. Počas tohto procesu sa v databáze ku plánu vytvára príslušný záznam popisujúci jeho vlastnosti. Jedna z týchto vlastností je jeho stav.

Nové plány sú v databáze klasifikované pod stavom *NEW* zobrazený na obrázku 5.1. Ostatné plány sa taktiež nachádzajú v rozličných stavoch. Ďalšie spracovávanie plánov následne prebieha v závislosti od týchto stavov v jadre systému k-Dispatch.



Obr. 5.1: Ukážka plánu v stave *NEW*

V jadre prebieha v pravidelných intervaloch monitoring databázy, špecificky sa dotazuje v akých stavoch sa nachádzajú samotné plány. Podľa nich sú plány ďalej posúvané funkciám systému na ďalšie spracovanie.

Plány v stave *NEW* sú po zozbieraní uvedené do nového stavu *PROCESSING*, a sú posunuté na počiatočné spracovanie. V rámci procesu spracovávania je jeden z krokov programu aj výber kandidátnych alokácií k danému plánu. Kandidátne alokácie sú tie, ktoré sú dostupné na vyhodnotenie liečebného plánu. Tieto alokácie sú vyberané na základe kontroly doby platnosti a kontroly dostupných výpočtových hodín v rámci alokácie. V prípade, že sú obe kontroly splnené je alokácia zvolená medzi kandidátne.

Ku každej alokácii sa viaže istý počet HPC zhlukov, v jednej ich môže byť priradených viacero. V testovacích dátach, ktoré mi boli ku systému k-Dispatch poskytnuté sa nachádza jedna ukážková alokácia, ktorá v sebe zahŕňa dva dostupné HPC zhluky: Karolina[1] a Barbora[2]. Tieto zhluky sú prevádzkované Technickou Univerzitou v Ostrave v rámci superpočítačového centra IT4Innovations[3].

Po výbere kandidátnych alokácií pokračuje tok programu vytvorením grafu úloh. Ním je popísaný postup úloh, ktoré sa majú v rámci plánu vykonať. Jeden z krokov tejto akcie je zvolenie finálnej alokácie z pomedzi kandidátnych. Takáto alokácia sa stane tou, ktorej prostriedky sa budú využívať v procese vyhodnocovania plánu.

V rámci výberu alokácie sa taktiež vyberie z alokácie aj HPC zhluk, na ktorom vyhodnotenie prebehne. Výber HPC zhluku alokácie prebieha v troch krokoch:

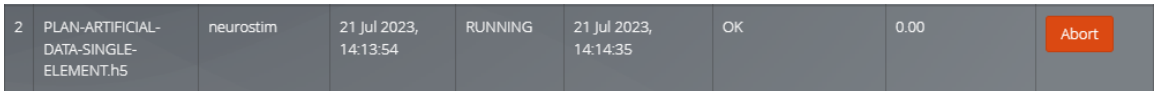
1. **Tvorba inštancie triedy** - V rámci tvorby inštancie triedy reprezentujúcej daný zhluk sa deje pokus o pripojenie pomocou SSH protokolu 4.2. Ak je neúspešné, inštancia sa nevytvorí, a zhluk sa nevyberie.

2. **Kontrola počtu odovzdaných úloh** - k-Dispatch má nastavený limit maximálneho počtu úloh, ktoré má užívateľ zadané v jeden moment. Ak by zadanie ďalšej tento limit prekročilo, zhluk sa nevyberie.

3. **Vyhodnotenie** - Zhluky, ktoré predchádzajúce dva kroky splnia sú vyhodnotené na základe odhadovanej ceny pre výpočet. Zhluk s najlepším hodnotením je vybraný.

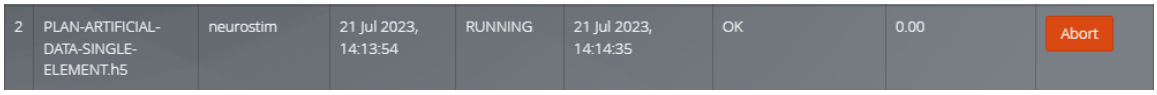
Po zvolení finálnej alokácie a HPC zhluku na vyhodnotenie liečebného plánu pokračuje systém generovaním skriptov pre jednotlivé úlohy vyplývajúce zo spomínaného grafu úloh. V rámci tejto operácie sú skripty nahrávané pomocou nástroja *rsync* na zhluk zvolený v výpočtu, a taktiež sú aj v zhluku predložené na spracovanie.

Po predložení všetkých úloh prechádza celkový plán do stavu *RUNNING* zobrazeného na obrázku 5.2, v ktorom je pri každom periodickom priebehu monitoringu kontrolovaný na stav úloh, kde pri prípadnom zlyhaní sa systém pokúša z tohto stavu plán zotaviť.



Obr. 5.2: Ukážka plánu v stave *RUNNING*

V prípade že všetky úlohy patriace liečebnému plánu sú úspešne vyhodnotené, považuje sa tým pádom aj celý plán za kompletný, a dostáva sa do stavu *FINALIZED* zobrazeného na obrázku 5.3.



Obr. 5.3: Ukážka plánu v stave *FINALIZED*

Systém k-Dispatch zozbiera z HPC zhluku, na ktorom bol plán vyhodnotený súbory, ktoré sú spojené z jeho vyhodnotením, zabalí ich do archívu vo formáte *zip*, nahrá na lokálne

[1]Karolina hhttps://www.it4i.cz/infrastruktura/karolina
[2]Barbora https://www.it4i.cz/infrastruktura/barbora
[3]IT4Innovations https://www.it4i.cz/.

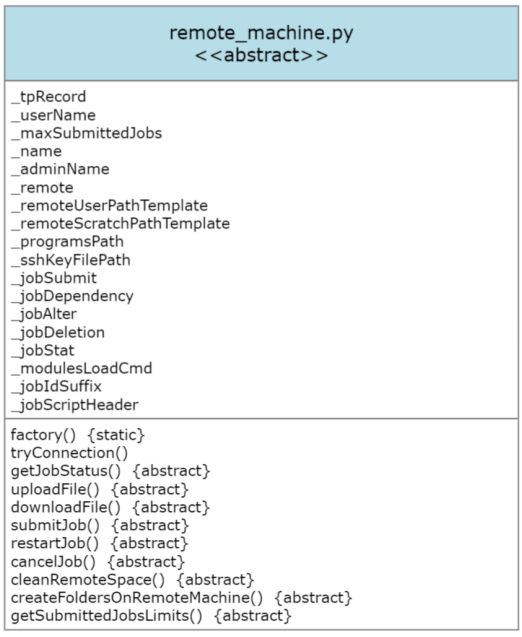
úložisko systému k-Dispatch a z lokálneho úložiska HPC zhluku ich zmaže. Užívateľovi sa takéto plány vo webovom rozhraní zobrazujú ako hotové, a ponúkne sa mu možnosť priradený *zip* archív stiahnuť do svojho zariadenia.

5.2 Reprezentácia zhlukov v k-Dispatch

Jednotlivé HPC zhluky sú v systéme k-Dispatch modelované pomocou objektovo orientované prístupu, kde každý zhluk je reprezentovaný ako rozšírenie abstraktnej triedy *remote_machine.py*. Táto trieda sa nachádza v projektovom priečinku *remotemachine*. V tejto zložke sa taktiež nachádza rada už existujúcich implementácií triedy *remote_machine.py*, ktoré reprezentujú rôzne HPC zhluky, a poskytujú pre ne ich špecifickú funkcionalitu a nastavenia.

Abstraktná trieda *remote_machine.py* (UML reprezentácia na obrázku 5.4) je navrhnutá tak, aby poskytovala univerzálnu šablónu pre ďalšie implementácie nových HPC zhlukov. Obsahuje širokú sadu parametrov, ktoré poskytujú možnosti na prispôsobenie funkcionality konkrétneho zhluku. Trieda *remote_machine.py* ďalej obsahuje aj sadu abstraktných metód, ktoré je potrebné implementovať v rozšíreniach. Tieto metódy je povinné implementovať v každom potomkovi abstraktnej triedy, v súlade s konkrétnymi potrebami daného HPC zhluku a softvéru, ktorý je na ňom nainštalovaný.

