

HEAppE Middleware rozšíření pro lokální výpočty a uživatelsky definované šablony výpočtů

HEAppE Middleware Extension for Local Computing and User Defined
Command Templates

Jakub Konvička

Bakalářská práce

Vedoucí práce: Ing. Václav Svatoň, Ph.D.

Ostrava, 2022

Zadání bakalářské práce

Student:

Jakub Konvička

Studijní program:

B0613A140014 Informatika

Téma:

HEAppE Middleware rozšíření pro lokální výpočty a uživatelsky definované šablony výpočtu

HEAppE Middleware Extension for Local Computing and User Defined Command Templates

Jazyk vypracování:

čeština

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit rozšíření aplikačního rámce HEAppE Middleware (High-End Application Execution Middleware) se zaměřením na lokální spouštění výpočetních úloh a uživatelsky definované šablony pro tyto úlohy. HEAppE Middleware je open-source implementace konceptu HPC-as-a-Service. Tento aplikační framework je postaven na technologii .NET Core a poskytuje uživatelům vzdálený přístup k výpočetním clusterům se zaměřením na bezpečnost a jednoduchost přístupu. Cílem práce je tedy vytvořit rozšíření tohoto aplikačního rámce tak, aby bylo možno simulovat spouštění HPC úloh na lokálním počítači, tedy bez nutnosti napojení na skutečný výpočetní cluster a také umožnit uživatelům jednoduše si vytvořit vlastní tzv. šablony pro tyto HPC úlohy, které si tak budou moci lokálně vyzkoušet nejprve na svém počítači a až následně použít pro spuštění reálné HPC úlohy na skutečném výpočetním clusteru.

Body zadání:

1. Prostudování State of the Art v oblasti vzdáleného přístupu k HPC (HPC-as-a-Service)
2. Popis problematiky, aktuálního řešení a motivace pro rozšíření funkcionality HEAppE
3. Rozšíření funkcionality HEAppE umožňující spouštění HPC úloh na lokálním počítači
4. Rozšíření funkcionality HEAppE umožňující správu uživatelsky definovaných šablon výpočetních úloh
5. Otestování nově vytvořených rozšíření
6. Aktualizace instalovačního plánu a plánu nasazení dle nově vytvořených rozšíření.
7. Vytvořená rozšíření budou dostupná na oficiálním veřejném GIT repositáři projektu HEAppE

Seznam doporučené odborné literatury:

- [1] Oficiální stránky a repositář projektu HEAppE Middleware <http://heappe.eu>
- [2] C# 9 and .NET 5 – Modern Cross-Platform Development. Mark J. Price; 5th edition. Packt Publishing, 2020. ASIN: B08KQK22LJ
- [3] .NET Core in Action, Second Edition. Andrew Lock. Manning Publications, 2021. ISBN: 9781617298301
- [4] Advanced Software Testing - Vol. 3, 2nd Edition. Jamie L. Mitchell, Rex Black. Rocky Nook, 2015. ISBN: 9781457189104
- [5] API Testing and Development with Postman. Dave Westerveld. Packt Publishing, 2021. ISBN: 9781800569201

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Václav Svatoň, Ph.D.**

Datum zadání: 01.09.2021

Datum odevzdání: 30.04.2022

doc. Ing. Petr Gajdoš, Ph.D.
vedoucí katedry

prof. Ing. Jan Platoš, Ph.D.
děkan fakulty

Abstrakt

Cílem této práce je návrh a následná implementace rozšíření Open-Source aplikačního frameworku HEAppE Middleware o podporu lokálního spouštění výpočetních úloh a rozšíření funkcionality middleware o správu uživatelsky definovaných šablon výpočtů, které se využívají při specifikaci HPC úloh. HEAppE Middleware je systém implementující koncept HPC-as-a-Service s důrazem na bezpečnostní nároky a politiky superpočítáčových center. Uživatelům poskytuje rozhraní ke vzdálené správě HPC úloh a přístupu k jejich datům na superpočítáčových clusterech.

Klíčová slova

High-performance computing, HPC-as-a-Service, HEAppE Middleware, výpočetní cluster

Abstract

The purpose of this thesis is to design and then implement an extension of the Open-Source application framework HEAppE Middleware by supporting local execution of computational tasks and extending the functionality of the middleware by managing user-defined computation templates that are used in the specification of HPC jobs. HEAppE Middleware is a system implementing the concept of HPC-as-a-Service with an emphasis on security requirements and supercomputing center policies. It provides an interface for users to remotely manage HPC jobs and access their data on supercomputer clusters.

Keywords

High-performance computing, HPC-as-a-Service, HEAppE Middleware, computing cluster

Poděkování

Rád bych na tomto místě poděkoval vedoucímu bakalářské práce Ing. Václavu Svatoňovi, Ph.D. a Ing. Janu Křenkovi za rady a jejich čas, který mi věnovali při řešení dané problematiky.

Obsah

Seznam použitých symbolů a zkratek	8
Seznam obrázků	9
Seznam tabulek	10
1 Úvod	11
2 Superpočítac	12
2.1 Historie	12
2.2 Výkon superpočítacé	13
2.3 Plánovač úloh	14
3 State of the Art v oblasti HPC-as-a-Service	17
3.1 Výhody HPC-as-a-Service	17
3.2 Případy užití	18
3.3 Největší poskytovatelé platforem HaaS	19
4 Seznámení s HEAppE Middleware	22
4.1 Účel HEAppE Middleware	22
4.2 Technické zázemí HEAppE Middleware	23
4.3 Abstrakce HPC úlohy	25
4.4 Uživatelské a servisní účty	26
4.5 Šablony výpočtů	26
4.6 Zajímavé funkcionality HEAppE Middleware	27
5 Rozšíření HEAppE pro lokální výpočty	30
5.1 Návrh řešení	30
5.2 Konfigurace virtuálního stroje	31
5.3 Komunikace mezi HEAppE a virtuálním HPC	33
5.4 Data a stav úloh	33

5.5	Implementace na straně HEAppE Middleware	34
5.6	Implementace na straně virtuálního stroje	34
5.7	Nutné kroky ke spuštění	37
6	Uživatelsky definované šablony výpočtů	38
6.1	Generická šablona	38
6.2	Návrh řešení	39
6.3	Implementace	39
6.4	Prerekvizity pro správné fungování	40
7	Testování softwaru	41
7.1	Historie testování	41
7.2	Proces vývoje softwaru	41
7.3	Testování API	45
7.4	Manuální a automatizované testování	45
7.5	Framework pro automatizované testování HEAppE	46
7.6	Testovací scénář pro HEAppE	47
8	Závěr	52
Literatura		53
Přílohy		55
A	Příklad jednotlivých segmentů specifikace pro lokální HPC	56
B	Spuštěný kontejner HEAppE	58
C	Příklad specifikace testovacího plánu pro Robot Framework	59
D	Robot Framework - výstup testu	60
E	Výstup testování přenosu souborů	62

Seznam použitých zkrátek a symbolů

API	– Application Programming Interface
DOS/DDOS	– (Distributed) Denial of Service
DTO	– Data Transfer Object
Flop/s	– Floating-point Operations Per Second
HEAppE	– High-End Application Execution (Middleware)
HPC	– High-performance computing / High-performance cluster
HTTP	– Hypertext Transfer Protocol
HaaS	– HPC-as-a-Service
JSON	– JavaScript Object Notation
PID	– Process ID
REST API	– Representational State Transfer Application Programming Interface
RPC	– Remote Procedure Call
RSA	– Rivest–Shamir–Adleman cryptosystem
SCP	– Secure Copy Protocol
SSH	– Secure Shell
SW	– Software

Seznam obrázků

2.1	Seymour Cray [2]	12
2.2	Garry Kasparov hraje proti Deep Blue [3]	13
2.3	Plánování úloh [7]	14
2.4	Algoritmus FCFS bez použití techniky Backfilling [9]	16
2.5	Algoritmus FCFS s použitím techniky Backfilling [9]	16
3.1	HPC-as-a-Service / HaaS [12]	18
3.2	Snímek aplikace HPE Greenlake Central [14]	19
3.3	Snímek aplikace The Smart HPC Web Portal [15]	20
3.4	Snímek aplikace Atos Extreme Factory [15]	21
4.1	Vybrané HEAppE Middleware REST API endpointy	24
4.2	Architektura Docker [18]	25
4.3	Cirkulární závislost [21]	28
4.4	Rozdělení extrémně dlouhé úlohy na podúlohy	29
5.1	Návrh rozšíření o lokální HPC cluster	31
7.1	Záznam o první počítačové chybě [25]	42
7.2	Model velkého třesku	43
7.3	Model „programuj a opravuj“	43
7.4	Vodopádový model	44
7.5	Spirálový model [27]	44
7.6	Průběžná integrace [31]	47
B.1	Ukázka aplikace Docker Desktop se spuštěným kontejnerem	58
D.1	Report Robot Frameworku	60
D.2	Log Robot Frameworku	61

Seznam tabulek

7.1	Seznam kroků pro testování funkcionalit souvisejících s uživatelem HEAppE	48
7.2	Seznam kroků pro testování úloh HEAppE	49
7.3	Seznam kroků pro testování Management sekce HEAppE	49

Kapitola 1

Úvod

Oblast vysoce výkonného počítání (HPC) se neustále rozrůstá. Navyšují se kapacity, prostředky, a s tím narůstají i možnosti tyto služby využít na stále sofistikovanější problémy. I přes tento vývoj je výpočetní kapacity trvalý nedostatek, uživatelé jsou tak omezováni např. plánovačem nebo počtem jádrohodin, které mohou v rámci svého projektu využít.

Vývojové týmy v superpočítáčových centrech se snaží maximálně zjednodušit přístup ke koncovým výpočetním uzelům, aniž by došlo ke snížení výkonnosti. V IT4Innovations se jeden z výzkumných týmů zabývá vývojem platformy, která uživateli umožňuje k superpočítáči přistupovat jako ke službě (HPC-as-a-Service). Tato platforma nese označení HEAppE Middleware (High-End Application Execution Middleware) a mezi její stěžejní funkce patří vytváření, spouštění a získávání stavu úlohy ze superpočítáčového clusteru. Smyslem tohoto projektu je vytvoření rozhraní za účelem snadného přístupu k HPC a „odstínění“ standardních uživatelů od specifických funkcionalit. Rozhraní HEAppE je pak dále integrováno do dalších aplikací.

Cílem této práce je již zmíněný middleware (HEAppE Middleware) rozšířit o možnost spouštění testovací úlohy lokálně na počítači uživatele bez nutnosti napojení na fyzický výpočetní cluster a umožnit uživateli spravovat jeho vlastní šablony pro spouštění úloh. Praktickým využitím bude pak simulace HPC úlohy přímo na počítači koncového uživatele. Až bude s návrhem své šablony spokojen a vše si lokálně otestuje, bude si moci jednoduše danou úlohu spustit na skutečném výpočetním clusteru.

Součástí práce je i vytvoření testovacích plánů, do kterých bude zahrnuto i testování výše uvedených rozšíření HEAppE Middleware.

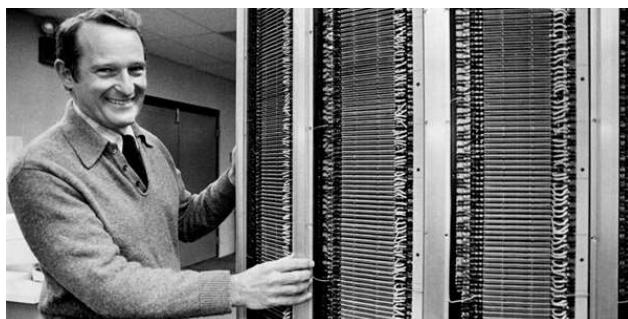
Kapitola 2

Superpočítač

Výpočetní cluster, také nazýván jako superpočítač, slouží především k řešení výpočetně náročných úloh. Mezi typické úlohy mohou být zařazeny matematické, fyzikální či chemické výpočty nad velkým objemem dat. Největší výhodou superpočítačového clusteru je pak možnost paralelizace úloh, v jednom okamžiku je zpracováváno více vstupů. Paralelní přístup se využívá například při práci s grafikou, při renderování scén nebo u různých simulací.

2.1 Historie

První superpočítače se začaly objevovat kolem roku 1960. Za pomyslného otce zakladatele a prvního vynálezce konceptu superpočítače je považován Seymour Cray. Cray hrál klíčovou roli při vývoji prvního superpočítače UNIVAC 1103. Jednalo se o první počítač, který byl určen ke komerčním účelům. Později si Cray založil společnost Cray Research a se svým týmem pracoval na vývoji několika modelů superpočítačů. První superpočítač této společnosti s názvem Cray-1 byl schopen provést 240 miliónů výpočtů za sekundu, pozdější model Cray-2 se pyšnil schopností provádět přes 1 miliardu výpočtů za sekundu. S těmito počítači mohly pracovat i vědecké instituce. [1]



Obrázek 2.1: Seymour Cray [2]

Společnost Cray Research využívala ke konstrukci jejich superpočítáčů až $10\times$ rychlejší procesory v porovnání s procesory využívanými pro komerční počítače této doby. Později, počátkem 90. let 20. století se však do popředí dostávají společnosti IBM a HP, které výkonem jejich superpočítáčů zastiňují všechny menší firmy pokoušející se o vývoj vlastních superpočítáčů.

V minulosti byly superpočítáče navrhovány pro konkrétní úlohu. Příkladem takto specializovaného výpočetního clusteru je superpočítáč Deep Blue, který byl vyvinut společností IBM. Jeho úkolem bylo počítat postavení šachových figur na hracím poli. Tento superpočítáč měl k dispozici 64 procesorů, na každou buňku hracího pole jeden. Deep Blue byl schopen spočítat až 200 milionů postavení šachových figur za sekundu. V květnu roku 1997 tento superpočítáč porazil ruského šachového velmistra Garryho Kasparova [3].



Obrázek 2.2: Garry Kasparov hraje proti Deep Blue [3]

2.2 Výkon superpočítáče

Celosvětově využívanou metrikou pro měření výkonu superpočítáčů je jednotka Flop/s. Jedná se o metriku, která je založena na počtu vykonaných operací v pohyblivé řádové čárce za sekundu. V době psaní této práce nejvýkonnější superpočítáče světa dosahují výkonu přes 440 PFlop/s.

Dle světového žebříčku TOP500 je nejvýkonnějším superpočítáčem v této době „Supercomputer Fugaku“ s ověřeným praktickým výkonem 442010 TFlop/s. Teoretický výkon tohoto clusteru je pak přes 537 PFlop/s, je počítán z udávaných hodnot frekvence použitých procesorů. Tento superpočítáč má k dispozici přes 7,5 milionu jader a vlastní jej největší japonská vědecká instituce RIKEN [4].