Okrem abstraktných metód sa nachádzajú v triede *remote_machine.py* implementácie *getter* a *setter* funkcií pre čítanie a úpravu atribútov inštancií, implementácia triedy reprezentujúca výnimky typu *RemoteMachineException*. Na záver sa tu nachádza statická metóda *factory()*, ktorej význam spočíva v tvorbe tvorby inštancií rôznych potomkov abstraktnej triedy *remote_machine.py*, a už implementovaná metóda *tryConnection*, slúžiaca na overenie konektivity na HPC zhluk pomocou protokolu SSH 4.2.



Obr. 5.4: Skrátená verzia UML reprezentácie remote_machine.py

5.3 Návrh implementácie komunikačného rozhrania pre AWS zhluk

Z predchádzajúcej sekcie vyplýva, že ak je potreba do systému uviesť nový HPC zhluk, je v prvom rade potrebné navrhnúť implementáciu potomka triedy *remote_machine.py* (UML reprezentácia v 5.2). Pre jednoduchosť bude ďalej označovaný ako komunikačné rozhranie. Implementácia tohto modulu bude spočívať v dvoch krokoch:

1. Implementovaní abstraktných metód rodičovskej triedy
2. Nastavovaní parametrov v novom komunikačnom rozhraní na hodnoty umožňujúce komunikáciu so vzdialeným HPC zhlukom

Pre začiatok je vhodné pozrieť sa na implementáciu už existujúcich komunikačných rozhraní. Implementácie pre HPC zhluky Barbora a Karolína sú svojou prakticky identické, čiže je možne zvoliť ktorýkoľvek ako referenčný.

Implementácie jednotlivých abstraktných metód v referenčnom module fungujú na princípe spúšťania príkazov na diaľku na vzdialenom HPC zhluku. Túto funkcionalitu poskytuje knižnica Fabric, popísaná v 4.2, obsahujúca rozhranie ktoré umožňuje exekúciu shellovských príkazov na vzdialenom zariadení prostredníctvom SSH.

Abstraktné metódy sú v referenčných moduloch navrhnuté tak, aby väčšina príkazov bola zostavená z parametrov definovaných v komunikačnom rozhraní. Metódy týmto postupom tvoria šablónu, ktorá je vypĺňaná priradenými parametrami komunikačného rozhrania. Toto umožňuje úpravu volaných príkazov čisto cez atribúty. Na základe tohto faktu je možné usúdiť, že ako základ nového komunikačného rozhrania môže pre jednoduchosť slúžiť kópia už existujúceho referenčného rozhrania.

Najzásadnejším rozdielom medzi referenčným komunikačným rozhraním, a navrhovaným rozhraním pre HPC zhluk na AWS je ten, že existujúce rozhrania sú navrhnuté pre HPC zhluky používajúce plánovač úloh OpenPBS. Konfigurácia nového HPC zhluku bude generovaná prostredníctvom nástroja AWS ParallelCluster, ktorý tento plánovač nepodporuje. Miesto toho bude v novom HPC zhluku implementovaný plánovač Slurm. Na základe tohto rozdielu bude potrebné vykonať v navrhovanom komunikačnom rozhraní radu zmien, ktorá sa bude týkať využitia nového plánovača úloh.

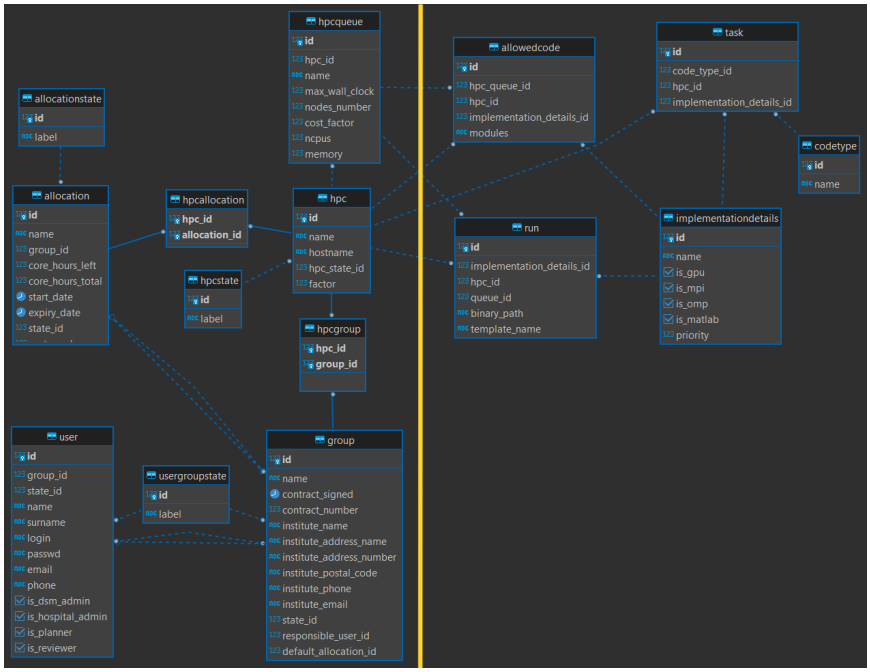
5.4 Návrh integrácie AWS zhluku do databázy systému k-Dispatch

Poslednou fázou integrácie AWS HPC zhluku do systému k-Dispatch je vloženie záznamov týkajúcich sa tohto zhluku do SQL databázy samotného systému. Na základe údajov v nej je pri spracovávaní nahraného liečebného plánu mimo iné vyberaný HPC zhluk podľa postupu popísanom v sekcii 5.1.

Návrh integrácie do systému k-Dispatch bude vychádzať z ukážkových dát poskytnutých v súbore *database/docker_testing_data.py*. Na základe nich je možné rozbehnúť systém k-Dispatch s podporou HPC zhlukov Karolina a Barbora.

Dáta sú v súbore reprezentované pomocou knižnice *peewee*[4], čo je knižnica poskytujúca objektovo relačné mapovanie (ORM) pre jazyk Python. ORM je technika, ktorá slúži na

[4] https://docs.peewee-orm.com/en/latest/



Obr. 5.5: Skrátená verzia ER diagramu databázy k-Dispatch

mapovanie dát z tabuliek v relačných databázach do objektov v objektovo orientovaných programovacích jazykoch [22].

Tabuľky sú reprezentované ako rozšírenia triedy *BaseModel*, ktorá je samotná rozšírenie *peewee* triedy *Model*. Trieda *Model* poskytuje programátorovi rozhranie na prevádzanie CRUD operácii (Create, Read, Update, Delete) nad tabuľkami v databáze, kým trieda *BaseModel* ju rozširuje o referenciu na pripojenie k databáze systému k-Dispatch.

Úpravy, ktoré budú v databáze prevedené sa skladajú z dvoch častí, ktoré budú vykonané nad tabuľkami popísanými v skrátenom ER diagrame v obrázku 5.5.

Prvá časť sa nachádza v ľavej polovici vyznačenej žltou čiarou. V nej sa nachádzajú údaje týkajúce sa registrácie zhluku do systému k-Dispatch. Integrácia začne vytvorením záznamu o HPC zhluku (*hpc*), ktorá je súčasťou alokácie (*allocation*) a užívateľskej skupiny (*group*).

V ukážkových dátach sa nachádza ako aj alokácia, tak aj užívateľská skupina, ktorá má v sebe vytvorené funkčné užívateľské účty. Do nich bude pridaný nový AWS HPC zhluk práve pomocou ich spojenia vrámci databázy. Taktiež je potrebné vytvoriť záznamy pre fronty v zhluku prostredníctvom tabuľky *hpcqueue*

V druhej časti vyznačenej na pravej strane diagramu sú tabuľky udržujúce dáta, ktoré sa týkajú registrácie binárnych súborov na používanie pre vyhodnocovanie liečebných plánov. Táto časť pozostáva z vytvorenia záznamov v troch tabuľkách:

1. *task* – popisuje, aké úlohy je možné spustiť na rôznych HPC zhlukoch
2. *allowedcode* – popisuje, v ktorých frontách konkrétnych HPC zhlukov môže byť binárny súbor spustený
3. *run* – popisuje umiestnenie binárneho súboru na HPC zhluku, a ktorá šablóna sa má vybrať na generáciu skriptu na jeho spustenie

Kapitola 6

Implementácia

V nasledujúcej kapitole bude popísaná implementačná časť tejto práce. Prvá sekcia je tvorbu testovacie HPC zhluku na platforme AWS. Zameriava sa na ukážku generovania konfiguračného súboru pre daný zhluk pomocou AWS ParallelCluster CLI aplikácie so sadou úprav pre potreby systému k-Dispatch.