Nejvýkonnějším superpočítáčem v České republice je nyní cluster Karolina s celkovým teoretickým výkonem 3,8 PFlop/s [5]. Tento výpočetní cluster je umístěn v národním superpočítáčovém centru IT4Innovations při Vysoké škole báňské – Technické univerzitě Ostrava. Superpočítáč Karolina je v době psaní této práce umístěn na 71. pozici ve světovém žebříčku TOP500 [6].

Vývoj v odvětví IT se však žene stále kupředu a je jen otázkou času, kdy se v žebříčku objeví nový superpočítáč a opětovně celým pořadím zamíchá.

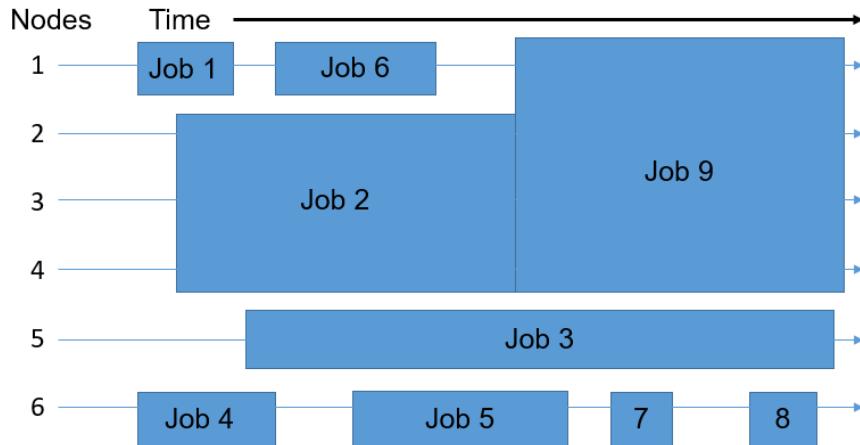
2.3 Plánovač úloh

Výpočetní cluster je z praktického hlediska implementací dávkového systému. Uživatelé sami jejich programy na systému nespouští, ale dávají pokyn plánovači, který zajistí zpracování úlohy [7]. Jedná se o program, který spravuje životní cyklus úloh na clusteru. Dle vypočtené priority řadí vytvářené úlohy do fronty, ze které jsou následně úlohy spouštěny. Stěžejními atributy pro zařazení úlohy do fronty mohou být například žádosti o množství zdrojů, doba zpracování nebo priorita úlohy.

Aby nedocházelo k plýtvání nevyužívaných zdrojů, je také jedním z hlavních faktorů zařazení úlohy do fronty využití samotných uzlů na výpočetním clusteru. Pokud tedy plánovač volí pořadí spouštění jednotlivých úloh, mezi metriky zařazuje i vlastní volné zdroje na kterých mohou být spouštěny kratší úlohy, i když pokyn k jejich zpracování přišel později než u časově náročnějších úloh.

I přes vysoký výkon superpočítáčů a jejich neustálý vývoj není aktuálně možné uspokojit veškeré uživatelské požadavky ke zpracování úloh. Vytížení superpočítáče a jeho zdrojů se průměrně pohybuje mezi 80-90 %, každou dokončenou úlohu tak většinou nahrazuje úloha další. Uživatel tedy zpravidla na spuštění úlohy čeká. I tento problém pomáhá řešit plánovač na superpočítáčovém clusteru.

Následující obrázek 2.3 ilustruje plánování úloh s ohledem na co možná nejfektivnější využití výpočetních zdrojů. V tomto případě se jedná o dávkový systém se 6 výpočetními uzly. Cílem plánovače je zvolit pořadí spouštění úloh, tak, aby bylo minimalizováno plýtvání zdroji. Jednou z podmínek plánovače je omezení naplánování nanejvýš jedné úlohy na jeden uzel v jednom čase. V tomto příkladu se plánuje 9 úloh.



Obrázek 2.3: Plánování úloh [7]

Mezi hlavní úkoly každého plánovače se řadí zajištění spuštění každé úlohy (žádná úloha nesmí čekat ve frontě nekonečně dlouhou dobu), minimalizace nevyužití zdrojů a maximalizace průchodusnosti úloh systémem.

2.3.1 Plánovací algoritmy

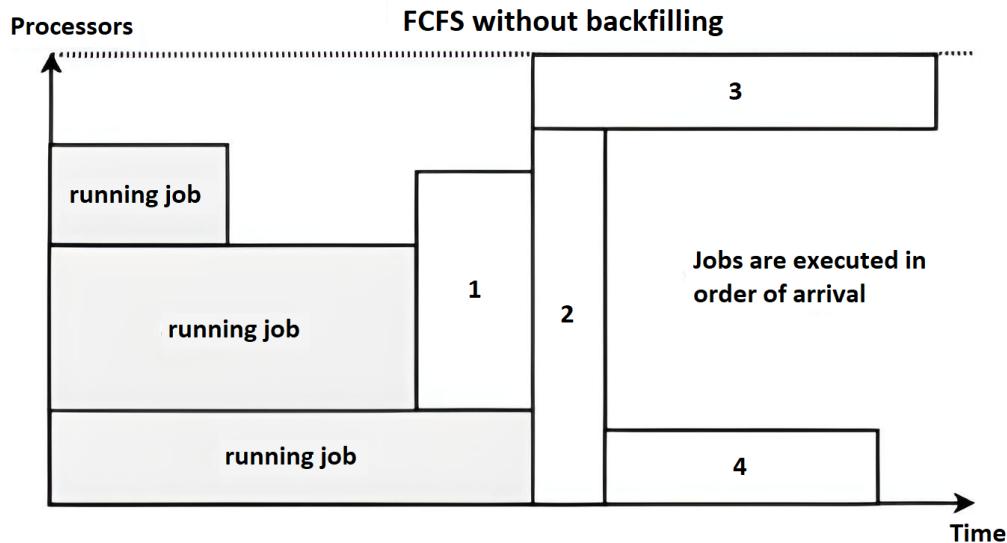
Jak už bylo popsáno výše, plánovač je odpovědný za řazení úloh do fronty. Další úlohou plánovače je přidělování zdrojů jednotlivým úlohám v době jejich běhu dle zásad a dostupnosti zdrojů.

Mezi nejčastěji využívané plánovací algoritmy se řadí algoritmus zvaný FCFS (first-come first-served). Při této variantě použitého algoritmu plánovač kontroluje, zda je možné spustit první úlohu ve frontě. Tato fronta je pak v praxi označována jako fronta typu FIFO (first-in-first-out).

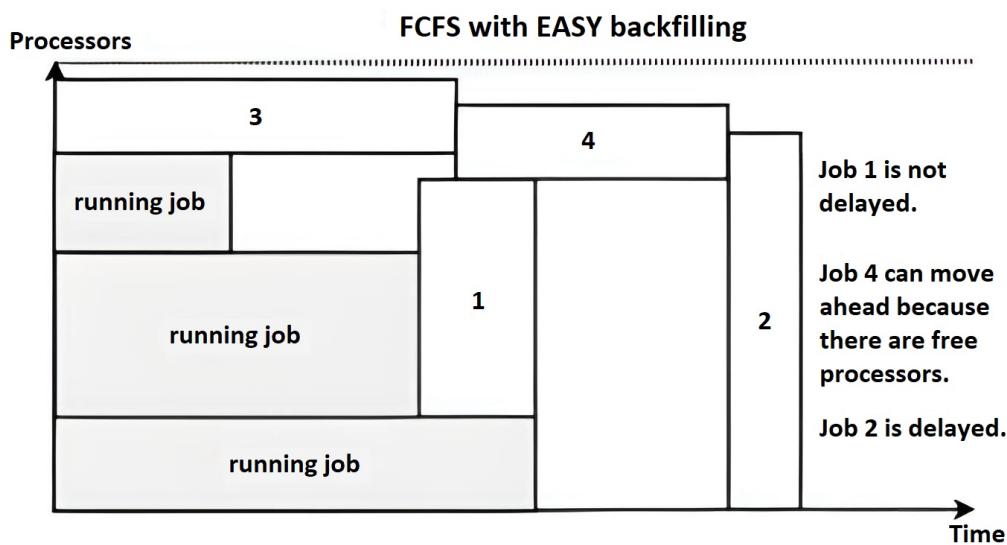
K plánování HPC úloh může být také využit algoritmus ALCF (Argonne Leadership Computing Facility), princip tohoto algoritmu je obdobný jako u již zmiňovaného FCFS s rozdílem prioritizace časově náročnějších úloh a úloh, které jsou ve frontě již dlouhou dobu.

Nejčastěji je však při plánování úloh využíváno techniky Backfilling. Technika Backfilling je optimalizovanou variantou algoritmu či principu FCFS, plánovač zkонтroluje, zda je možné spustit první úlohu ve frontě. Pokud je to možné (je dostatek prostředků), úloha se spustí bez dalšího čekání. Pokud to však možné není, plánovač projde zbytek fronty a provádí kontrolu, zda je možné spustit další úlohu, aniž by se prodloužila čekací doba první úlohy ve frontě [8].

Na obrázku 2.4 je ilustrována průchodnost systémem s využitím plánovacího algoritmu FCFS bez využití techniky Backfilling. Ilustrace 2.5 znázorňuje taktéž algoritmus FCFS s využitím techniky Backfilling. Při porovnání těchto ilustračních obrázků je možné vidět značné omezení plýtvání zdroji a maximalizaci průchodnosti systémem při použití techniky Backfilling. Bloky s čísly znázorňují plánované úlohy z fronty. Již běžící úlohy jsou označeny jako „running job“.



Obrázek 2.4: Algoritmus FCFS bez použití techniky Backfilling [9]



Obrázek 2.5: Algoritmus FCFS s použitím techniky Backfilling [9]

Kapitola 3

State of the Art v oblasti HPC-as-a-Service

Superpočítače a jejich výkon byl v minulosti vyhrazen pouze pro oblast výzkumu a vývoje. HPC infrastruktura byla k dispozici vládním, lékařským a výzkumným organizacím nebo inovativním filmařům. K majoritním případům užití superpočítačů v této době patřily simulace jaderných testů, mapování lidského genomu nebo „oživování dinosaurů“ ve filmech.

V současné době však dochází k masivnímu vzestupu datově náročných technologií. Typickým příkladem může být problematika umělé inteligence nebo problémy vyžadující masivní paralelismus¹. Tento fakt nutí širší okruh organizací zkoumat problematiku vysoce výkonného počítání. Ne každá společnost však má čas, peníze a odborné znalosti potřebné k jejich nasazení.

Z toho důvodu mohou tyto společnosti namísto náročného sestavování a správ superpočítačových clusterů využít již hotová řešení HPC jako služby (HPC-as-a-Service/HaaS). Díky HPC poskytovanému jako služba mohou organizace v některých případech spolupracovat s dodavatelem na konfiguraci infrastruktury, kterou potřebují pro své konkrétní výpočetní potřeby. Namísto nákupu nebo pronájmu hardwaru či softwaru si je však předplatí prostřednictvím modelu spotřeby „Pay for what you consume“ [10].

3.1 Výhody HPC-as-a-Service

HPC poskytované jako služba má výhody oproti hostování vlastní HPC infrastruktury. První důležitou výhodou může být model spotřeby „Pay for what you consume“, který byl již zmiňován výše. Zákazník tak platí jen za zdroje, které skutečně využil. Vlastní řešení, konstrukce a následná správa vysoce výkonných výpočetních clusterů je velmi náročná na zdroje.

Společnosti či organizace poskytující HPC-as-a-Service mají tendenci rychleji integrovat novější technologie. Tyto technologie jsou pak přístupné uživatelům využívající jejich služeb. Někteří posky-

¹Masivní paralelismus je technika využívající velkého počtu procesorů či samotných počítačů k paralelnímu zpracování úloh.

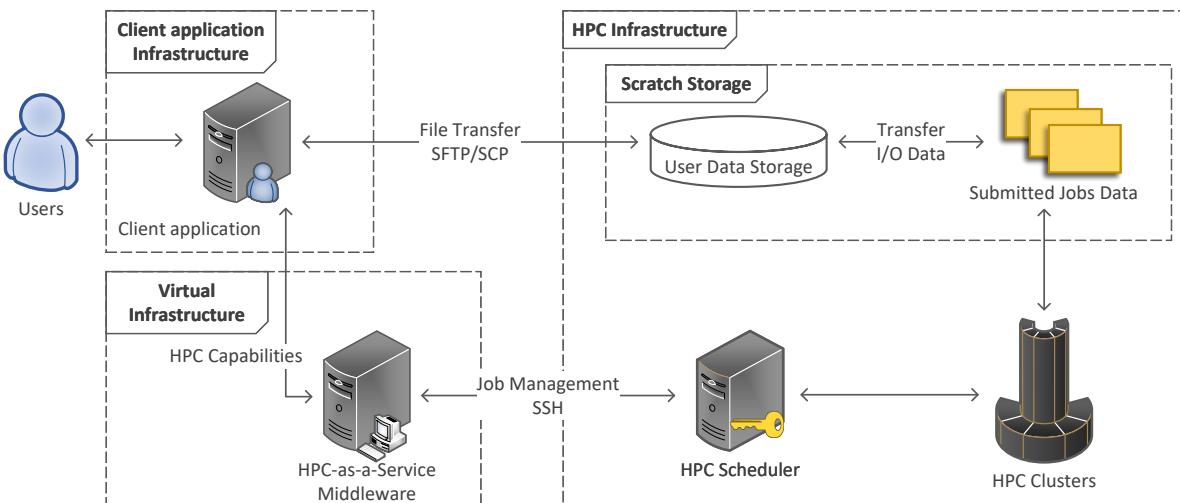
tovatelé HaaS umožňují jejich zákazníkům provádění aktualizací či škálování² výpočetních kapacit na vyžádání. [11]

3.2 Případy užití

V dnešní době se poskytovatelé HPC-as-a-Service snaží co nejvíce přizpůsobit požadavkům jejich zákazníků. HaaS využívají společnosti, které pro řešení jejich problémů vyžadují vysoké výkonné či paralelní zpracování dat. Poměrně velký segment zákazníků tvoří oblast Automotive³, ti využívají HPC například ke zkrácení času simulací proudění tekutin či plynů. Při využití výpočetních clusterů nejen oboru Automotive dochází k přesnějším a rychleji dosaženým výsledkům při vývoji. Dalším oborem, ve kterém je HPC hojně využíváno, je obor lékařství. Vývoj léčiv je díky paralelnímu zpracování dat na superpočítáčích efektivnější a rychlejší, dochází pak i k minimalizaci vzniku chyb.

HPC je využíváno v mnoha dalších oborech či odvětvích, patří tam i obor výzkumu a vývoje. HPC umožňuje analyzovat rozsáhlé datové soubory v poměrně krátkém čase a sdílet výsledky se spolupracovníky. Dále je HPC využíváno třeba ke 3D modelování a renderování scén, toho využívají grafická, herní a filmová studia.

Obrázek 3.1 ilustruje využití HPC-as-a-Service / HaaS.