Ďalej táto kapitola dáva do pozornosti implementáciu komunikačného rozhrania pre AWS HPC zhluk do systému k-Dispatch. Pomocou neho sú zhluku zadávané úlohy na vypracovania, je sledovaný ich stav, sú preberané ich výsledky a podobne. Na záver sa v tejto kapitole nachádza popis dát vložených do databázy, potrebný pre korektnú registráciu AWS HPC zhluku do systému k-Dispatch.

6.1 Príprava testovacieho HPC zhluku na platforme AWS

Pred začiatkom implementácie nástrojov na riadenie HPC zhlukov vytvorených na platforme AWS, je v prvom rade potrebné jeden takýto zhluk vytvoriť, a pripraviť na integráciu do systému k-Dispatch.

Na vytvorenie tohto HPC zhluku bude použitá AWS ParallelCluster CLI aplikácia, ktorá bola spomenutá tu 3.6. Pomocou nej je možné vytvoriť základnú konfiguráciu, na báze ktorej bude vytvorený výsledný HPC zhluk. Pre štart interaktívneho sprievodcu konfiguráciou slúži príkaz:

```
pcluster configure --config <nazov_konfiguracneho_suboru.yaml>
```

Tento sprievodca poskytuje možnosti na konfiguráciu základných nastavení pre zhluk menovite:

- Region zhluku
- Pár SSH kľúčov pre prístup ku zhluku
- Plánovač úloh
- Operačný systém
- Typ inštancie hlavného uzlu
- Nastavenia pre fronty podľa vybraného plánovača úloh (počet front, ich názvy, v prípade slurm inštančné typy výpočtových prostriedkov a podobne)
- VPC a jej podsieť, v ktorom bude zhluk operovať

Základná konfigurácia

V nasledujúcej sekcii budú spomínané rozhodnutia za konfiguračnými nastaveniami, v rámci ktorých bude generovaný nový testovací HPC zhluk. Ku každej významnej časti bude priložená pre ilustráciu ukážka výpisu z AWS ParallelCluster CLI aplikácie ku danému nastaveniu.

Region zhluku vyberieme pre ukážku *eu-central-1*, kvôli jeho blízkej lokácii relatívne ku miestu vypracovania tejto práce (Brno). Na použitie páru SSH kľúčov je potrebné tento pár mať zaregistrovaný k svojmu AWS účtu. Toto sa dá prostredníctvom AWS EC2 managementovej konzoly. Tá poníka možnosť vytvorenia nového páru alebo aj importovania už existujúceho v užívateľovom počítači.

```
Allowed values for AWS Region ID:
1. af-south-1
...
9. eu-central-1
...
AWS Region ID [eu-central-1]:
Allowed values for EC2 Key Pair Name:
1. ssh-key-pair-1
2. ssh-key-pair-2
EC2 Key Pair Name [ssh-key-pair-1]:
```

Výpis 6.1: Ukážka výberu regiónu a páru SSH kľúčov

Vo verzii 3 nástroju ParallelCluster je obsiahnutá podpora dvoch plánovačov úloh: Slurm a AWS Batch. AWS Batch funguje na princípe tvorenia Dockerovských kontajnerov, v ktorých sú vyhodnocované zadávané úlohy. Od nášho zhluku požadujeme beh úloh na EC2 inštanciách, čo je spôsob, ktorým funguje v ParallelCluster plánovač Slurm. Z tohto dôvodu je ďalej vyberaný plánovač Slurm

```
Allowed values for Scheduler:
1. slurm
2. awsbatch
Scheduler [slurm]:
```

Výpis 6.2: Ukážka výberu plánovača úloh

V nástroji ParallelCluster je operačný systém pre uzly v zhluku poskytovaný prostredníctvom sady EC2 AMI obrazov 3.4. ParallelCluster podporuje aj zhluky vytvorené s vlastne upravenými AMI obrazmi, na túto prácu bude však využitý pripravený obraz s operačným systémom Ubuntu 20.04.

```
Allowed values for Operating System:
1. alinux2
2. centos7
3. ubuntu1804
4. ubuntu2004
```

```
5. rhel8
Operating System [alinux2]:ubuntu2004
```

Výpis 6.3: Ukážka výberu operačného systému

Ako inštančný typ pre hlavný uzol bol zvolený *t2.micro*. Hlavným dôvodom jeho zvolenia je fakt, že patrí do AWS Free Tier. AWS Free Tier umožňuje užívateľom vyskúšať si niektoré AWS služby na bezplatných prostriedkoch. V prípade t2.micro ide až o 12 mesiacov bezplatného prístupu do 750 hodín za mesiac na Linuxových strojoch [3]. Tento typ poskytuje veľmi limitovaný, ale perfektne dostatočný výkon na správu réžie malého počtu uzlov v HPC zhluku.

```
Head node instance type [t2.micro]:
```

Výpis 6.4: Ukážka výberu inštančného typu hlavného uzlu

V konfigurovanom HPC zhluku bude pre potreby testovania postačovať jedna fronta. Táto fronta sa bude menovať *qprod*. Názov tejto fronty je dôležitý, pretože systém k-Dispatch sa pokúša práce, na ktorých bude tento zhluk testovaný posielať do práve tejto fronty.

Do fronty je možné priradiť ľubovoľný počet výpočtových prostriedkov. Každý výpočtový prostriedok predstavuje jeden typ inštancie, podľa ktorého sa vo fronte budú vytvárať výpočtové uzly, a určitý maximálny počet uzlov tohto typu.

Pre nový HPC zhluk počet výpočtových zdrojov obmedzíme na jeden, a ten bude t2.xlarge. Tento bol zvolený kvôli svojej relatívne nízkej cene, a dostatočnému výkonu na vyhodnotenie jednoduchých úloh v rozumnom čase. Porovnanie s t2.micro (hlavným uzlom) je možné pozorovať v tabuľke 6.1.

```
Number of queues [1]:
Name of queue 1 [queue1]:
Number of compute resources for queue1 [1]:
Compute instance type for compute res 1 in queue1 [t2.micro]: t2.xlarge
Maximum instance count [10]:
```

Výpis 6.5: Ukážka tvorby fronty pre HPC zhluk

	t2.micro	**t2.xlarge**
Max. rýchlosť CPU	3.3 GHz	3.0 GHz
Počet vCPU	1	4
Pamäť	1 GB	16 GB
Cena za hodinu	$0.0116	$0.1856

Tabuľka 6.1: Popis inštančného typu t2.micro(hlavný uzol) a t2.xlarge(výpočtové uzly)

Na záver sprievodcu je potrebné priradiť zhluk do VPC (popísané v 3.5), v ktorej bude HPC zhluk zaobalený. AWS ParallelCluster CLI aplikácia poskytuje možnosť tvorby nového VPC pre zhluk, pre túto prácu bude postačovať predvolený, ktorý je každému užívateľovi vytvorený v každom AWS regióne. Aplikácia taktiež poskytuje možnosť vytvoriť subnety pre VPC, tu však ale opäť zvolíme možnosť zvolenia z predvolených subnetov v danej VPC.

```
Automate VPC creation? (y/n) [n]: n
Allowed values for VPC ID:
  # id name number_of_subnets
--- --------------------- ------------------------------- ------------------
  1 vpc-xxxxxxxxxxxxxxxxx ParallelClusterVPC-xxxxxxxxxxxxx 1
  2 vpc-yyyyyyyyyyyyyyyyy ParallelClusterVPC-yyyyyyyyyyyyy 3
VPC ID [vpc-yyyyyyyyyyyyyyyyy]: 1
```

Výpis 6.6: Ukážka výberu VPC pre HPC zhluk

Výstupom z tohto sprievodcu je konfiguračný súbor vo formáte yaml, ktorý je možné použiť na vygenerovanie nového HPC zhluku. Ešte pred tým, než ho spustíme je však potrebné pridať ďalšie konfiguračné nastavenia, ktoré budú od zhluku požadované, menovite zdieľané úložisko a nastavenia pre pripojenie na MySQL databázu, ktorá bude slúžiť na udržovanie dát o účtovníctve pre Slurm, čo bude využité na perzistenciu údajov o dokončených úlohách. Tieto nastavenia je potrebné do konfiguračného súboru doplniť manuálne.

Zdieľané úložisko

Nastavenie konfigurácie pre zdieľané úložisko je pomerne triviálne. Do konfiguračného súboru pridáme sekciu *SharedStorage*, ktorá je zobrazená nižšie.

```
SharedStorage:
  - StorageType: Ebs
    MountDir: /sharedStorage
    Name: matlab-runtime-volume
    EbsSettings:
      VolumeType: gp3
      Size: 1000
      DeletionPolicy: Retain
  - StorageType: Efs
    MountDir: /sharedStorageEfs
    Name: shared-volume
```

Výpis 6.7: Sekcia konfigurácie pre zdieľané úložisko

V tejto sekcii je možné vidieť dva záznamy pre dve rôzne zdieľané úložiská. Každý záznam reprezentuje úložisko, ktoré sa má počas tvorby HPC zhluku vytvoriť taktiež. Tieto dve úložiská budú tie, ktoré budú vygenerované pre AWS HPC zhluk.

Prvé je úložisko prostredníctvom služby EBS. EBS ponúka blokové úložisko, ktoré fungujú na princípe podobnom virtuálnych pevných diskov pre inštancie EC2. Na tomto úložisku budú udržiavané binárne súbory potrebné pre vyhodnocovanie liečebných plánov zadaných do systému k-Dispatch. Ďalej tu bude inštalácia MATLAB Runtime[1], čo je softvér potrebný na rozbiehanie aplikácii skompilovaných v programovacom jazyku MATLAB[2]. V ňom sú niektoré zo spomínaných binárnych súborov napísané.

Druhé je úložisko zo služby EFS. Toto bude slúžiť ako dočasné úložisko pre bežiace výpočty v rámci vyhodnocovania liečebných plánov. Toto úložisko bolo k tomuto účelu zvolené, pretože poskytuje oveľa vyššiu priepustnosť ako EBS zväzok. EFS sa taktiež veľkosťou adaptuje svojím aktuálnym požiadavkám, a tak užívateľ platí len za to, koľko využije.

[1] MATLAB Runtime https://www.mathworks.com/products/compiler/matlab-runtime.html
[2] MATLAB https://www.mathworks.com/products/matlab.html

Medzi položky, ktoré sa nastavujú na oboch úložiskách patrí typ úložiska, jeho meno a cesta, pod ktorou bude úložisko pripojené. Obidve možnosti poskytujú nepovinné prídavné nastavenia špecifické pre danú službu.

V prípade EBS nastavujeme typ zväzku, veľkosť zväzku v gigabytoch a politku vymanzania. Typ zväzku nastavuje, aký druh úložiska bude použitý na udržiavanie dát (gp3 ponúka general-purpose SSD zväzky). Politika vymazania nastavuje, čo sa má stať zo zväzkom po zmazaní HPC. V tomto prípade nastavujeme, aby bol ponechaný. Vďaka tomuto ho je možné neskôr pripojiť na nové zhluky bez potreby opätovného pripravovania jeho obsahu.

Na EFS úložisku nebudeme nastavovať žiadne ďalšie parametre. Pre všetky konfiguračné nastavenia jednotlivých druhov úložísk je možné nájsť bližšie informácie v sekcii *SharedStorage*[3], v dokumentácii pre nástroj AWS ParallelCluster

MySQL databáza pre Slurm účtovníctvo

Druhá časť konfigurácie HPC zhluku, ktorú je potrebné doplniť manuálne, sú atribúty týkajúce sa pripojenia do databázy, ktorá bude slúžiť na ukladanie dát o účtovníctve plánovača úloh Slurm.

AWS poskytuje na tieto účely šablónu do služby AWS CloudFormation, z ktorej je vytvorená MySQL databáza infraštruktúre AWS prostredníctvom služby poskytujúcej relačné databázy v cloude s názvom Amazon Aurora [4].

- Poverenia na pripojenie ku databáze, špecificky administrátorské užívateľské meno a heslo
- Veľkosť zhluku pre databázu bežiacu na Amazon Aurora
- Cieľovú VPC a podsiete, v ktorej sa databáza bude nachádzať, alebo CIDR block, z ktorého budú podsiete vytvorené

Databázu je potrebné vytvoriť v rovnakej VPC a jej podsieti, v akej bude vytvorený aj HPC zhluk.

Po vygenerovaní databázy je možné nahliadnuť prostredníctvom online rozhrania AWS Management Console [5] do CloudFormation Stacku vygenerovaného šablónou, kde sa nachádzajú jeho výstupy. Časť týchto výstupov reprezentujú údaje, ktoré je potrebné zadať do konfiguračného súboru pre nový HPC zhluk. Popis konfigurácii, ktoré je potrebné pridať, a hodnoť, ktoré v nich majú byť zadané pridávam nižšie.

```
HeadNode:
  ...
  Networking:
   AdditionalSecurityGroups:
     - DatabaseClientSecurityGroup
Scheduling:
  ...
```

[3] SharedStorage konfiguračné nastavenia https://docs.aws.amazon.com/parallelcluster/latest/ug/SharedStorage-v3.html

[4] Šablóna dostupná v rámci tutoriálu https://docs.aws.amazon.com/parallelcluster/latest/ug/tutorials_07_slurm-accounting-v3.html

[5] AWS Management Console https://aws.amazon.com/console/

```
SlurmSettings:
  Database:
    Uri: DatabaseHost
    UserName: DatabaseAdminUser
    PasswordSecretArn: DatabaseSecretArn
```

Výpis 6.8: Sekcia konfigurácie pre pripojenie na databázu pre Slurm Účtovníctvo

Vytvorenie HPC zhluku

Po pridaní vyššie uvedených nastavení do konfiguračného súboru je tento súbor pripravený na tvorbu AWS HPC zhluku. Táto akcia bude vykonaná opäť prostredníctvom AWS ParallelCluster CLI aplikácie. Použijeme na to príkaz:

```
pcluster create-cluster --cluster-name <nazov_zhluku>
                        --cluster-configuration <nazov_konf_suboru.yaml>
```

V prípade úspechu tohto príkazu bude jeho výstupom popis vo formáte JSON o niektorých vlastnostiach vytváraného HPC zhluku. Kľúčovou informáciou je jeho stav pod záznamom *"clusterStatus"*. Zhluk bude spočiatku v stave *"CREATE_IN_PROGRESS"*. Priebežne kontrolovať tento stav sa dá pomocou príkazu:

```
pcluster describe-cluster --cluster-name <nazov_zhluku>
```

Po tom, ako sa zhluk dostane do stavu *"CREATE_COMPLETE"* je zhluk vytvorený. Po jeho vytvorení je možné sa pripojiť na hlavný uzol tohto zhluku. Toto je možné buď prostredníctvom ľubovoľného SSH klienta, alebo prostredníctvom príkazu:

```
pcluster ssh --cluster-name <nazov_zhluku>
             -i <~/cesta/ku/klucu.pem>
```

Inštalácia dodatočného softvéru

Pre to, aby bol zhluk použitným v systéme k-Dispatch je potrebné nainštalovať softvér, ktorý je potrebný na vyhodnocovanie liečebných plánov. Úložiskom pre tento softvér bude zdieľaný EBS zväzok. Inštalácia softvéru na tento zväzok bude vykonaná na hlavnom uzle vzdialeným pripojením pomocou SSH. Samotný zväzok je pripojený na hlavný uzol, odkiaľ je zdieľaný ostatným výpočtovým zdrojom v HPC zhluku.

Pred začatím inštalácie dodatočného softvéru je odporúčané robiť si zápisky príkazov použitých počas tohto procesu. AWS ParallelCluster umožňuje spúšťanie bootstrap skriptov, ktoré sa môžu vykonať počas tvorby zhluku, vďaka čomu je tento proces v budúcnosti možné automatizovať. Viac o bootstrap skriptoch je popísané v sekcii 6.1.

Inštalácia dodatočného softvéru bude pozostávať z inštalácie dvoch individuálnych časti: Matlab Runtime a k-Wave binárne súbory.