Obrázek 3.1: HPC-as-a-Service / HaaS [12]

²Škálování je pojmem vyjadřující navýšování kapacit.

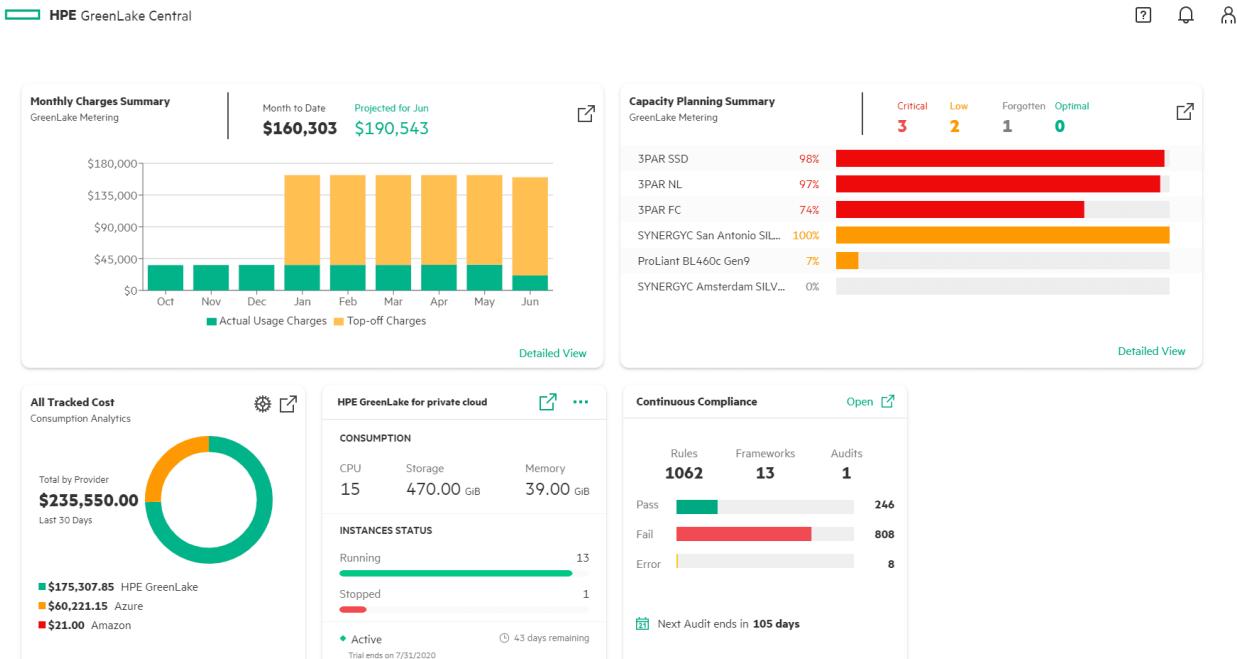
³Segment automobilového průmyslu.

3.3 Největší poskytovatelé platforem HaaS

Mezi největší poskytovatele HPC-as-a-Service v době psaní této práce patří společnosti Hewlett Packard Enterprise⁴(HPE) a Atos⁵. Obě tyto společnosti se dlouhodobě specializují na poskytování IT služeb.

3.3.1 HPE Greenlake

HPE nabízí z pohledu HaaS řešení, které bylo pojmenováno jako HPE Greenlake⁶. Pro potřeby zákazníků poskytuje platforma HPE GreenLake jednoduché a bezpečné řešení návrhu či využívání vlastní vzdálené HPC infrastruktury bez nutnosti správy fyzických výpočetních clusterů. Mezi hlavní výhody této služby patří obchodní model „Pay for what you consume“. Využívání HPC zdrojů je měřeno a zákazník platí jen za to, co skutečně spotřeboval. Protože požadavky na pracovní zátěž HPC mohou kolísat, platforma HPE Greenlake umožňuje měnit konfigurace HPC kapacit. Výpočetní clustery, které jsou součástí infrastruktury zmiňované služby, jsou pravidelně aktualizovány po stránce hardwaru i softwaru. Uživateli to přináší nové možnosti bez nutnosti jakéhokoli zásahu. Na obrázku 3.2 se nachází snímek z aplikace sloužící ke konfiguraci, správě a sledování využití výpočetních kapacit služby HPE Greenlake [13].



Obrázek 3.2: Snímek aplikace HPE Greenlake Central [14]

⁴<https://www.hpe.com/us/en/compute/hpc.html>

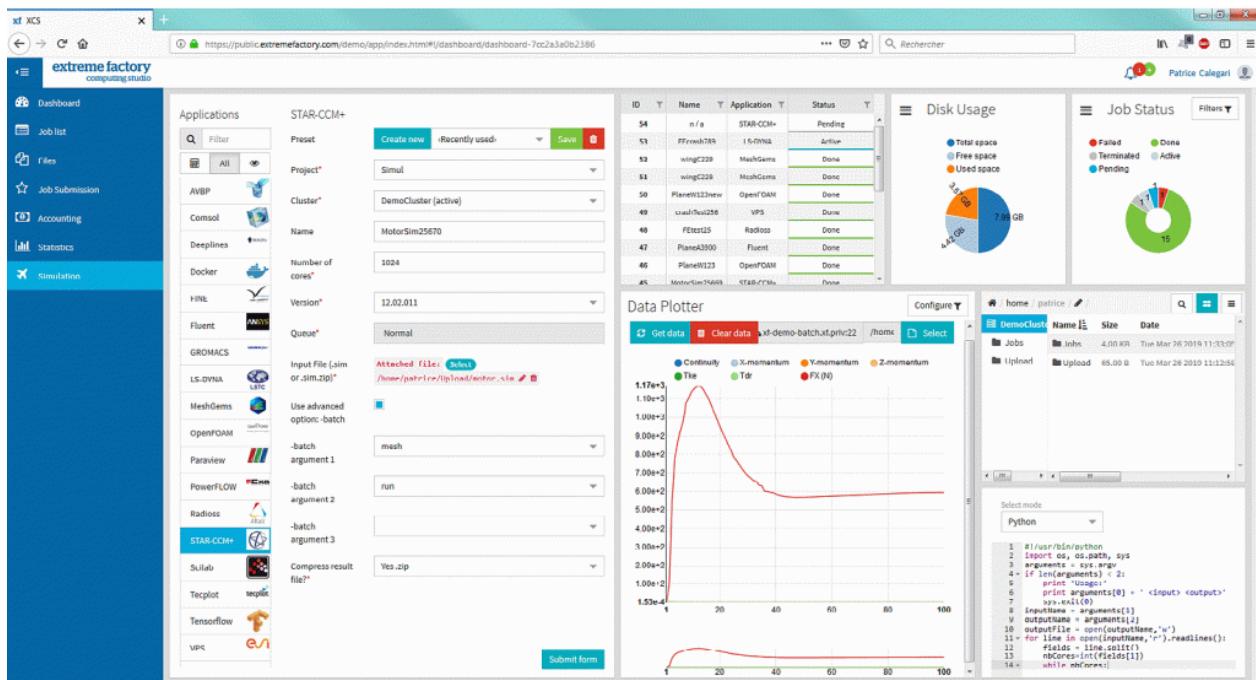
⁵<https://atos.net/en/>

⁶<https://www.hpe.com/us/en/greenlake/high-performance-compute.html>

3.3.2 Nimbix JARVICE XE

Společnost Atos nabízí své vlastní řešení problematiky HPC-as-a-Service v podobě platformy JARVICE™ XE⁷. Tato platforma umožňuje správu libovolné HPC úlohy na libovolném HPC clusteru. Nezáleží na tom, zda se výpočetní cluster nachází v soukromém datacentru uživatele nebo v cloudu. JARVICE™ XE poskytuje akcelerované aplikace a pracovní postupy, které využívají různorodou infrastrukturu. Podpora je zajištěna například i pro platformu InfiniBand⁸ a výpočetní jednotky pro paralelní zpracování dat. Společnost navíc uživatelům platformy umožňuje snadno přejít z lokálních řešení na veřejné clouдовé systémy nebo spravovat interní systémy jako soukromé cloudy.

Správa HPC aplikací prostřednictvím zmínované platformy probíhá přes rozhraní zvané The Smart HPC Web Portal. Toto rozhraní uživatelům systému zaručuje bezpečný a přímý přístup ke všem prostředkům a aplikacím. Po přihlášení má uživatel přístup ke kompletnímu pracovnímu prostředí přizpůsobenému jeho pracovní roli. Odtud může načítat a spravovat data, nastavovat parametry simulací, spouštět výpočty, sledovat jejich průběh a poté přejít k následnému zpracování a vizualizaci výsledků. Na obrázku 3.3 se nachází snímek obrazovky popisované aplikace.

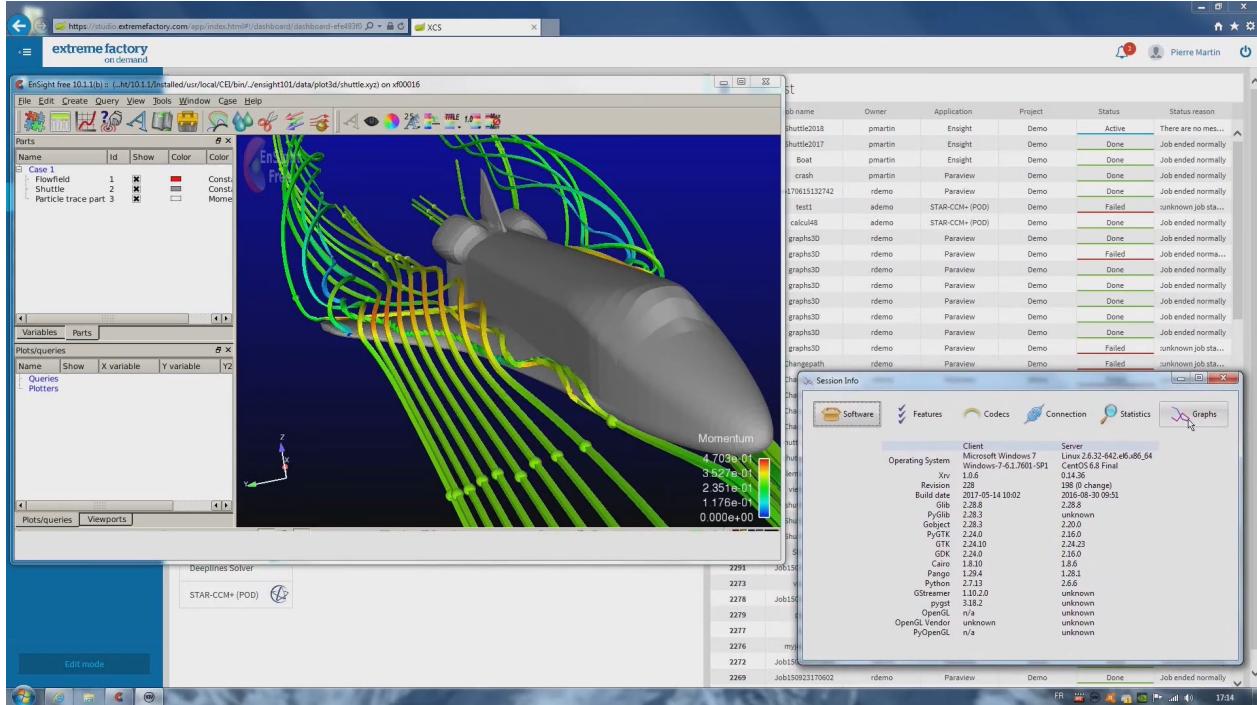


Obrázek 3.3: Snímek aplikace The Smart HPC Web Portal [15]

⁷<https://www.nimbix.net/jarvicexe>

⁸<https://www.nvidia.com/en-us/networking/products/infiniband/>

Vizualizace a následné zpracovávání výsledků je možné prostřednictvím modulu, který je součástí zmiňovaného správcovského rozhraní. Výsledky lze zpracovávat vzdáleně přímo na serveru a na pracovní stanici uživatele se šifrovaně přenášejí pouze zkomprimovaná data (obrázky) [16]. Příklad takové vizualizace v rozhraní popisovaného modulu se nachází na obrázku 3.4.



Obrázek 3.4: Snímek aplikace Atos Extreme Factory [15]

Kapitola 4

Seznámení s HEAppE Middleware

Kapitola je zaměřena na popis aplikačního rámce HEAppE Middleware.

4.1 Účel HEAppE Middleware

HEAppE Middleware¹ (High-End Application Execution Middleware) poskytuje aplikační rozhraní (API) pro kompletní správu úlohy na superpočítacovém clusteru, od vytvoření až po přenos zpracovaných dat. Jedná se o Open-Source² projekt implementující koncept HPC-as-a-Service a jeho cílem je zjednodušit komunikaci s HPC plánovačem a umožnit uživateli pohodlnou implementaci do jeho klientské aplikace či systému. HEAppE Middleware zajišťuje bezpečnou a šifrovanou komunikaci s výpočetním clusterem prostřednictvím protokolu SSH. Komunikace pak stojí na autentizaci pomocí asymetrického šifrování s využitím veřejného klíče (RSA).

Tato implementace konceptu HaaS je tvořena tak, aby respektovala interní bezpečnostní politiky výpočetního centra IT4Innovations³ a metodické pokyny ISO⁴. Především tedy z důvodu specifických bezpečnostních nároků nebylo možné použít některou z dostupných HaaS platforem. HEAppE Middleware je však použitelný i na další typy HPC infrastruktur, více informací o kompatibilitě HEAppE je uvedeno v sekci 4.2.5.

Abstraktně si můžeme HEAppE Middleware představit jako spolehlivého prostředníka mezi uživatelem a výpočetním clusterem, popř. jeho plánovačem. Poskytuje uživatelům vzdálený přístup k výpočetním clusterům se zaměřením na bezpečnost a jednoduchost přístupu.

Prakticky se jedná o službu, prostřednictvím které může koncový uživatel komunikovat s HPC clusterem přes standardizované aplikační rozhraní zvané jako REST API, stojící na HTTP komunikaci.

¹<https://heappe.eu/>

²Kód, který je navržen tak, aby byl veřejně přístupný.

³<https://www.it4i.cz/>

⁴<https://www.iso.org/home.html>

V době psaní této práce je HEAppE Middleware využíván ve 4 superpočítacových/datových centrech a je navázán na 7 projektů, které middleware využívají ke správě HPC úloh.

4.2 Technické zázemí HEAppE Middleware

Velký důraz je v tomto projektu kladen na multiplatformní využití a nezávislost použitého hardwaru. Tento aplikační framework je postaven na technologii .NET Core⁵, poskytuje REST API a při nasazování je využívána kontejnerizační technologie Docker. Komunikace s plánovačem výpočetního uzlu probíhá prostřednictvím protokolu SSH, uživatel je autentizován jedním z uložených servisních účtů metodou RSA.