Matlab Runtime je možné stiahnuť zo stránky dodávateľa [6], kde je poskytovaný aj pre operačný systém Linux, ktorý beží aj na novovytvorenom zhluku ako jeho distribúcia, Ubuntu. Jeho prevzatie a inštalácia pozostáva zo 4 krokov:

1. Stiahnutie programu z URL linku na stránke dodávateľa prostredníctvom nástroja *wget*.

[6] MATLAB Runtime https://www.mathworks.com/products/compiler/matlab-runtime.html

2. Rozbalenie programu pomocou nástroja unzip.
3. Inštalácia programu pomocou pribaleného inštalačného skriptu (používanie je popísane v tutoriáli na stránke dodávateľa[7].
4. Nastavenie cesty Matlab Runtime pre nasadenie v enviromentálnej premennej *LD_LIBRARY_PATH* (špecifické cesty popísane na stránke dodávateľa[8].

k-Wave binárne súbory mi boli poskytnuté mojím vedúcim práce prostredníctvom svojho osobného online úložiska, z ktorého je súbor dostupný na stiahnutie pomocou URL linku. Postup pri sťahovaní tohto nástroju bude teda podobný, ako pri predchádzajúcom, a bude pozostávať z týchto krokov:

1. Presunutie sa do zložky, kde majú binárne súbory byť lokalizované
2. Stiahnutie binárnych súborov z poskytnutého URL linku prostredníctvom nástroja *wget*
3. Rozbalenie programu pomocou nástroja *tar*
4. Nastavenie správnych povolení pre prístup ku binárnym súborom

Automatizácia inštalácie dodatočného softvéru

Po inštalácii dodatočného softvéru je zhluk pripravený na používanie v systéme k-Dispatch. Ak bola v konfigurácii zdieľaného úložiska zvolená možnosť *DeletionPolicy: Retain*, je toto zachované aj po zmazaní zhluku, a je možné ho znovu pripojiť na nadchádzajúcich zhlukoch bez potreby opätovnej inštalácie dodatočného softvéru.

V prípade, že zdieľané úložiská nemajú byť zachované, jednou alternatívou oproti opätovnej manuálnej inštalácii softvéru je tvorba bootstrap skriptu. AWS ParallelCluster ponúka možnosť pridávania bootstrap skriptov do konfigurácie, ktoré sa vykonávajú počas zvolených bodov tvorby zhluku. Skriptu môžu byť napísané v ľubovoľnom skriptovacom jazyku, ktorý je podporovaný operačným systémom na zhluku. Skripty sú spúšťané s právami superusera[9]. Takýto skript môže obsahovať napríklad príkazy potrebné pre vykonanie inštalácie dodatočného softvéru.

Spustenie skriptu je možné naplánovať v troch štádiách uzlu: *OnNodeStart* (po spustení uzlu), *OnNodeConfigured* (po nakonfigurovaní uzlu) a *OnNodeUpdate* (po úprave uzlu). Bližší popis všetkých konfiguračných parametrov pre dodatočné skripty je možné nájsť v dokumentácii[10].

Inštalácia dodatočného softvéru je manuálne prevedená po kompletnom vytvorení hlavného uzlu, preto skript na automatizáciu tohto procesu bude taktiež spustení po nakonfigurovaní hlavného uzlu. Skript je možné predať do konfigurácie pomocou *https://* linku, prípadne *s3://* linku, ak je skript uložený v AWS službe S3. Ak je skript posúvaný do konfigurácie zo služby S3, a tento súbor nemá nastavený verejný prístup pre čítanie, je potrebné pre daný bucket, v ktorom sa súbor nachádza nastaviť aj patričné prístupové pravidlá, ako je možné vidieť na výpise 6.9

[7] Matlab Runtime inštalačný tutorial https://www.mathworks.com/help/compiler/install-the-matlab-runtime.html

[8] Matlab Runtime pridávanie cesty do https://www.mathworks.com/help/compiler/mcr-path-settings-for-run-time-deployment.html

[9] https://www.beyondtrust.com/resources/glossary/superuser-superuser-accounts

[10] https://docs.aws.amazon.com/parallelcluster/latest/ug/HeadNode-v3.html#HeadNode-v3-CustomActions

```
HeadNode:
  ...
  Iam:
    S3Access:
      - BucketName: <my_bucket>
        KeyName: scripts_dir/*
        EnableWriteAccess: false
  CustomActions:
    OnNodeConfigured:
      Script: s3://<my_bucket>/scripts_dir/add-software-install.sh
```

Výpis 6.9: Ukázka konfiguracie bootstrap skriptu v S3 buckete

6.2 Implementácia komunikačného rozhrania pre AWS zhluk do k-Dispatch

Na zabezpečenie komunikácie medzi novovytvoreným HPC zhlukom a systémom k-Dispatch je potrebné vytvoriť implementáciu komunikačného rozhrania podľa návrhu zo sekcie 5.3.

Ako z návrhu vyplýva, prvým krokom v implementácii nového komunikačného rozhrania je vytvorenie kópie zdrojového kódu z už existujúcej implementácie iného zhluku. Táto práca bola tvorená nad implementáciou pre HPC zhluk Karolina. V tejto kópii je potrebné vytvoriť niekoľko základných zmien pre registráciu komunikačného rozhrania do systému.

Názov triedy je potrebné zmeniť z pôvodného (Karolina) na nový (napríklad AWS). Rovnako je potrebné zmeniť aj názov súboru (napríklad z *_karolina_copy* na *_aws*). A na záver v priečinku *remotemachines* sa nachádza súbor *___init___.py*, v ktorom je potrebné z novovytvoreného rozhrania premenovanú triedu importovať (príkaz napríklad *from remotemachine._aws import AWS*. Týmto sa táto trieda stane verejnou, a môže byť teda volaná vo vnútri systému k-Dispatch.

V počiatočných riadkoch zdrojového súboru sa nachádza hlavička zaobalená v komentároch, v ktorej sú zapísané informácie o tomto zdrojovom súbore, ako sú napríklad autor, dátum vytvorenia a poslednej úpravy, popis a ďalšie. Vzhľadom k zmenám oproti pôvodnému komunikačnému rozhraniu je potrebné informácie v tejto hlavičke pozmeniť.

Hlavná zmena v implementácii komunikačného rozhrania spočíva v zmene parametrov novej triedy. Práve pod nimi sú definované príkazy pripadajúce funkciám, ktoré sú prostredníctvom plánovača úloh prevádzané na HPC zhluku. V pôvodných HPC zhlukoch, ktorých zdrojový kód slúžil ako podklad pre nové komunikačné rozhranie pre AWS zhluk bol používaný plánovač úloh PBS. Na AWS zhluku je používaný plánovač Slurm. Medzi týmito dvoma plánovačmi sú rozdiely v názvoch funkcii, i keď ich samotná funkcionalita je veľmi podobná. Na ukážku je pridaná tabuľka 6.2, na ktorej sú tieto rozdiely zaznamenané:

Posledná zmena týkajúca sa výmeny plánovačov zahŕňa úpravu parametru *self._jobScriptHeader*. Tento parameter v sebe drží šablónu, pomocou ktorej sa do skriptov jednotlivých úloh pridávajú skripty (označované ako direktívy) na úpravu nastavení zadávaných skriptov. Hlavným rozdielom medzi direktívami plánovačov PBS a Slurm je, že direktívy PBS začínajú s prefixom *#PBS*, kým v Slurm začínajú s prefixom *#SBATCH*. V tabuľke 6.3 je priložená analýza direktív použitých v šablóne, s poznamenanými rozdielmi medzi plánovačmi.

Väčšina implementácii metód ostane v skopírovanom zdrojovom kóde rovnaká ako v pôvodnom, najvýraznejšie zmeny sa týkajú metódy *getJobStatus()*. Pomocou tejto metódy sa

Popis funkcie	Názov parametru	PBS príkaz	Slurm príkaz
Vloženie práce	self._jobSubmit	qsub	sbatch
Arg. pre závislosti práce	self._jobDependency	-W depend=afterok	-d afterok
Zmena práce	self._jobAlter	qalter	scontrol
Zmazanie práce	self._jobDeletion	qdel	scancel
Výpis prác vo fronte	self._jobStat	qstat -xf	squeue

Tabuľka 6.2: Rozdiely medzi parametrami pre volania funkcii plánovačov PBS a Slurm

Popis direktívy	OpenPBS direktíva	Slurm direktíva
Fronta/partícia pre prácu	#PBS -q <queue>	#SBATCH -p «queue»
Projekt pre prácu	#PBS -A <project>	#SBATCH -A <project>
Počet uzlov a jadier	#PBS -l select= <nodes>:ncpus=<cpus>	#SBATCH -N <nodes> #SBATCH -n <cpus>
Zmazanie práce	#PBS -l walltime=<walltime>	#SBATCH -t walltime
Zmazanie práce	#PBS -N <job_name>	#SBATCH -J <job_name>
Chybový súbor	#PBS -e path/to/res.err	#SBATCH -e path/to/res.err
Logový súbor	#PBS -o path/to/res.err	#SBATCH -o path/to/res.log

Tabuľka 6.3: Rozdiely medzi direktívami plánovačov PBS a Slurm

systém dopytuje HPC zhluku na stav úloh vo frontách, na základe ktorých sa riadi ďalšie spracovávanie liečebných plánov.

Narozdiel od plánovača PBS, ktorý je schopný získavať informácie aj o dokončených úlohách pomocou príkazu na výpis práci vo frontách (*qstat*), je potrebné pri plánovači Slurm tieto informácie získavať z účtovníckych dát. Kvôli tomuto je nutné mať nakonfigurované v AWS zhluku tak, ako bolo popísané v sekcii 6.1. Tieto dáta sú získané pomocou príkazu:

```
sacct -j <job_id> -X --format state,cputime,ncpus,comment
```

Pomocou tejto funkcie komunikačné rozhranie získava informácie, ktoré sú potrebné pre popis úlohy, a to jej stavu, využitého procesorového času, počtu cpu a prípadného komentáru.

Tento stav umožňuje systému detekovať zmeny v priebehu práce a reagovať na ne adekvátnou zmenou stavu v databáze. Podľa týchto zmien v databáze môže úloha prechádzať do rôznych stavov. Najdôležitejšou časťou odpovede na tento príkaz je práve stav danej úlohy. Na základe neho môže systém zaznamenať zmenu v priebehu práce, a patrične na ňu zareagovať zmenou stavu v databáze. Podľa týchto zmien môže vyhodnocovanie liečebného plánu vnútri systému prejsť do rozličných stavov, na ktoré systém postupne reaguje. Jednotlivé stavy úloh sú popísané v enum triede *JobStatus*. V tabuľke 6.4 je pridaný popis rozdielov týchto stavov na základe vrátených stavov z volaní funkcii PBS a Slurm plánovačov.

Ostatné stavy, ktoré môže príkaz *sacct* vrátiť sú dostupné v manuálových stránkach pre daný príkaz[11].

[11] https://slurm.schedmd.com/sacct.html#SECTION_JOB-STATE-CODES

Názov stavu v k-Dispatch	PBS	Slurm
QUEUED	Q, W	PENDING,REQUEUED
RUNNING	R, E	RUNNING
SUSPENDED	H, S	SUSPENDED
COMPLETED	F; exit_status == 0	COMPLETED
DELETED	F; exit_status >= 256	CANCELLED
ERROR	F; exit_status 0<x<256	ostatné stavy

Tabuľka 6.4: Rozdiely medzi stavmi úloh plánovačov PBS a Slurm

6.3 Integrácia AWS HPC zhluku do databázy

Pred začiatkom integrácie AWS HPC zhluku do databázy je potrebné získať verejnú IP adresu hlavného uzlu. Toto je možné pomocou AWS ParallelCluster CLI aplikácie s použitím príkazu:

```
pcluster describe-cluster-instances --cluster-name <nazov_zhluku>
                                    --node-type HeadNode
```

Výsledkom tohto príkazu je výpis informácii o hlavnom uzle v HPC zhluku, jedna z ktorých je aj jeho verejná IP adresa. Táto bude slúžiť ako hostname pre tento zhluk, na ktorý sa systém k-Dispatch bude pripájať prostredníctvom SSH protokolu.

Vytváranie nových záznamov prebieha pomocou použitia metódy *create()*, ktorá je volaná z implementácie triedy BaseModel pre tabuľku, do ktorej je záznam pridávaný. Ako parametre tejto metódy sú prijímané dáta, ktoré budú v zázname uložené. Táto funkcia vracia referenciu v databáze na vytvorený záznam, táto referencia sa hodí pri tvorení záznamov zo závislosťami na záznamy z iných tabuliek.

V súbore s ukážkovými dátami sa záznamy pre HPC zhluky generujú z kombinácie troch zoznamov, v ktorých na nachádzajú ich jednotlivé položky (hostname, názov zhluku a faktor). Do týchto listov sú pridávané položky patriace novému AWS HPC zhluku (napríklad "<hostname>", "AWS", 1.0). Je vhodné si referenciu na tento záznam uložiť do premennej.

Tento zhluk je potrebné priradiť do alokácie, na toto môže poslúžiť už existujúca alokácia v ukážkových dátach. K pridaniu zhluku do alokácie je potrebné vytvoriť záznam s modelom *HpcAllocation*, ktorý popisuje asociatívnu tabuľku medzi HPC zhlukmi a alokáciami. Podobne je potrebné priradiť zhluk aj ku užívateľskej skupinke do asociatívnej tabuľky reprezentovanej modelom *HpcGroup*.

Pre zhluk je ešte potrebné vytvoriť záznamy o jeho dostupných frontách. V tejto implementácii HPC zhluku je len jedna fronta s názvom *qprod*. Na tvorbu záznamov pre fronty slúži BaseModel *HpcQueue*.

Kvôli jednoduchosti riešenia je potrebné vytvoriť taktiež nový užívateľský profil s menom "ubuntu", pomocou ktorého bude potrebné prihlásenie do systému k-Dispatch v prípade práce s novým AWS HPC zhlukom. Dôvodom za tým bude absencia iných užívateľských profilov na novom zhluku. Ďalšie profily môžu byt v budúcnosti pridané pomocou integrácie služby Active Directory do daného HPC zhluku[12].

Na záver je potrebné vytvorenie záznamov spájajúcich sa s binárnymi súbormi potrebnými na vykonávanie potrebných výpočtov. Na testovanie mi bolo poskytnutých 7 takýchto súborov. Pre všetky z nich je v ukážkových dátach vytvorená referencia na záznamy v ta-

[12] https://docs.aws.amazon.com/parallelcluster/latest/ug/multi-user-v3.html

buľke *implementationdetails*. Pomocou nich sú identifikované binárne súbory, ktoré môžu byť použité na vyhodnocovanie liečebných plánov. Na základe referencii sú do databázy pridávané záznamy do tabuliek popísaných v sekcii 5.4.

6.4 Implementácia nástroju na správu AWS HPC zhlukov

K práci je nakoniec vytvorený nástroj, ktorý zaobaľuje funkcionalitu AWS ParallelCluster do spustiteľnej konzolovej aplikácie s jednoduchým užívateľským rozhraním. Ponúka základnú funkcionalitu na prácu s HPC zhlukmi vytvorenými na AWS, ako je tvorba zhlukov, zobrazovanie informácii o zhlukoch alebo mazanie zhlukov.

Nástroj taktiež ponúka vlastného sprievodcu tvorením konfiguračného súboru pre nový HPC zhluk. Tento sprievodca je založený na sprievodcovi obsiahnutom v AWS ParallelCluster CLi aplikácii, je však rozšírený o možnosť priradenia zdieľaných úložísk pre daný zhluk. V aktuálnej verzii podporuje úložiská služby EBS a EFS.

Kapitola 7

Testovanie

V tejto kapitole bude popísané testovanie AWS HPC zhluku v systéme k-Dispatch. Na testovanie mi boli poskytnuté štyri testovacie súbory, ktoré sú v systéme vyhodnotiteľné. Všetky testy budú prebiehať na súbežnom spustení všetkých štyroch testov. Špecifiká každého testu budú popísané ich sekcii.

7.1 Test rýchlosti zdieľaných úložísk

Prvým testom, ktorý bude vykonaný nad AWS HPC zhlukom je testovanie zdieľaného úložiska v použitom zhluku. V konfigurácii spomínanej v sekcii bolo povedané, že AWS HPC zhluk bude disponovať dvoma druhmi zdieľaných úložísk. Prvým je blokové úložisko pomocou služby EBS.

EBS úložisko slúži ako virtuálny pevný disk, ktorý je pripojený na hlavný uzol, a udržuje v sebe softvér potrebný na vyhodnocovanie úloh v systéme k-Dispatch. Toto úložisko je zdieľané s ostatnými uzlami v zhluku, ktoré k nemu pristupujú pomocou NFS protokolu. Takéto disky môžu trpieť obmedzenou priepustnosťou v závislosti od výkonu hlavného uzlu, na ktorý sú pripojené.

Na riešenie tohto problému je ku AWS HPC zhluku pripojené ešte jedno zdieľané úložisko, vytvorené pomocou služby EFS. Úložiska zo služby EFS sú navrhnuté na vysokoparalelizovaný prístup, pri ktorom s úložiskom pracujú mnohý klienti na raz s minimálnou dobou odozvy. Priestupnosť pri EFS zväzkoch je riešená dynamicky, použitá priepustnosť je účtovaná do celkovej finančnej sumy za prenajímanie tejto služby.