V současnosti tento framework zajišťuje komunikaci s plánovači OpenPBS a Slurm. Podporuje autentizaci prostřednictvím SSH Agenta a je zajištěna podpora OpenID nebo OpenStack autentizace pro přístup k HEAppE Middleware API.

4.2.1 Konfigurace HEAppE Middleware

HEAppE je nutné před spuštěním konfigurovat. V první řadě je sestaveno konfigurační schéma (docker-compose) pro službu Docker, na kterém middleware běží. V Docker kontejneru se pak mimo samotný HEAppE Middleware nachází veškeré podpůrné služby, jako je MS SQL Server nebo SSH Agent. V návaznosti je přidána konfigurace samotné aplikace (AppConfig), která primárně specifikuje spojení s databází, port middleware API a další. Nedílnou součástí konfigurace je i počáteční stav dat uložených v databázi (seed), který si každý uživatel může specifikovat. Tato data jsou v okamžiku prvního spuštění HEAppE nahrána do databáze.

4.2.2 REST API

REST API je standardizované aplikační rozhraní využívající protokol HTTP. Služba poskytující toto API vystavuje tzv. endpointy (koncový uzel zpřístupněný uživateli) a specifikaci pro práci s nimi. V případě HEAppE Middleware se jedná o url adresy se specifikací JSON⁶ struktury dat, které systém na vstupu očekává. Z praktického hlediska může jít např. o komunikační bod pro vytvoření specifikace úlohy, na vstupu se očekává JSON struktura popisující úlohu a specifický řetězec, který uživatel obdrží po autentizaci na jiném endpointu. Na obrázku 4.1 se nacházejí vybrané endpointy REST API middleware z uživateslkého rozhraní Swagger⁷.

⁵<https://docs.microsoft.com/en-us/dotnet/core/introduction>

⁶JSON neboli JavaScript Object Notation je formát využívaný pro výměnu dat. Formát je jednoduše čitelný i pro člověka [17].

⁷<https://swagger.io>

JobManagement	
POST	/heappe/JobManagement/CreateJob Create job specification
POST	/heappe/JobManagement/SubmitJob Submit job
POST	/heappe/JobManagement/CancelJob Cancel job
POST	/heappe/JobManagement/DeleteJob Delete job
POST	/heappe/JobManagement/ListJobsForCurrentUser Get all jobs for user
POST	/heappe/JobManagement/GetCurrentInfoForJob Get current job information
POST	/heappe/JobManagement/CopyJobDataToTemp Copy job data to temp folder
POST	/heappe/JobManagement/CopyJobDataFromTemp Copy job data from temp folder
POST	/heappe/JobManagement/GetAllocatedNodesIPs Get allocated node IP addresses
JobReporting	
POST	/heappe/JobReporting GetUserResourceUsageReport Get resource usage report for user
POST	/heappe/JobReporting GetUserGroupResourceUsageReport Get resource usage for user group
POST	/heappe/JobReporting/GetResourceUsageReportForJob Get resource usage for executed job

Obrázek 4.1: Vybrané HEAppE Middleware REST API endpointy

4.2.3 Protokol SSH a metoda šifrování RSA

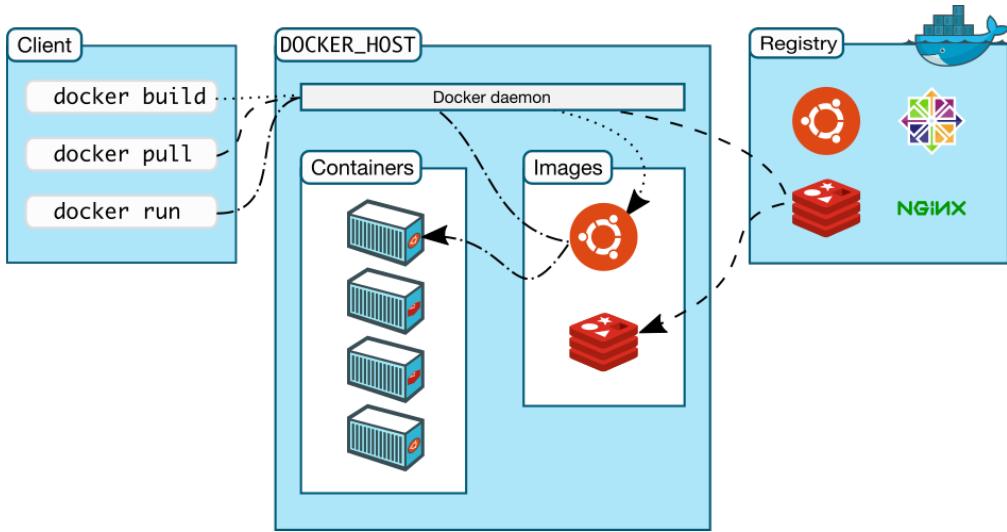
Jak už bylo popsáno výše, HEAppE Middleware ke komunikaci s HPC clustery využívá protokolu SSH. Koncový uživatel je od této komunikace odstíněn a komunikuje pouze s vrstvou REST API. Při navazování spojení SSH je relace autentizována prostřednictvím soukromého klíče jednoho ze servisních účtů v HEAppE Middleware, na HPC clusteru je uložen veřejný klíč. Tento typ komunikace uživateli zaručuje bezpečnost pro jeho práci. Hlavní výhodou asymetrického RSA šifrování je vysoká složitost pro dešifrování, metoda je založena na prvočíselném rozkladu a je v současné chvíli prakticky neprolomitelná.

4.2.4 Kontejnerizace a Docker

Při nasazování HEAppE je využívána technologie Docker. Ta umožňuje nakonfigurovat systém, a pak už jen HEAppE Middleware sestavit na libovolném serveru. Docker umožňuje spustit aplikaci v izolovaném prostředí zvaném kontejner. To umožňuje na hostitelském systému spustit několik kontejnerů současně. Kontejnery jsou oddělené a obsahují vše potřebné pro běh aplikace, takže se uživatel nemusí spoléhat na to, co je na hostitelském počítači aktuálně nainstalováno [18]. Docker automaticky stáhne požadované obrazy (image) a spustí aplikaci.

Výhoda tkví především v přenositelnosti specifikace na jiné servery, kontejner je oddělen od ostatních, tudíž je možné specificky nastavit jeho komunikační rozhraní dle potřeb uživatele.

Architektura technologie Docker je ilustrována na obrázku 4.2.



Obrázek 4.2: Architektura Docker [18]

4.2.5 Plánovače OpenPBS a Slurm

HEAppE Middleware v době psaní této práce podporuje plánovače OpenPBS⁸ a Slurm⁹, tyto plánovače jsou hojně využívány v superpočítacích centrech po celém světě. Jedním z cílů HEAppE Middleware je platformní nezávislost, komunikační rozhraní jednotlivých plánovačů je však rozdílné a musí se k nim přistupovat individuálně. Koncový uživatel je však od této skutečnosti odstíněn a může tedy pracovat stále stejně, bez nutnosti znalosti dalšího systému. Na pozadí to řeší samotný middleware.

4.3 Abstrakce HPC úlohy

HEAppE Middleware poskytuje uživateli určitou míru abstrakce nad skutečnou specifikací HPC úlohy. Úloha je v HEAppE nazvána jako Task. Několik těchto Tasků tvoří Job, ten obsahuje obecné informace a nastavení. Příkladem může být maximální doba běhu úlohy, zvolený cluster nebo maximální doba čekání na spuštění úlohy. Každý z takto specifikovaných Jobů obsahuje minimálně jednu úlohu (Task), tedy samostatnou specifikaci spuštěné úlohy na clusteru. Těchto Tasků je možné definovat více, může u nich být specifikovaná i závislost na ostatní úlohy. Specifikace Tasku především obsahuje požadavky na využití jader, časový limit pro zpracování úlohy, frontu úloh a další specifické parametry pro běh spuštěného programu.

Tasky je možné spouštět jen s vazbou na předem vytvořené šablony výpočtů. V těchto šablonách je definován program, který se má na clusteru spouštět a je zde specifikován výčet argumentů daného programu. Argumenty programu musí uživatel uvést vždy u specifikace úlohy (tasku).

⁸<https://www.openpbs.org>

⁹<https://slurm.schedmd.com>

Prostřednictvím HEAppE Middleware si uživatel definuje Job s Tasky, tento Job pak může spustit a kontrolovat jeho stav. V průběhu zpracovávání je možné kdykoli úlohu zrušit. HEAppE uživateli poskytuje možnost kompletně úlohu spravovat, a to bez nutnosti znalosti principů komunikace s plánovačem HPC clusteru.

Následuje příklad struktury příkazu ke spuštění úlohy na HPC clusteru, který plně nahrazuje jeden HEAppE RestAPI endpoint.

```
qsub -q <queue> -w e -N <job_name> -l h_vmem=<memory> -l h_rt=<time> -l s_rt=<time> -pe
      smp <num_slots> -o <outputlogfile> -e <errorlogfile> <pathToScript> <arg1> <arg2>
```

Listing 4.1: Struktura příkazu qsub [19]

4.4 Uživatelské a servisní účty

HEAppE Middleware mapuje HEAppE uživatelské účty na skutečné clusterové účty (servisní). Správce HEAppE instance je oprávněn k vytváření uživatelských účtů. Uživatel se autentizuje prostřednictvím svého loginu a hesla, OpenID nebo OpenStack v HEAppE. Následně obdrží vygenerovaný klíč (alfanumerický hash) označovaný jako Session ID s časově omezenou platností. Při volání dalších HEAppE RestAPI endpointů tento hash využívá. Při konečné komunikaci se superpočítacovým clusterem HEAppE využívá jeden z uložených servisních účtů. Tyto účty jsou schváleny superpočítacovým centrem a správce HEAppE instance stojí za „nezávadností“ spouštěných úloh.

HEAppE Middleware tyto clusterové (servisní) účty střídá, aby např. nedošlo k zablokování ze strany superpočítacového centra. Systém bez rotace využívaných servisních účtů by mohl nést znaky DOS/DDOS¹⁰ útoku a účty by mohly být preventivně zablokovány.

Z výše uvedeného textu je zřejmé, že uživatelé spouštějící úlohy prostřednictvím HEAppE Middleware mohou ve skutečnosti využívat jeden servisní clusterový účet. HEAppE řeší přístupy uživatelů k jejich datům na výpočetním clusteru. Každý uživatel HEAppE se pak nemusí verifikovat přímo superpočítacovému centru, k použití stačí mít HEAppE uživatelský účet.

4.5 Šablony výpočtů

Uživatel HEAppE Middleware musí ve specifikaci úloh uvádět požadovanou šablonu výpočtu. Tyto šablony výpočtů se v systému HEAppE označují jako Command Templates. Uživatel může využít jen schválené šablony výpočtů, které se nacházejí v databázi HEAppE Middleware. Zmíněné šablony výpočtů obsahují cestu k programu či skriptu, který se nachází na výpočetním clusteru a výčet parametrů, které uživatel při specifikaci úlohy musí nastavit. Uživatel HEAppE Middleware tedy

¹⁰DOS nebo také DDOS je typ útoku na internetové služby nebo stránky, jehož cílem je cílovou službu znefunkčnit a znepřístupnit ostatním uživatelům; může k tomu dojít zahlcením obrovským množstvím požadavků, ve variantně DDOS jde o distribuovaný DOS [20]

není oprávněn ke spuštění libovolného programu v rámci HPC úlohy, ale je odkázán na předem definované šablony výpočtů.

Command Templates vytváří administrátor dané instance HEAppE Middleware. Ten je zpravidla také odpovědný za „nezávadnost“ spouštěných programů či skriptů, které jsou uvedeny v šablonách výpočtů, protože se tyto programy spouští pod servisními účty na výpočetních HPC clusterech. Spouštěné programy nesmí porušovat politiky a nároky daného superpočítacového centra, účty by následně mohly být zablokovány, a tím by došlo ke znefunkčnění celého HEAppE Middleware.

4.6 Zajímavé funkcionality HEAppE Middleware

4.6.1 JobArrays

K úlohám definovaným prostřednictvím HEAppE je možné přidat specifikaci pro opakování spouštění úlohy. Tato funkcionalita je nazvana jako JobArrays, teminologie vychází z funkce plánovače. Uživatel specifikuje iterace dané úlohy a plánovač pak úlohu spouští v cyklu. Praktické využití této funkce je například u renderování scén a vykreslování grafiky.

Následuje příklad formátu pro definování JobArrays.

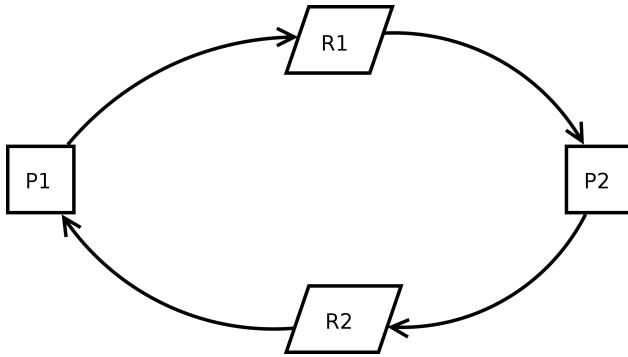
```
<start_index>-<end_index>:<step>
```

Listing 4.2: Struktura definice JobArrays

4.6.2 Závislosti úloh

U jednotlivých úloh je možné specifikovat jejich závislosti na jiných úlohách. Uživatel může jednoduše definovat závislosti úloh. Prakticky se jedná o podmíněné spouštění dalších navázaných úloh až po úspěšném vykonání úlohy předešlé. Omezuje se tímto možné plýtvání prostředky a výpočetním časem, které by mohlo nastat při selhání jedné z úloh, na jejímž výstupu závisí úloha další.

HEAppE ale musí řešit kontrolu těchto závislostí, aby plánovači byly zaslány validní specifikace úloh. Na straně výpočetního clusteru také probíhá kontrola specifikací, aby nedošlo ke kruhové/cirkulární závislosti. Zjednodušeně je tento problém možné specifikovat jako problém s nekonečným čekáním. Na straně výpočetního clusteru by mohl nastat tzv. Deadlock. Jedná se o uváznutí systému při vzájemném čekání. Na superpočítací se tento problém řeší kontrolou specifikace úloh a také detekcí kruhové závislosti v grafu. HEAppE cirkulární závislost kontroluje také, aby se celý proces urychlil a nedocházelo k chybám na straně výpočetního clusteru. Na obrázku 4.3 se nachází znázornění typického uváznutí – cirkulární závislosti procesů (P1, P2) a prostředků (R1, R2). Závislost je znázorněna šipkou.