Na nasledujúcom teste budú testované dva scenáre. V prvom budú výpočtové zhluky ukladať dáta používané počas vyhodnocovania liečebných plánov na EFS úložisko, kde EBS úložisko bude slúžiť len na prístup k potrebnému softvéru na výpočty. V druhom bude EBS úložisko slúžiť na celý beh zhluku. Zvyšná konfigurácia zostáva rovnaká, ako bola spomínaná v sekcii 7.1. Počas testu bude prebiehať jeden beh nad všetkými štyrmi dostupnými liečebnými plánmi.

Na základe výsledkov zobrazených v tabuľke vyššie je jasné, že čas použitý na vypracovanie liečebných plánov pri úložisku na službe EFS je výrazne nižší ako pri službe EBS, špeciálne v prípadoch plánov s poľom elementov na spracovanie. Tieto plány budú náročnejšie na spracovanie v závislosti na veľkosť dát, ktoré obsahujú, preto tieto výsledky dávajú zmysel.

Jedným potenciálnym riešením tohto problému je zvýšenie výpočtovej kapacity na hlavnom uzle. Nasledujúci pokus bude prebiehať rovnako ako predchádzajúci, kde jediný rozdiel

Názov plánu	čas (EFS)	čas (EBS)
ARTIFICIAL-DATA-ANNULAR-ARRAY	412,37s	869,63s
ARTIFICIAL-DATA-SINGLE-ELEMENT	269,62s	365,81s
REAL-DATA-ANNULAR-ARRAY	1776,61s	3721,38a
REAL-DATA-SINGLE-ELEMENT	1024,25s	1343,36s

Tabuľka 7.1: Výsledky testu popísaného v sekcii 7.1, hlavný uzol inštančného typu t2.micro.

bude v inštančnom type hlavného uzlu. Tento krát bude jeho inštančný typ rovnako ako aj na výpočtových uzloch t2.xlarge, čo mu navýši počet jadier z jedného na štyri, a veľkosť pamäte z 1GB na 16GB.

Názov plánu	čas (EFS)	čas (EBS)
ARTIFICIAL-DATA-ANNULAR-ARRAY	388,7s	386,03s
ARTIFICIAL-DATA-SINGLE-ELEMENT	259,1s	xxxxx
REAL-DATA-ANNULAR-ARRAY	1720,51s	1787,34s
REAL-DATA-SINGLE-ELEMENT	942,08s	993,02s

Tabuľka 7.2: Výsledky testu popísaného v sekcii 7.1, hlavný uzol inštančného typu t2.xlarge.

Z dát zobrazených v tabuľke 7.2 je zrejmé, že pri navýšení výpočtovej kapacity hlavného uzlu sa zvyšuje aj priepustnosť EBS úložisku zdieľanom s ostatnými uzlami v zhluku. Vzhľadom na vyššiu cenu EFS úložiska oproti EBS toto môže byť dobrým spôsobom, ako ušetriť na nákladoch za prevádzku (0.192$ za EFS v jednej zóne dostupnosti proti 0.095$ za EBS). Je však dôležité poznamenať, že pokiaľ pri EBS úložisku sa platí čiastka za ustanovenú veľkosť zväzku, EFS sa cenovo dynamicky prispôsobuje aktuálnemu využívaniu, a tak jeho cena v dobách nízkej záťaže môže byť výhodnejšia.

7.2 Porovnanie výkonu inštancii Intel a AMD

V ďalší test sa bude zameriavať na porovnanie rýchlostí inštancii bežiacich na procesoroch Intel v porovnaní s inštanciami, ktoré používajú procesory AMD.

	c6i.4xlarge	c6a.4xlarge
Názov CPU	Intel Xeon 8375C	AMD EPYC 7R13
Počet vCPU	16	16
Pamäť	32 GB	32 GB
Max. rýchlosť CPU	neznáme	3.6 GHz
Cena za hodinu	$0.68	$0.612

Tabuľka 7.3: Popis inštančného typu c6i.4xlarge(Intel) a c6a.4xlarge(AMD).

Prvým krokom je zvoliť si správne inštančné typy pre oba procesory. Tieto inštancie budú potrebovať prioritne vysoký výkon, preto budú vyberané s kategórie *compute optimized*. Po konzultácii s mojím vedúcim mi bolo zdieľané, že najefektívnejší počet jadier v procesore na výpočty v systéme k-Dispatch sa pohybuje okolo 16 jadier.

Na základe týchto faktov je možné zvoliť pre výpočtové uzly inštančné typy *c6i.4xlarge* pre Intel a *c6a.4xlarge*. Bližšie porovnanie týchto dvoch inštancii pridávam v tabuľke 7.3.

Testy budú prebiehať v štyroch rôznych behoch. Na základe nich budú v tabuľkách nižšie popísané relevantné informácie o časoch na vyhodnotenie.

Úloha	priemer. čas Intel	priemer. čas AMD
PREPROCESSOR_0_0	15,58s	16,52s
SIMULATION_1_0	104,91s	126,98s
SIMULATION_1_1	108,89s	139,90s
POSTPROCESSOR_2_0	157,95s	133,44s
THERMAL_SIM_3_0	5,02s	5,05s
Súčet	392,36s	421,89s

Tabuľka 7.4: Tabuľka pre plán ARTIFICIAL-DATA-ANNULAR-ARRAY

Úloha	priemer. čas Intel	priemer. čas AMD
PREPROCESSOR_0_0	9,84s	9,92s
SIMULATION_1_0	110,50s	124,80s
POSTPROCESSOR_2_0	89,62s	68,69s
THERMAL_SIM_3_0	3,56s	3,60s
Súčet	213,52s	207,00s

Tabuľka 7.5: Tabuľka pre plán ARTIFICIAL-DATA-SINGLE-ELEMENT

Úloha	priemer. čas Intel	priemer. čas AMD
PREPROCESSOR_0_0	26,48s	29,37s
SIMULATION_1_0	279,39s	288,35s
SIMULATION_1_1	274,37s	308,63s
POSTPROCESSOR_2_0	797,24s	664,44s
THERMAL_SIM_3_0	37,62s	37,57s
Súčet	1415,10s	1328,36s

Tabuľka 7.6: Tabuľka pre plán REAL-DATA-ANNULAR-ARRAY

Úloha	priemer. čas Intel	priemer. čas AMD
PREPROCESSOR_0_0	17,66s	20,10s
SIMULATION_1_0	246,91s	286,10s
POSTPROCESSOR_2_0	419,48s	354,55
THERMAL_SIM_3_0	19,78s	20,50
Súčet	703,83s	681,25

Tabuľka 7.7: Tabuľka pre plán REAL-DATA-SINGLE-ELEMENT

Súčet priemerných časov na vyhodnotenie plánov hovorí v prospech Intelu v pomere tri plány k jednému, avšak vo väčšine prípadov ide o zanedbateľné časové rozdiely. Zaujímavejšie údaje sa dajú odpozorovať pri bližšom skúmaní výsledkov pre úlohy, z ktorých plány pozostávajú.

Časové rozdiely medzi úlohami typu *PREPROCESSOR* a *THERMAL_SIM* sú zanedbateľné. V každej jednej úlohe typu *SIMULATION* však Intel procesory berú značné vedenie, v percentuálnej výhode od 3% do 23%. AMD procesory si ale poradili lepšie v každej jednej úlohe typu *POSTPROCESSOR*, v percentách od 15% do 23%.

Vzhľadom k tomuto môžeme odporučiť vykonať testy na širšej vzorke plánov, a na základe výsledkov vytvorených z tejto vzorky pozrel na možnosť nasadenia dvoch výpočtových zdrojov do jednej fronty, kde jeden bude bežať na procesoroch AMD, a jeden na procesoroch Intel. Týmto by bolo možné využiť plný potenciál EC2 inštancii.

Kapitola 8

Záver

Výsledkom tejto bakalárskej práce bolo preskúmať možnosti využitia výpočtových prostriedkov služby Elastic Compute Cloud (EC2) na cloudovej platforme Amazon Web Services (AWS), následne navrhnúť implementáciu tejto služby do systému k-Dispatch, túto implementáciu realizovať, a nakoniec zhodnotiť výsledky dosiahnuté v rámci tohto vypracovania.

Návrh implementácie pozostával z troch častí. V prvej časti sa nachádzal technický pohľad na funkcionalitu systému k-Dispatch z pohľadu práce s jeho aktuálne používanými HPC zhlukmi. Na základe týchto pozorovaní bolo zhodnotené, ako sa má postupovať vrámci implementácie AWS HPC zhluku pri jeho nasadzovaní.

Ďalej bola analyzovaná reprezentácia HPC zhlukov v systéme k-Dispatch. Na základe už existujúcich riešení bol vytvorený návrh nového komunikačného rozhrania, pomocou ktorého bude umožnená komunikácia medzi systémom a AWS HPC zhlukom.

Na záver návrhu bola analýza databázy systému k-Dispatch. V nej bolo rozhodnuté, ktoré tabuľky obsahujú údaje, ktoré slúžia k registrácii ďalších HPC zhlukov do systému k-Dispatch, a ktoré všetky ďalšie položky musia byť zadané na zabezpečeniu vypracovania liečebných plánov, ktoré sú do systému zadávané.

Implementácia zhluku sa odvíjala od práve spomínaného návrhu. Prvým krokom bolo vytvorenie AWS HPC zhluku, ktorý bol integrovaný do systému k-Dispatch. Toto bolo dosiahnuté vďaka pomoci nástroju AWS ParallelCluster, pomocou ktorého je možné tvoriť konfigurácie vo súborovom formáte YAML, pomocou ktorého je nástroj schopný vygenerovať všetky požadované zdroje pre daný HPC zhluk.

Po úspešnom vytvorení AWS HPC zhluku, a nainštalovaní všetkého potrebného softvéru pre prácu v systéme k-Dispatch bola pre tento zhluk vytvorená implementácia komunikačného rozhrania pre systém k-Dispatch, a taktiež bol AWS HPC zhluk registrovaný v databáze.-

Na záver implementácie bol vytvorený nástroj, ktorý zaobalil funkcionalitu systému AWS ParallelCluster do konzolovej aplikácie, vytvorenej v programovacom jazyku Python, ktorá bola obohatená o funkcionalitu potrebnú pre správu HPC zhlukov na platforme AWS.

Práca je zakončená testovaním nového AWS HPC zhluku v systéme k-Dispatch. Testované boli menovite jeho možnosti zdieľaného úložiska, ich jednotlivé výhody a nevýhody, a nakoniec testovanie generačne rovnakých procesorov od AMD a Intel.

Zadanie tejto bakalárskej práce bolo vypracované kompletne. Je tu však rozhodne priestor na ďalšie rozšírenia. V aktuálnom prevedení je schopný nový AWS HPC zhluk pracovať len s predvoleným účtom s menom "ubuntu", čo výrazne obmedzuje jeho použiteľnosť pre verejnosť. Toto je riešiteľné implementáciou Active Directory do ďalších verzii AWS HPC

zhlukov. Taktiež na základe testov medzi procesormi AMD a Intel je možné skonštatovať, že najefektívnejším riešením by bolo spojenie obidvoch procesorov do pracovného postupu jedného plánu, čo je v rámci AWS HPC zhlukov možné.

Vypracovanie tejto práce mi prinieslo do života skúsenosti s platformou AWS, a mnohými službami, ktoré poskytuje. Tieto skúsenosti sú v dnešnej dobe na nezaplatenie, keďže značná časť dnešných softvérových produktov v nejakej kapacite využíva práve buď túto platformu, alebo nejakú inú na zúžitkovanie výhod, ktoré dokážu cloudové platformy poskytnúť koncovému zákazníkovi.

Literatúra

[1] *Amazon EC2 Mac instances*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-mac-instances.html.

[2] *Amazon Elastic Block Store (Amazon EBS)*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html.

[3] *AWS Free Tier*. AWS. Dostupné z: https://aws.amazon.com/free/.

[4] *AWS Global Infrastructure*. Dostupné z: https://aws.amazon.com/about-aws/global-infrastructure/?pg=WIAWS.

[5] *AWS ParallelCluster Python library API*. AWS. Dostupné z: https://docs.aws.amazon.com/parallelcluster/latest/ug/pc-py-library-v3.html.

[6] *AWS ParallelCluster UI*. Dostupné z: https://docs.aws.amazon.com/parallelcluster/latest/ug/pcui-using-v3.html.

[7] *Cloud computing with AWS*. Dostupné z: https://aws.amazon.com/what-is-aws/?nc1=f_cc.

[8] *Conda*. Dostupné z: https://docs.conda.io/en/latest/.

[9] *EC2 Security*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security.html.

[10] *EC2 Security groups*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html.

[11] *Instance types*. Dostupné z: https://aws.amazon.com/ec2/instance-types/.

[12] *Instances and AMIs*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html.

[13] *Shared Responsibility Model*. Dostupné z: https://aws.amazon.com/compliance/shared-responsibility-model/.

[14] *What is Amazon EC2? - Amazon Elastic Compute Cloud*. Dostupné z: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[15] *What is Amazon S3? - Amazon Simple Storage Service*. Dostupné z: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[16] *What is Amazon VPC?* Dostupné z: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

[17] *What is AWS Graviton?* Dostupné z: https://docs.aws.amazon.com/whitepapers/latest/aws-graviton-performance-testing/what-is-aws-graviton.html.

[18] *What is AWS ParallelCluster*. Dostupné z: https://docs.aws.amazon.com/parallelcluster/latest/ug/what-is-aws-parallelcluster.html.

[19] *What is Docker?* Dostupné z: https://www.ibm.com/topics/docker.

[20] *What's the difference between Anaconda, conda, and Miniconda?* Dostupné z: https://bioconda.github.io/contributor/faqs.html#conda-anaconda-minconda.

[21] *Introduction to HPC: What are HPC HPC Clusters?* Weka, Jul 2020. Dostupné z: https://www.weka.io/learn/hpc/what-are-hpc-and-hpc-clusters/.

[22] Abba, I. V. *What is an ORM – The Meaning of Object Relational Mapping Database Tools*. www.freecodecamp.org, Oct 2022. Dostupné z: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/.

[23] Bell, G. Supercomputers: The amazing race (a history of supercomputing, 1960-2020). *TechReport MSR-TR-2015-2. Ver. 1. 555 California, 94104 San Francisco, CA*. 2014.

[24] Bouffler, B. a Robinson, R. *New: Introducing AWS ParallelCluster 3*. Sep 2021. Dostupné z: https://aws.amazon.com/blogs/hpc/introducing-aws-parallelcluster-3/.

[25] Dongarra, J., Sterling, T., Simon, H. a Strohmaier, E. High-performance computing: clusters, constellations, MPPs, and future directions. *Computing in Science Engineering*. 2005, zv. 7, č. 2, s. 51–59. DOI: 10.1109/MCSE.2005.34.

[26] Forcier, J. *Getting started¶*. Dostupné z: https://docs.fabfile.org/en/stable/getting-started.html.

[27] Frye, M.-K. *What is an API?* Dostupné z: https://www.mulesoft.com/resources/api/what-is-an-api.

[28] Gogan, D. *Virtual Private Cloud (VPC)*. Dostupné z: https://n2ws.com/cloud-glossary/virtual-private-cloud.

[29] Jaros, M., Treeby, B. E., Georgiou, P. a Jaros, J. K-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. New York, NY, USA: Association for Computing Machinery, 2020. PASC '20. DOI: 10.1145/3394277.3401854. ISBN 9781450379939. Dostupné z: https://doi.org/10.1145/3394277.3401854.

[30] Kristinsson, J. T. *Introduction to HPC: What are HPC HPC Clusters?* Ubuntu Blog, Jul 2022. Dostupné z: https://ubuntu.com/blog/hpc-cluster-architecture-part-4.

[31] Regalado, A. *Who coined 'cloud computing'?* MIT Technology Review, Feb 2020. Dostupné z: https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/.

[32] Watts, S. a Raza, M. *SaaS vs PaaS vs IaaS: What's The Difference amp; How to choose*. Jun 2019. Dostupné z: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/.

[33] Ylonen, T. SSH - Secure Login Connections Over the Internet. In: *6th USENIX Security Symposium (USENIX Security 96)*. San Jose, CA: USENIX Association, Júl 1996. Dostupné z: https://www.usenix.org/conference/6th-usenix-security-symposium/ssh-secure-login-connections-over-internet.