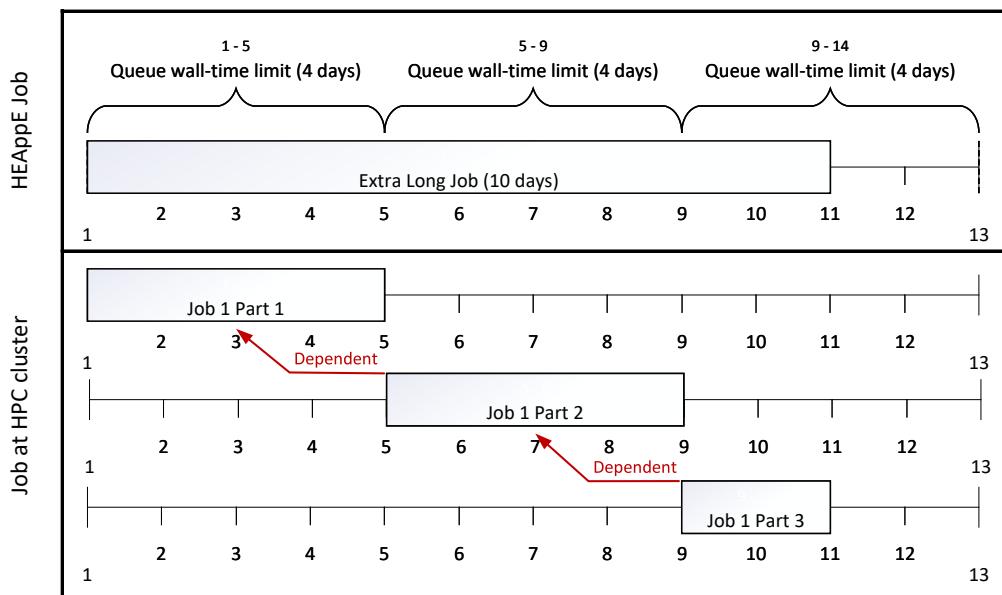


Obrázek 4.3: Cirkulární závislost [21]

4.6.3 Extrémně dlouhé úlohy

Každá výpočetní fronta pro úlohy má definován horní interval pro dobu vykonávání uživatelských úloh, v praxi je tento časový parametr nazýván jako Walltime Limit. Systém ukončí každou úlohu, pokud přesáhne časový limit pro vykonávání úlohy. Existují ale uživatelé, jejichž programy vyžadují extrémně dlouhý čas pro dokončení, který překračuje již zmíněný limit na výpočetní frontě superpočítacového clusteru. I na tento problém HEAppE Middleware pamatuje. Při definování úlohy je možné nastavit přepínač pro vykonávání v režimu „ExtraLong“. V tomto případě HEAppE logicky rozdělí úlohu na více podúloh se závislostmi na postupném zpracování. Rozdělení probíhá podle uživatelem uvedeného časového limitu a limitu pro zpracování na požadované výpočetní frontě superpočítací. Výhodou je to, že je uživatel odstíněn od manuálního rozdělování úlohy na podúlohy. Dále je takto upravená specifikace úlohy předána plánovači na superpočítací a ten se již stará o vykonání úlohy.

Tyto úlohy však musí podporovat „checkpointing“, tato funkce umožňuje ukládat stav úlohy tak, aby bylo možné výpočet po přerušení opět obnovit, aniž by se změnilo chování výpočtu. Zachovaný stav se nazývá kontrolní bod a pokračování se obvykle označuje jako restart [22].



Obrázek 4.4: Rozdělení extrémně dlouhé úlohy na podúlohy

Kapitola 5

Rozšíření HEAppE pro lokální výpočty

Jedním z hlavních cílů této práce je navrhnout a implementovat rozšíření HEAppE Middleware o možnost lokálních výpočtů, tedy umožnit uživateli nasimulovat spouštění úloh lokálně na počítači s využití rozhraní HEAppE, a to bez nutnosti spojení se superpočítáčem. Výhodou tohoto rozšíření je především možnost nezávislé přípravy specifikace úlohy a vyzkoušení práce s HEAppE.

Jelikož je systém HEAppE Middleware zpravidla nasazován za pomocí technologie Docker na servery superpočítáčového centra, je možné tento způsob pohodlně replikovat i na uživatelském počítači. Technologii Docker je možné nainstalovat jako službu na počítače s operačními systémy MS Windows, macOS i Linux.

Cílem tohoto rozšíření je co možná největší udržení systémové nezávislosti. Mělo by simulovat HEAppE komunikaci s výpočetním clusterem. Na straně simulovaného HPC clusteru v prostředí uživatelského počítače by mělo být dosaženo co nejpodobnějšího chování plánovače, jako je tomu na skutečných výpočetních clusterech.

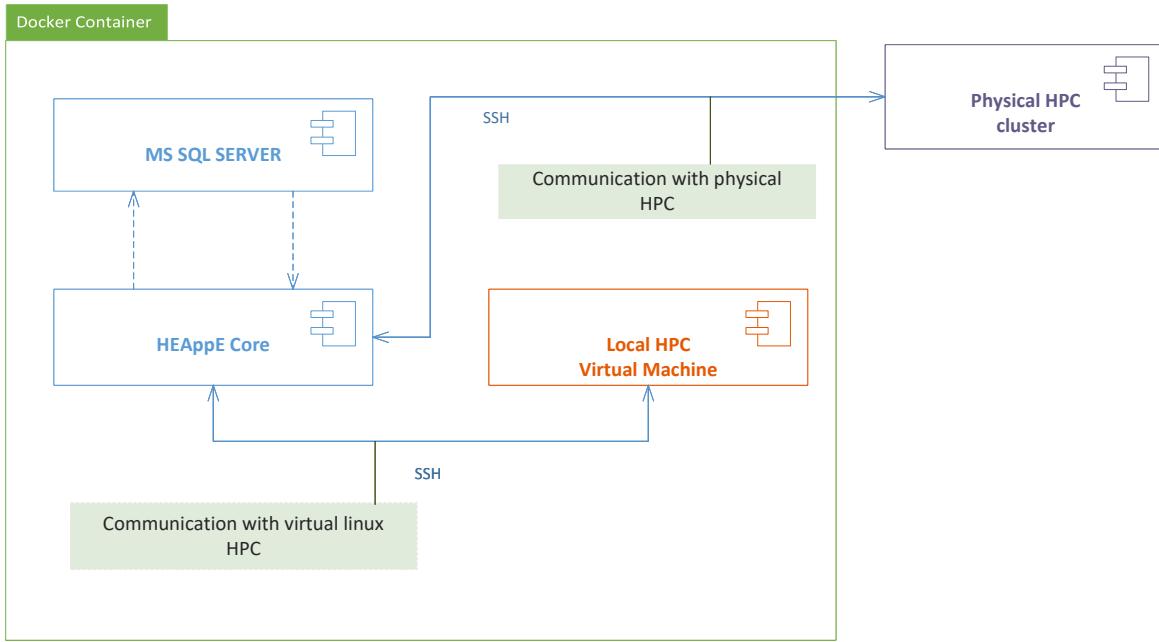
5.1 Návrh řešení

Toto rozšíření bylo navrženo s ohledem na platformní nezávislost a využití stávajících principů, na kterých stojí HEAppE Middleware.

Jelikož je HEAppE postaven na kontejnerizační technologii Docker, bylo navrženo, aby samotný virtuální HPC cluster byl spouštěn také jako kontejner. Pro uživatele by to mělo být poměrně jednoduché řešení, neboť by se jednalo jen o malou úpravu stávající Docker konfigurace (přidání vazby na kontejner) a přidání záznamu virtuálního clusteru do databáze (popř. upravení tzv. seedu) – přesně tak, jak je tomu i při přidávání specifikace skutečného HPC clusteru.

Kompletní specifikaci kontejneru pro Docker i konfiguraci HEAppE databáze je možné sdílet s uživateli, kteří o lokální spouštění úloh mají zájem. Pro tyto uživatele by mělo být velmi jednoduché systém HEAppE s dodanými konfiguračními soubory sestavit, spustit a používat.

Celá koncepce návrhu rozšíření se nachází na obrázku 5.1.



Obrázek 5.1: Návrh rozšíření o lokální HPC cluster

Lokální HPC Cluster je z technického hlediska instance virtuálního stroje s OS Linux. K násimulování chování plánovače, který je součástí superpočítáče, bude v tomto případě využito několik skriptů, které budou spouštěny na žádost HEAppE. HEAppE Middleware bude upraven především ve vrstvě pro komunikaci s HPC plánovači, avšak úpravy nebudou zasahovat do logiky užívání při spojení s plánovači na skutečných superpočítáčích, a to i z důvodu možnosti zanesení chyby do stávajícího řešení. Aplikační rámec HEAppE Middleware bude rozšířen o konektor (SchedulerAdapter) pro práci s lokálním virtuálním strojem simulujícím chování plánovače na superpočítáči.

Vrstvy v HEAppE jsou tvoreny s ohledem na pozdější úpravy či přidávání dalších adaptérů apod. Je využíváno návrhového vzoru Abstract Factory (abstraktní továrna), k tomuto návrhovému vzoru bude přihlízeno při výše popsaných úpravách.

5.2 Konfigurace virtuálního stroje

Nespornou výhodou použití Docker engine je snazší distribuce komponent systému. Pro jednotlivé kontejnery stačí vytvořit konfiguraci, popř. připravit obraz potřebných služeb. Služby v kontejneru jsou pak spouštěny dohromady, a je mezi nimi možné vytvořit komunikační kanály. Právě tímto způsobem pak bude realizována komunikace mezi HEAppE a virtuálním HPC clusterem.

Výhodu je také to, že do konfigurace je možné zanést aktuální stav virtuálního stroje tak, aby koncový uživatel měl vše připraveno. V případě této instance budou nastaveny komunikační porty,

vytvořen uživatel v systému OS Linux, uloženy skripty pro simulování komunikace s plánovačem, nebo zde např. budou připraveny testovací SSH-RSA klíče a povolena komunikace mezi kontejnery.

Tato konfigurace je v praxi označována jako dockerfile, jedná se o soupis kroků či povelů, které se stanou při spuštění kontejneru. Od stažení obrazu operačního systému, až po zpřístupnění komunikačního portu.

Následuje ukázka využívaného dockerfile pro instanci lokálního HPC clusteru.

```
FROM bitnami/minideb

RUN useradd -rm -d /home/heappeclient -s /bin/bash -g root -G sudo -u 1001 heappeclient
RUN echo 'heappeclient:pass' | chpasswd
WORKDIR /home/heappeclient

COPY .key_scripts/ ./key_scripts/
COPY .local_hpc_scripts/ ./local_hpc_scripts/
COPY .ssh/ ./ssh/
RUN apt-get update && apt-get install -y openssh-server
RUN apt-get install jq -y
RUN echo "/home/heappeclient/heappetests/" >> ./key_scripts/.heappeWorkDirInfo
RUN mkdir /home/heappeclient/heappetests/
RUN mkdir /home/heappeclient/heappetestsTemp/
RUN mkdir /var/run/sshd
RUN echo 'root:pass' | chpasswd

RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/
      sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i /
      etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]

RUN chown -R heappeclient: /home/heappeclient
```

Listing 5.1: dockerfile konfigurace pro virtuální stroj

5.3 Komunikace mezi HEAppE a virtuálním HPC

Komunikace mezi HEAppE a skutečným superpočítáčovým clusterem probíhá především prostřednictvím protokolu SSH. Komunikace je založena na principu klient-server (heappe-HPC). Prostřednictvím SSH HEAppE odesílá příkazy např. pro vytvoření, spuštění úlohy apod. Zpět dostává od plánovače výsledek provedení příkazu či skriptu, vyžádaná data, popř. aktuální stav úlohy.

Obdobně byla navržena komunikace i mezi HEAppE a virtuálním strojem simulujícím chování plánovače superpočítáčového clusteru. Komunikace taktéž probíhá prostřednictvím SSH s využitím autentizace tak, aby byla simulace co nejvhodnější.

5.4 Data a stav úloh

HEAppE Middleware je navržen tak, že se na obou stranách (HEAppE i HPC) uchovávají data o prováděných úlohách. HEAppE Middleware v době běhu úlohy na superpočítaci cyklicky zjišťuje stav svých spuštěných úloh a postupně se takto obě databáze synchronizují. Data jsou poskytována pouze na vyžádání, nejedná se tedy o sdílený datový prostor. Data jsou z HPC zasílána v předem známém formátu v textové podobě zpět při SSH komunikaci.

Obdobně je přistupováno k datům i v případě simulace HPC na lokálním virtuálním stroji. S ohledem na výkon a režii nejsou data na straně virtuálního HPC ukládána v databázi. V adresáři dané úlohy je sada souborů, obsahující data (stav, parametry) úlohy. Tato data se nachází v předem definované JSON struktuře, jsou ukládána a aktualizována při běhu úlohy. Po vyžádání jsou data zaslána v JSON struktuře textově zpět na HEAppE.

Pro práci s JSON strukturami slouží ve virtuálním stroji parser¹ jq [24]. Jedná se o program, který je schopen spravovat JSON struktury, umožňuje získávání dat, jejich aktualizaci, přidávání či odebrání. Tento program je instalován při sestavování Docker kontejneru. Žádost o instalaci se nachází v konfiguraci dockerfile.

```
{  
    "SessionCode": "string",  
    "CreatedJobInfoId": 0  
}
```

Listing 5.2: Příklad JSON struktury

¹Parser je program či jeho část provádějící syntaktickou analýzu textu. Analýza textu je proces, při němž je zpracováván text za pomoci definovaných pravidel daného jazyka [23].

5.5 Implementace na straně HEAppE Middleware

Jak už bylo zmíněno výše, HEAppE Middleware je stavěn na platformě .NET Core² a je vyvíjen v jazyce C#. Z hlediska pohledu architektury se systém HEAppE skládá z několika logicky oddělených vrstev. Uživatelské požadavky na REST API jsou zachyceny ve stejnojmenné vrstvě (RestAPI). Z této vrstvy jsou vstupní data po úspěšné validaci předávána do servisní a BusinessLogic vrstvy, ve které se nachází primární aplikační logika. Na závěr se požadavek dostává do vrstvy HpcConnectionFramework.

Vrstva HpcConnectionFramework je místem, kde dochází ke komunikaci s výpočetními clustery. V této vrstvě je implementováno několik adaptérů, které jsou schopny pracovat s odlišnými typy plánovačů na superpočítacích. Adaptéry jsou voleny na základě specifikace úloh, HEAppE musí mít předem v databázi informace o používaných clusterech a plánovačích. Právě v tomto místě se také nachází největší část implementace pro komunikaci s lokálním simulovaným výpočetním clusterem.

Samotná implementace výše popsaného řešení se skládá z adaptéra, ve kterém se nachází logika pro komunikaci s virtuálním strojem. V tomto místě jsou připravovány příkazy ke spouštění skriptů pro řízení úloh a zpracovávány odpovědi lokálního HPC. Důležitou součástí je také datový konvertor, jehož úkolem je mapovat získaná data z lokálního HPC v JSON formátu na objekty nebo jejich kolekce. S těmito objekty je pak dále v HEAppE pracováno již obvyklým způsobem, jako je tomu u komunikace se skutečnými HPC. Nedlouhou součástí celého adaptéra jsou připravené DTO³ objekty. Přesněji se jedná o objekty popisující stav prováděné úlohy. Při konstrukci tříd těchto objektů byl užit návrhový vzor Composite.

DTO objekty s namapovanými daty jsou pak v nižších vrstvách využity pro aktualizaci stavu a časů jednotlivých spouštěných úloh.

Funkcionalita je součástí HEAppE Middleware, projekt je dostupný na oficiálním veřejném GIT repositáři na adrese <https://github.com/It4innovations/HEAppE>.

5.6 Implementace na straně virtuálního stroje

Konfigurace již zmiňovaného virtuálního stroje s OS Linux je sepsána jako sekvence kroků pro spouštění v prostředí Docker. Součástí konfigurace je stažení obrazu OS Linux (Minideb), vytvoření uživatele pro práci, nastavení přístupových práv, nastavení domovského adresáře, instalace openssh serveru pro komunikaci nebo například instalace programu jq pro jednodušší práci s JSON strukturami.

Z důvodu možnosti jednoduché a rychlé úpravy byly pro řízení běhu simulovaného plánovače napsány scripty ve skriptovacím jazyce BASH. Zmíněné skripty programátor může jednoduše číst, orientovat se v nich, a protože se jedná o interpretovaný jazyk, programy napsané v BASH se

².NET Core je Open-Source vývojová platforma sloužící k vytváření aplikací.

³Objekt pro přenos dat.

spouštějí přímo bez nutnosti komplikace. Pro využití s Docker konfigurací tyto programy patří mezi nejlepší možné rychlé a jednoduché řešení. Celkově tato koncepce navazuje na stávající řídící skripty, které HEAppE využívá na skutečných výpočetních clusterech pro zjednodušení komunikace. Tyto skripty byly zvalidovány službou, která je dostupná na adrese shellcheck.net.

Mezi stávající využívané skripty patří například skript vytvářející pracovní adresář pro dané úlohy, přidávání či odebírání SSH klíčů nebo skript pro přesun dat. Tyto stávající skripty jsou využívány i na skutečných výpočetních clusterech.

Parametry jednotlivých skriptů, které jsou zasílány z HEAppE, jsou enkódovány do formátu Base64, aby nedocházelo k nechtěným problémům v případě zasílaných nepovolených znaků. Na straně virtuálního i skutečného výpočetního clusteru jsou tyto parametry dekódovány a užívány v textové podobě dle zvolené logiky.

Konfigurace popisovaného virtuálního stroje se všemi řídícími skripty a soubory je dostupná z veřejného GIT repositáře na adrese <https://github.com/It4innovations/HEAppE-scripts>.

```
Base64: VGVzdG92YWPDsRBOZXh0IHBybyB1bmVs2Rvds0hbs0tLiAoPC8+JSQjXiZAKQ==  
UTF-8: Testovací text pro enkódování. (</>%$#^&@)
```

Listing 5.3: Příklad kódování Base64

Byla vytvořena pětice skriptů, jejichž úkolem je nasimulovat chování plánovače. Jedná o skripty, které souží k přípravě úlohy, spouštění úlohy, získávání aktuálního stavu úlohy, ukončování úlohy nebo získávání počtu úloh na simulovaném clusteru. Všechny tyto podpůrné skripty pracují jen se soubory a adresáři, aby nedocházelo ke zbytečné zátěži hostitelského počítače (není použit DB server apod.).

5.6.1 Skript pro přípravu úlohy

Tento skript po spuštění připraví specifikace a parametry úlohy do kořenového adresáře úlohy na simulovaném HPC clusteru. JSON struktura s popisem úlohy je vložena do souboru *.job_info*. Příkazy kódované v Base64 jsou uloženy do souboru *.commands*. S těmito soubory, přesněji s jejich obsahem, pracují další skripty, které jsou popsány níže.

5.6.2 Skript pro spouštění úlohy

Skript věrohodně simuluje chování postupného spouštění jednotlivých úloh. Tyto úlohy se staví do fronty. Využívá program jq k úpravám JSON struktury celé úlohy, která je uložena v souboru adresáře úlohy. Skript pracuje následujícím způsobem:

1. Načtení předaných parametrů pro spouštění (příkazy uložené v souboru *.commands* v kořenovém adresáři dané úlohy)

2. Rozdelení jobů na jednotlivé na sebe navazující tasky
3. Zapsání aktuálního času do stavové struktury úlohy (soubor *.job_info* v kořenovém adresáři dané úlohy)
4. Zapsání stavu R (running) a PID (process ID) do stavové struktury
5. Správa jednotlivých tasků (s ohledem na jobArrays, závislosti)
 - (a) Vstup do adresáře tasku
 - (b) Uložení dat do stavové struktury k danému tasku (stav, čas)
 - (c) Spuštění úlohy
 - (d) Uložení stavu a času po dokončení úlohy
6. Vstup do adresáře jobu
7. Zapsání stavu a časů daného jobu (agregace stavů úloh)
8. Ukončení skriptu

Aktualizace datové struktury pro ukládání stavů (v souboru *.job_info*) je okamžitě propojena do souboru po každé úpravě z důvodu nemožnosti předvídat čtení aktuálního stavu prostřednictvím HEAppE. Data musí být v každém kroku aktuální.

5.6.3 Skript pro získání stavové struktury úlohy

Po vyžádání dat (spuštění skriptu s parametry úlohy) je přistoupeno do adresáře úlohy a na výstup skriptu jsou odeslána data v textové podobě (JSON). Tato data jsou pak přijata v HEAppE a jak již bylo výše popsáno, jsou namapována na DTO objekty, se kterými se dále pracuje a v posledním kroku dojde k synchronizaci dat s databází HEAppE.

5.6.4 Skript pro získání počtu úloh

Skript na výstupu vrací počet adresářů s úlohami, které byly prostřednictvím HEAppE a řídících skriptů na straně simulovaného HPC clusteru vytvořeny. Tento skript je využíván pro získávání informací o clusteru pomocí HEAppE endpointu. Výstupem je report s jednoduchou statistikou užívání daného clusteru.

5.6.5 Skript pro ukončení aktuálně běžící úlohy

Protože HEAppE podporuje ukončení aktuálně běžící úlohy na HPC clusteru, bylo nutné vytvořit skript i pro virtuální HPC cluster, který ukončí právě běžící proces, jenž je reprezentací úlohy či agregací úloh. Tento skript přečte uložené číslo procesu (PID), které se nachází v souboru *.job_info*

na lokálním stroji simulující chování HPC plánovače a zašle signál danému běžícímu procesu k vypnutí (příkaz kill). Následně jsou změněny stavy a časová razítka ve struktuře úlohy na virtuálním HPC clusteru.

5.7 Nutné kroky ke spuštění

V případě, že chce uživatel využívat HEAppE s rozšířením pro simulaci výpočtů na lokálním počítači, tak musí provést následující kroky.

1. Získání konfigurace kontejneru pro simulaci HPC se vsemi skripty a soubory (obsahuje konfiguraci dockerfile)
2. Úprava docker konfigurace HEAppE pro spouštění kontejnetu lokálního clusteru (příklad níže)
3. Přidání clusteru do výchozího stavu dat databáze (seed)
 - (a) Protože se bude jednat o komunikaci mezi Docker kontainery, musí být atribut MasterNodeName, který se využívá jako Host při nastavování spojení na cluster, nastaven na hodnotu "host.docker.internal" – není možné využít adresu localhost ani 127.0.0.1
 - (b) Musí být nastaveny údaje pro autentizaci (username, password, private key, key file)
 - (c) Musí být vytvořen nový uzel s frontou, kterou uživatel bude využívat na virtuálním HPC
 - (d) Atribut SchedulerType musí být nastaven na hodnotu 1, interně je v logice HEAppE tento typ plánovače využit při volbě adaptéru ve vrstvě HpcConnectionFramework

```
localhpc:  
  container_name: localhpc  
  build: C:/path/to/LocalLinuxHPCclusterConf  
  ports:  
    - "49005:22"
```

Listing 5.4: Docker konfigurace pro spouštění kontejnetu lokálního clusteru

Popis nutného nastavení s přesnými údaji je uveden v textovém souboru u samotné konfigurace kontejneru.

Bod č. 1 je možné nahradit za spuštění Docker kontejneru lokálního clusteru zvlášť. Bez návaznosti na ostatní služby HEAppE.

V příloze se nacházejí přesnější ukázky výše popsaných segmentů, které je nutné přidat do konfigurace dat pro databázi HEAppE (seed).

Kapitola 6

Uživatelsky definované šablony výpočtů

Dalším cílem této práce je navrhnou a implementovat řešení, které uživateli umožní jednoduchou správu šablon výpočtů (vytváření, modifikace a mazání). Taková šablona (v HEAppE nazvaná jako Command Template) obsahuje cestu k programu na clusteru a požadované parametry daného programu. Uživateli bude zpřístupněn nový endpoint na REST API, pomocí kterého bude možné novou šablonu vytvořit.

V současné chvíli HEAppE poskytuje funkcionality pro vytvoření tzv. generického command template. Prakticky se jedná o dynamicky vytvářející se šablonu výpočtů za běhu. Generická šablona bude použita pro vývoj tohoto řešení.

6.1 Generická šablona

Generic command template je postaven tak, že není nutné měnit jeho specifikaci při změně cesty ke spouštěnému programu na superpočítacovém clusteru. Uživatel tedy může používat generickou šablonu a spouštět různé typy výpočtů. Při vytváření úlohy se využije místo klasického Command Template tato generická šablona. Tato šablona se využívá primárně pro účely testování, pro pravidelné užívání je určena přesně definovaná šablonu, která je uložena v databázi HEAppE. Ta obsahuje cestu ke skriptu, který je spouštěn při vykonávání úlohy a set požadovaných argumentů skriptu.

Generická šablona obsahuje uživatelem specifikovanou cestu ke skriptu a parametry zadané ve specifickém formátu. Na clusteru se nachází skript, který spustí zadaný skript s parametry zadané uživatelem.

Před samotným spuštěním se však provádí validace zadaných parametrů a ověření existence skriptu na clusteru. Skript, který je spouštěn prostřednictvím generické šablony výpočtů, musí obsahovat hlavičku s parametry skriptu. Tato hlavička je v HEAppE přečtena a následně jsou názvy předaných argumentů či parametrů porovnány se zadanými parametry uživatelem.

```
#!/bin/bash
#HEAPPE_PARAM iterations
#HEAPPE_PARAM message
```

Listing 6.1: Ukázková hlavička skriptu s „generickými“ parametry iterations a message

Po validaci je na clusteru spuštěn řídící skript, který namapuje zadané parametry na proměnné, které je následně možné v programu využít.

6.2 Návrh řešení

Aby si uživatel mohl před přesnou specifikací nově vytvářené šablony výpočtů šablonu vyzkoušet, bude uživatelské vytváření nových šablon výpočtů navázáno na generickou šablonu výpočtů. Při vytváření nové šablony uživatel uvede stejné údaje, jako při užívání generické šablony ve specifikaci úlohy.

Během vytváření šablon výpočtů bude zkонтrolován uživatelem zadaný skript (hlavička s parametry) pro požadované názvy parametrů. Pokud bude vše souhlasit, šablona výpočtů se vytvoří.

Další funkcionalitou bude modifikace již existující šablon výpočtů, editovat bude možné atributy, které jsou ke specifikaci tzv. Command Template využívány. Pokud uživatel změní cestu k cílovému skriptu, který se spouští na clusteru, dojde tím i ke změně dostupných parametrů skriptu. Tyto parametry musí být uvedeny v hlavičce cílového skriptu. V čase změny musí být již tento skript dostupný na superpočítáčovém clusteru.

Součástí správy šablon výpočtů je i mazání existujících Command Template. Tato funkcionalita však nebude fyzicky mazat záznamy daných Command Template z databáze z důvodu možných referencí na úlohy. Pokud by v databázi existovala šablona výpočtů, která již byla použita ve specifikaci HPC úlohy, bylo by nutné kaskádově smazat záznamy ze všech tabulek, které obsahují referenci na Command Template. Tato funkce tedy bude nastavovat příznak IsEnabled na hodnotu False v daném záznamu Command Template. Pak tuto šablonu výpočtů systém HEAppE neumožní dále využívat. Bude třeba upravit logiku aplikace, upravit model a vytvořit novou migraci databáze¹.

Výše popsané funkcionality budou dostupné jako endpointy na REST API HEAppE Middleware. Bude vytvořen nový Management controller² v kódu HEAppE. Pro využívání těchto funkcionalit bude oprávněn pouze uživatel s rolí Administrátora.

6.3 Implementace

Do vrstvy REST API byl přidán nový controller a trojice endpointů s názvem CreateCommandTemplate, EditCommandTemplate a RemoveCommandTemplate. Pro mapování dat zadaných uži-

¹Migrace databáze je proces, při kterém dochází k verzování databáze/modelu aplikace.

²Controller je část systému HEAppE reagující na události přicházející od uživatele.

vatelem byly vytvořeny třídy popisující vstupní data. Dále byla upravena servisní vrstva a vrstva aplikační (business) logiky tak, aby bylo možné implementovat výše popsané funkcionality HEAppE.

Po obdržení žádosti na endpointu CreateCommandTemplate proběhne validace, zda zadaná generická šablona existuje, a zdali odkazovaný skript obsahuje v hlavičce názvy parametrů. Hlavička má z definice jednoznačný formát, proto se při parsování názvů parametrů z textu využívá regulárního výrazu.

```
#HEAPPE_PARAM ([A-z0-9]+)\n
```

Listing 6.2: Regurální výraz pro parsování názvů parametrů

Po úspěšné kontrole je vytvořen objekt CommandTemplate s vazbou na skript a parametry. Následně je šablona uložena do databáze a připravena k použití.

Správa šablon je složitější operace, předpokládá se, že správu jednotlivých šablon bude provádět poučená osoba pro práci s HEAppE pro další uživatele, kteří se starají primárně o vytváření a spouštění úloh. Proto je funkctionalita dostupná pouze pro uživatele s rolí Administrátora.

Analogicky dle výše popsaného návrhu řešení fungují endpointy EditCommandTemplate a RemoveCommandTemplate.

6.4 Prerekvizity pro správné fungování

Předpokladem pro vytváření nových šablon výpočtů je existence generické šablony v systému HEAppE. Dále musí uživatel vytvořit skript, který bude chtít spouštět společně s hlavičkou obsahující parametry ve výše uvedeném formátu. Při vytváření uživatel také může uvést další parametry pro tvorbu šablony výpočtů.

```
{
    "SessionCode": "string",
    "GenericCommandTemplateId": 0,
    "Name": "string",
    "Description": "string",
    "Code": "string",
    "ExecutableFile": "string",
    "PreparationScript": "string"
}
```

Listing 6.3: JSON struktura pro endpoint CreateCommandTemplate

Kapitola 7

Testování softwaru

Cílem testování softwaru je zjištění kvality daného softwaru, zda splňuje veškeré smluvěně nároky například na výkonnost nebo správnost řešení a pomáhá odhalovat chyby. Testování je nedílnou součástí vývoje SW, obvykle se software či jeho části testují průběžně. A to především z důvodu včasného odhalení chyb, které je možné v průběhu vývoje snáze opravit. Testování softwaru může sloužit jako prostředek k mitigaci rizik, cílem je snížení či eliminace rizik spojených s nasazením chybného kódu do produkčního prostředí.

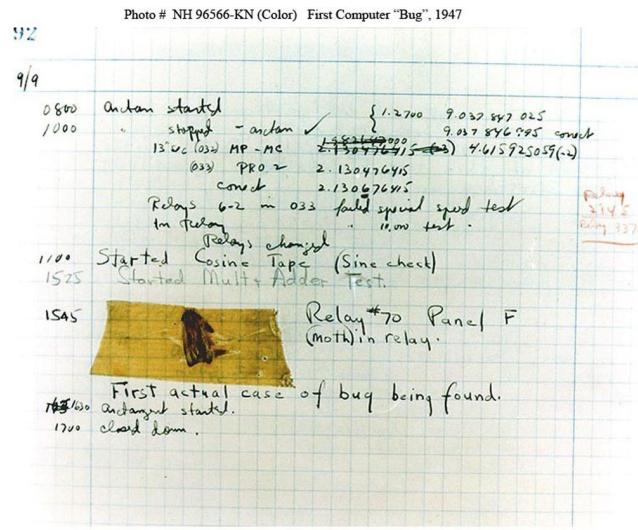
7.1 Historie testování

První chyba či selhání celého počítače bylo zaznamenáno v padesátých letech 20. století na Harvardově univerzitě. Chyba však nestála na straně softwaru, nýbrž příčinou byla zaklíněná můra v součástkách tehdejšího sálového počítače s označením Mark II, a ten vypověděl službu. Z tohoto prostého důvodu se dnes při řeči o chybách v programu využívá označení „moucha“ či „bug“. Prvotně toto označení použil Thomas Edison v jeho publikacích z 19. století, nyní však šlo o první počítačovou chybu.

Chyba v programu vede k selhání softwaru. Záznam o této chybě vzniknul v roce 1945. Podařilo se dochovat i onu můru, tedy strůjce celého problému [25]. Kopie tohoto záznamu se nachází na obrázku 7.1.

7.2 Proces vývoje softwaru

Mezi základní modely popisující proces vývoje softwaru patří model velkého třesku, model „programuj a opravuj“, vodopádový model nebo spirálový model. Mezi stěžejní body propracovanějších modelů se řadí specifikace, vývoj a testování. Každý tým vyvíjející software si může tento proces definovat tak, aby byl efektivní a splňoval očekávané podmínky zadavatele. Metodika či proces vývoje softwaru obvykle kombinuje hned několika principů z výše vyjmenovaných modelů. Především se



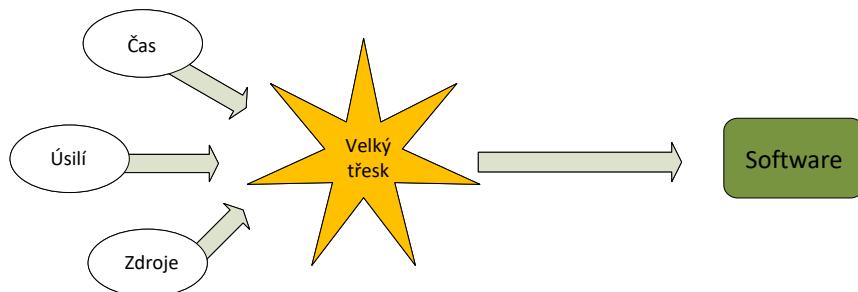
Obrázek 7.1: Záznam o první počítačové chybě [25]

však proces ubírá směrem iteračního vývoje softwaru. Vývoj softwaru v iteracích či cyklech vychází z praktických zkušeností především z důvodu přicházejících změn či doplňování specifikací už při vývoji softwaru.

Velmi důležitým krokem celého procesu vývoje softwaru je testování. Testování slouží především k odhalování chyb, které mohly být při vývoji zaneseny do kódu, a na první pohled nemusí být jasné, že se v kódu tyto chyby nacházejí. Proto je hned několik principů, jak software testovat. Metodiky testování se liší u různých projektů, a z praxe je zřejmé, že i přes sebelepší postup při testování softwaru je možné, že se chyba objeví až při užívání aplikace zákazníkem (obecně uživatelem). Cílem testování je minimalizace možnosti objevení chyb až ve fázi, kdy aplikaci využívá koncový uživatel. Některé testy či testové sady je možné automatizovat a zefektivnit tak fázi testování.

7.2.1 Model velkého třesku

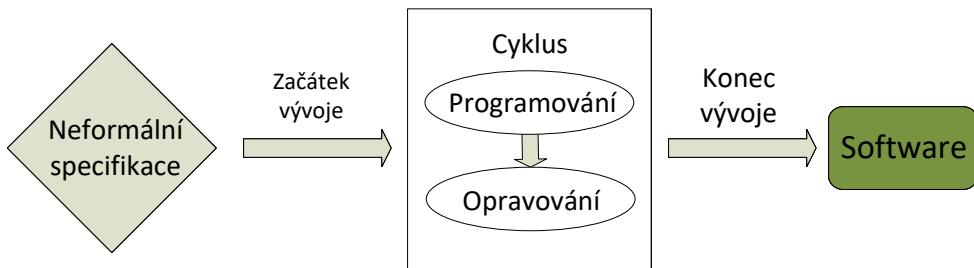
Metoda vývoje, při které se využívá modelu velkého třesku stojí, na jednoduchosti celého konceptu. Veškeré úsilí je soustředěno na vývoj softwaru – konečného produktu. Model počítá s velmi nízkou úrovní specifikace výsledného softwaru. Součástí tohoto modelu není fáze formálního tetování [26]. Model je vyobrazen na obrázku 7.2.



Obrázek 7.2: Model velkého třesku

7.2.2 Model „programuj a opravuj“

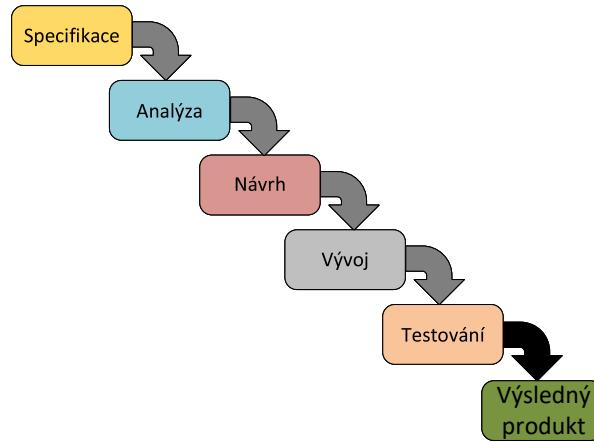
Tento model nevyžaduje příliš mnoho režie či řízení. Je určen pro malé projekty vyžadující prezentaci výsledků v rychlém sledu. Začíná se hrubou představou o finálním softwaru, pokračuje cykly vývoje, testování a opravou chyb. Po několika iteracích těchto cyklů je rozhodnuto o ukončení práce a vydání softwaru. Model „programuj a opravuj“ se v praxi využívá např. u prototypů [26]. Model je vyobrazen na obrázku 7.3.



Obrázek 7.3: Model „programuj a opravuj“

7.2.3 Vodopádový model

Vodopádový model spočívá v kontinuálním vývoji s přesně danou specifikací. Jednotlivé kroky tohoto modelu jsou atomické, nepřekrývají se a „není cesty zpět“. V jednotlivých krocích není možné postupovat zpětně, musí se dokončit daná iterace a začít specifikací znova [26]. Model je vyobrazen na obrázku 7.4.

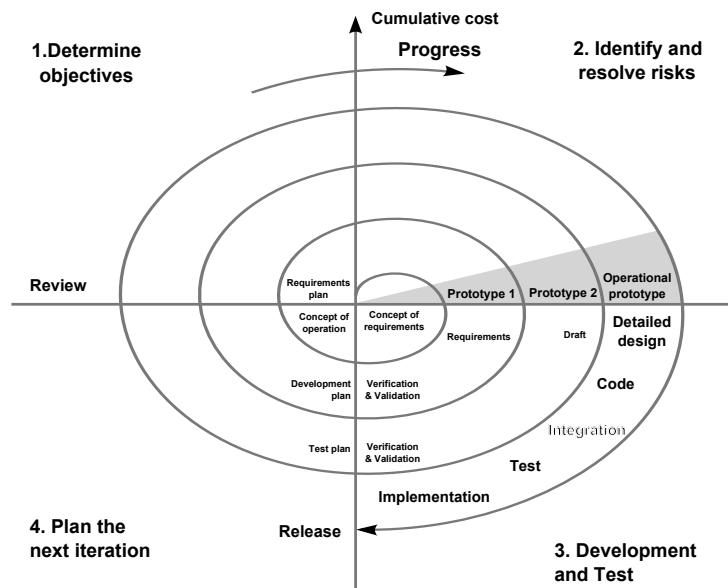


Obrázek 7.4: Vodopádový model

7.2.4 Spirálový model

Spirálový model řeší nedostatky výše uvedených modelů. Vývoj softwaru, který se řídí postupem daným spirálovým modelem je efektivní. Nevýhodou může být podstatně větší režie, než tomu bylo u modelů jako je model velkého třesku či model vodopádový. Jednotlivé kroky ve spirálovém modelu se mohou na rozdíl od vodopádového modelu překrývat [26].

V modelu jsou patrné 4 skupiny akcí, které je třeba provádět. Jmenovitě se jedná o plánování, analýzu rizik, vývoj a zhodnocení. Vývoj podle tohoto modelu probíhá v iteracích, při každém průchodu cyklem je pomyslně dokončena další část spirály. Model je vyobrazen na obrázku 7.5.



Obrázek 7.5: Spirálový model [27]

7.3 Testování API

Předchůdcem API bylo vzdálené volání procedur (RPC), které vycházelo z potřeby zpracovávaní velkých dat či náročných komplexních výpočetních operací. V této době už k řešení zmíněných problémů nestačily procesory v osobních počítačích. Proto se zavedla metoda distribuovaných výpočtů, při kterých uživatelé výkon svých počítačů sdíleli. Možnost rozdělit úlohy na více částí se ale neobešla bez potíží. Například pokud někdo vypnul svůj počítač v době, kdy byl využíván ke vzdálenému zpracování úloh jiných uživatelů, mohlo dojít k tomu, že se proces vypnul, aniž by dokončil svou práci.

V dnešní době můžeme hovořit o dvou typech API. První možností je lokální aplikační rozhraní využívané například k interakci s operačními systémy nebo databázemi. Příkladem API operačního systému může být systémové volání pro čtení ze souboru. Tato systémová volání jsou programátorovi však skryta za funkcemi programovacích jazyků.

Z druhého pohledu na aplikační rozhraní se jedná o způsob propojení organizací či aplikací. Toto propojení může zajíšťovat vzájemnou integraci. Strana využívající API tak nemusí implementovat své vlastní řešení, které již strana poskytující API vyřešila. Na straně druhé může dojít na straně poskytovatele API k chybě, v tomto případě může u strany využívající API dojít také k problémům, které je nutné řešit [28].

7.4 Manuální a automatizované testování

Testování softwaru je možné rozdělit do dvou základních skupin. Manuální testování zpravidla provádí člověk. Naopak automatizované testování zajišťuje stroj, který dle zvoleného scénáře většinou generuje požadavky na systém a čeká na odpověď, kterou porovnává s předpokládaným výstupem.

Automatizované testování nám může v krátkém čase přinést mnoho zajímavých výsledků. Dává nám náhled na to, na co se v daném systému zaměřit. Stojí však na poměrně náročném vytváření testových plánů či schémat. V případě nalezení chyby při automatickém testování se ale obvykle k problému musí dostat v posledním kroku i člověk, jehož úkolem je nalezení chyby v kódu. Využití automatizovaných testů je nesporně výhodné, především z časových důvodů, neboť můžeme stejně scénáře spouštět několikrát za sebou, např. s upravenými vstupy, které si můžeme nechat generovat. Výstupem automatizovaného testování je většinou souhrn či stručný přehled akcí, které se provedly. Tyto akce jsou pak označeny, zda se výstup rovná očekávanému výstupu. V případě projektu HEAppE Middleware jde o testování rozhraní REST API.

Automatizace testování je takřka nutností. Manuální techniky jednoduše neumožňují dostatečné testování ke zmírnění rizik, vzhledem k tomu, že jedno API samo o sobě může vyžadovat stovky testů, které je ve většině případů nutné opakovat [28].

7.5 Framework pro automatizované testování HEAppE

Mezi základní úkoly pro automatizované testování softwaru se řadí výběr frameworku¹. Pro účely HEAppE Middleware byl vybrán Robot Framework [29]. Jedná se o automatizační Open-Source framework, který pracuje s vytvořenou provozuschopnou aplikací či API. Tento framework se využívá na úrovni integračního testování, v praxi jde o tzv. smoke testy [30]. Hlavním úkolem smoke testů je rychlé vyhodnocení funkčnosti aplikace, aby bylo možné přejít k dalšímu vývoji. Pomáhají odhalit nefunkční části aplikace či aplikačního rozhraní. Robot Framework je schopný pracovat s grafickým uživatelským rozhraním, ale pracuje i na úrovni REST API.

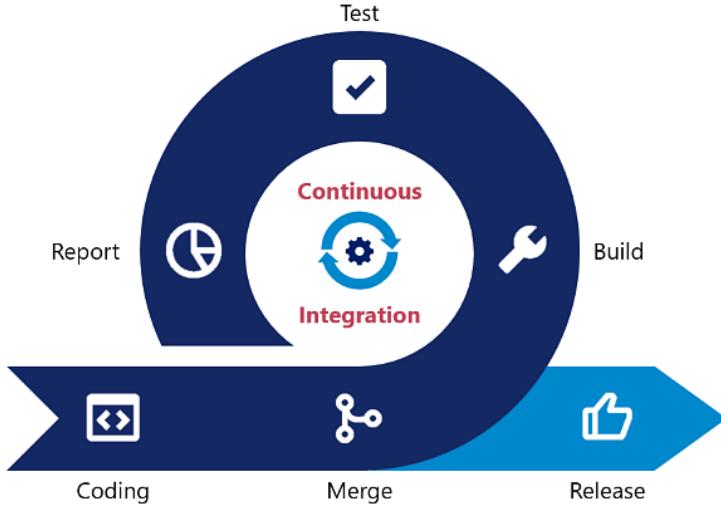
Robot Framework je nezávislý na operačním systému, jádro frameworku pak stojí na běhovém prostředí Python Interpreter². To umožňuje především snadnou rozšířitelnost a použitelnost. Základem automatizovaného testování prostřednictvím Robot Frameworku je sestavení testovacích plánů. Jedná se o popis akcí, které má framework vykonat. V případě HEAppE Midleware bude Robot Framework testovat endpointy REST API, pro dané požadavky pak budou dynamicky vytvářeny JSON specifikace. Základním plánem pro automatizované testování bude autentizace, vytvoření a spuštění úlohy. Tento plán může být dále rozšiřován.

Velkou výhodou tohoto frameworku je možné napojení na průběžnou integraci ve fázi vývoje. Průběžná integrace (Continuous Integration) slouží k urychlení nalezení chyb, integrace je spouštěna na integračním serveru podle předem zvoleného schématu. Metoda průběžné integrace je využívána i v projektu HEAppE, v průběhu sestavování aplikace je možné provést sadu testů z Robot Frameworku. Vývojáři pak budou mít lepší přehled o funkčnosti systému v dané fázi vývoje. Průběžná integrace je spouštěna předem nastavenou akcí. Může se například jednat o úpravu repositáře³ či dané větve repositáře projektu. Sestavení a smoke testy jsou pak spuštěny automaticky, vývojář tak může sledovat stav integrace s informacemi o provedených testech. Metoda Continuous Integration je znázorněna na obrázku 7.6.

¹Framework je set programů, knihoven a rozhraní, které programátorovi zjednodušují práci. Framework většinou stačí nakonfigurovat a používat.

²Python Interpreter převádí kód psaný v jazyce Python do jazyka strojových instrukcí, tyto instrukce jsou následně spuštěny a program v jazyce Python je vykonáván.

³Repositář je typ datové úložiště verzovacího systému.



Obrázek 7.6: Průběžná integrace [31]

7.6 Testovací scénář pro HEAppE

Robot Framework je skvělým kandidátem pro testování hotových softwarových řešení. Pomocí tohoto frameworku je možné otestovat aplikační rozhraní daného systému z pohledu uživatele. Pro komplexní otestování HEAppE je možné vytvořit několik testovacích scénářů, které na sebe mohou navazovat. Základním scénářem pro používání HEAppE Middleware je vytvoření a spuštění úlohy na výpočetním clusteru. Z tohoto případu užití vychází i základní scénář pro automatizované testování HEAppE prostřednictvím Robot Framework.

Důležitým krokem při užívání HEAppE Middleware je provedení autentizace. HEAppE po úspěšné autentizaci uživatele vrátí vygenerovaný unikátní identifikátor, který uživatel následně přidává do každé další žádosti při volání endpointu.

Hlavním úkolem tohoto testovacího plánu bude otestování spuštění úloh na výpočetních clusterech. Na každý cluster bude podána žádost o spuštění několika konfigurací úloh tak, aby byly otestovány všechny varianty specifikací úloh.

Součástí této práce je vytvoření testovacího plánu k otestování funkcionalit HEAppE při užívání rozšíření pro lokální spuštění úloh. Testovací plán taktéž bude testovat nově vytvořené endpointy v sekci Management, které jsou určené ke správě šablon výpočtů (Command Template).

Před spouštěním úloh budou otestovány funkcionality související s uživatelem HEAppE. Tyto kroky se nachází v následující tabulce 7.1.

Krok	Název	Endpoint
1	Získání seznamu dostupných clusterů	/heappe/ClusterInformation/ ListAvailableClusters
2	Autentizace uživatele	/heappe/UserAndLimitationManagement/ AuthenticateUserPassword
3	Získání využití zdrojů a limity uživatele HEAppE	/heappe/ UserAndLimitationManagement/ GetCurrentUsageAndLimitationsForCurrentUser
4	Získání využití zdrojů na výpočetních clusterech	/heappe/ClusterInformation/ CurrentClusterNodeUsage
5	Získání využití zdrojů uživatele HEAppE	/heappe/JobReporting/ GetUserResourceUsageReport
6	Získání využití zdrojů skupiny uživatelů HEAppE	/heappe/JobReporting/ GetUserGroupResourceUsageReport

Tabulka 7.1: Seznam kroků pro testování funkcionalit souvisejících s uživatelem HEAppE

Dále bude otestováno spouštění pěti různě specifikovaných úloh na superpočítáčových clusterech s různými typy plánovačů. Kroky určené k otestování jedné úlohy popisují kroky uvedené v tabulce 7.2.

Zmiňovaná pětice specifikací úloh obsahuje úlohy popsané v následujícím seznamu. Konfigurace těchto úloh byly popisovány výše v kapitole 4.

- Základní úloha
- Úloha využívající generickou šablonu výpočtu
- Úloha se specifikací extrémně dlouhé úlohy
- Úloha využívající funkcionality JobArrays
- Úloha s definovanou závislostí na jiné úloze

Krok	Název	Endpoint
1	Vytvoření úlohy	/heappe/JobManagement/CreateJob
2	Spuštění zpracování úlohy	/heappe/JobManagement/SubmitJob
3	Získání spotřebovaných zdrojů úlohy	/heappe/JobReporting/ GetResourceUsageReportForJob
4	Získání využití zdrojů na výpočetních clusterech	/heappe/ClusterInformation/ CurrentClusterNodeUsage
5	Získání stavu úlohy	/heappe/JobManagement/GetCurrentInfoForJob
6	Vyčkání na spuštění úlohy	/heappe/JobManagement/GetCurrentInfoForJob
7	Získání IP adres alokovaných uzlů	/heappe/JobManagement/GetAllocatedNodesIPs
8	Zrušení úlohy	/heappe/JobManagement/CancelJob
9	Vyčkání na ukončení úlohy	/heappe/JobManagement/GetCurrentInfoForJob
10	Smazání úlohy	/heappe/JobManagement/DeleteJob

Tabulka 7.2: Seznam kroků pro testování úloh HEAppE

Provedením kroků, které se nacházejí v tabulce 7.3, se otestuje funkčnost již výše popsané správy Command Template. Implementace tohoto rozšíření HEAppE o správu Command Template je taktéž součástí této práce. Dle specifikace jsou Management endpointy dostupné jen uživateli HEAppE s rolí Administrátor.

Krok	Název	Endpoint
1	Vytvoření Command Template	/heappe/Management/CreateCommandTemplate
2	Modifikace specifikace Command Template	/heappe/Management/ModifyCommandTemplate
3	Smazání Command Template	/heappe/Management/ RemoveCommandTemplate

Tabulka 7.3: Seznam kroků pro testování Management sekce HEAppE

Protože je práce s úlohou na výpočetním clusteru řízena plánovačem, prakticky se jedná o předávání pokynů. Proto se v testovacím plánu musí vyskytovat čekání na vykonání příkazu. Při pokynu pro spuštění úlohy tak musíme čekat na to, až naši úlohu spustí plánovač na výpočetním clusteru. V testovacím scénáři je toto čekání implementováno opakováním volání HEAppE REST API s prodlevou, aby nedošlo k zablokování (toto chování by mohlo nést znaky DOS/DDOS útoku). Následuje ukázka kódu testovacího scénáře Robot Frameworku obsahujícího popsané cyklické volání REST API. V tomto případě se cyklus ukončí, pokud HEAppE vrátí stav, který značí, že je úloha spuštěna. Očekává se, že se stav změní v rámci jednotek či desítek minut, Robot Framework neumožnuje práci s cykly s neznámým počtem opakování. Proto byl při implementaci využit cyklus se známým počtem opakování, horní interval cyklu byl nastaven na maximální povolenou hodnotu.

Útržek testovacího scénáře obsahuje funkci WaitForJobRuns. Úkolem této funkce je počkat na spuštění úlohy plánovačem na výpočetním clusteru.

```
WaitForJobRuns
    ${jsonstr}      Get file ../json/JobManagement/OperationJob.json
    Set Global Variable    ${jsonstr}    ${jsonstr}
    Set SessionCode
    Set JobId To SubmittedJobInfoId
    FOR    ${i}    IN RANGE    999999
        ${response}=  Post /heappe/JobManagement/GetCurrentInfoForJob  ${json}
        Set Global Variable    ${Response}    ${Response}
        Check Response Code 200
        Log    ${response}
        Exit For Loop If    ${response['body']['State']} >= 8
        Sleep    10s
    END
```

Listing 7.1: Segment Robot Framework testu

Výsledkem tohoto testování je vygenerovaný report a log. Programátor pak má lepší představu o funkčnosti jednotlivých částí systému. Tyto reporty Robot Framework připraví v grafické podobě. Dostupné jsou ve formátu html. Surová data jsou součástí reportu ve struktuře XML⁴.

Test či testový plán může být prováděn pravidelně a několikrát za sebou. Počáteční časové náklady na sestavení tohoto scénáře jsou nesrovnatelné s přínosem, které automatizované testování přináší. Test navíc může běžet v pozadí automaticky a vývojář si tak jen vyzvedne výsledek.

Tento testovací scénář je možné volně rozšiřovat nebo vytvářet scénáře obdobné. Velmi praktické je i využití Robot Frameworku pro testování softwaru v době vývoje. Scénář můžeme měnit podle aktuálních požadavků např. při vývoji nové funkcionality.

Výše popsaný testový scénář byl použit k otestování funkčnosti HEAppE Middleware s užitím lokálního spuštění úloh na hostitelském počítači. Tímto byla prokázána validita implementovaných

⁴Výměnný formát pro přenos dat

rozšíření, které popisuje tato práce. Pro uživatele je užití lokálního simulovaného clusteru prostřednictvím HEAppE takřka nerozeznatelné od použití HEAppE se skutečným výpočetním clusterem.

Navíc byly otestovány další endpointy, které je vhodnější testovat manuálně. Jedná se především o funkcionality HEAppE Middleware pro přenos dat, kdy je vhodnější provádět kontroly ručně. Výsledky těchto automatizovaně prováděných testů jsou součástí práce v kapitole příloh v sekci D.

Manuálně byly otestovány endpointy */heappe/FileTransfer/GetFileTransferMethod* a */heappe/-FileTransfer/EndFileTransferMethod* za pomocí aplikace, která je součástí reposítáře HEAppE. Tyto endpointy slouží k otevření přenosového kanálu mezi klientským počítačem a superpočítačovým clusterem. Funkcionalita tedy umožňuje přenos souborů mezi fyzickým počítačem uživatele a superpočítačovým clusterem. Při tomto testování byl otevřen přenosový kanál, pomocí protokolu SCP⁵ byl na cluster přesunut soubor, a následně byl přenosový kanál uzavřen. Výsledek tohoto testu se nachází v kapitole příloh v sekci E.

⁵Protokol sloužící k bezpečnému přenosu mezi dvěma počítači.

Kapitola 8

Závěr

Při návrhu a implementaci podpory simulovaného plánovače spouštěného na lokálním stroji uživatele byla vytvořena konfigurace virtuálního stroje, jehož cílem je simulace chování superpočítáčového clusteru s plánovačem úloh. Důraz byl kladen také na co možná nejvhodnější simulaci chování a komunikaci plánovače na lokálním virtuálním clusteru. Virtuální stroj byl navržen tak, aby jej uživatel mohl jednoduše upravit či zaměnit za své řešení. Musí přitom však dodržovat předepsané podmínky pro komunikaci HEAppE s lokálním simulovaným výpočetním clusterem.

Cílem tohoto rozšíření je přinést uživatelům možnost vyzkoušet si HEAppE Middleware na svém počítači bez nutnosti napojení na fyzický superpočítáč. Proto i komunikace s lokálním simulovaným výpočetním clusterem probíhá stejně, jako je tomu u skutečných výpočetních clusterů. Využívá se stejných protokolů, metod šifrování nebo postupů při autentizaci. To vše proto, aby byl následný přechod z lokálně simulovaného prostředí na skutečné superpočítáče co nejsnazší a přitom byly dodrženy veškeré nároky na bezpečnostní politiky superpočítáčových center.

Dále bylo navrženo a implementováno rozšíření o funkcionality pro vytváření uživatelsky definovaných šablon. Prerekvizitou je existence generické šablony, která slouží především k testovacím účelům. Z této generické šablony se pak vychází při definování uživatelské šablony, která je následně uložena persistently do systému HEAppE, a koncový uživatel ji může libovolně využívat.

Všechny nově přidané funkcionality byly důkladně otestovány. Současný stav projektu s těmito změnami byl publikován do nové verze HEAppE Middleware. Verze nese označení 3.0.0 a je dostupná na veřejném GIT repositáři projektu HEAppE. Současně byl vytvořen testovací scénář pro automatizované testování, který je možné spouštět při každé publikaci změny do repositáře projektu.

Věřím, že díky výše popsaným funkcionalitám bude dále systém HEAppE Middleware rozširován do superpočítáčových center, především díky rozšíření o možnost lokálního spouštění úloh. Správci HPC center si budou moci HEAppE Middleware vyzkoušet lokálně na počítači před nasazením systému do produkčního prostředí. Po zhodnocení bezpečnostních nároků a přínosu HEAppE Middleware pro uživatele daného superpočítáčového centra může být nasazena instance této služby již s přístupem na skutečné výpočetní clustery.

Literatura

1. *SEYMOUR CRAY: The Supercomputer* [online] [cit. 2022-03-01]. Dostupné z: <https://www.lemelson.mit.edu/resources/seymour-cray>.
2. TRONNER, Pavel. *Seymour Cray: Stavitel superpočítačů* [online] [cit. 2022-03-01]. Dostupné z: <https://www.zive.cz/clanky/seymour-cray-stavitel-superpocitacu/sc-3-a-178330/default.aspx>.
3. HOSCH, William L. *Supercomputer* [online] [cit. 2022-03-01]. Dostupné z: <https://www.britannica.com/technology/supercomputer>.
4. *NOVEMBER 2021* [online] [cit. 2022-03-01]. Dostupné z: <https://www.top500.org/lists/top500/2021/11/>.
5. *KAROLINA* [online] [cit. 2022-03-01]. Dostupné z: <https://www.it4i.cz/infrastruktura/karolina>.
6. *TOP500 LIST - NOVEMBER 2021* [online] [cit. 2022-03-01]. Dostupné z: <https://www.top500.org/lists/top500/list/2021/11/>.
7. *Scheduling Basics* [online] [cit. 2022-03-01]. Dostupné z: https://hpc-wiki.info/hpc/Scheduling_Basics.
8. FAN, Yuping. Job Scheduling in High Performance Computing: Illinois Institute of Technology. In: [online]. [B.r.] [cit. 2022-03-04]. Dostupné z: <https://arxiv.org/pdf/2109.09269.pdf>.
9. GÓMEZ-MARTÍN, César; VEGA-RODRÍGUEZ, Miguel A.; GONZÁLEZ-SÁNCHEZ, José-Luis. Fattened backfilling: An improved strategy for job scheduling in parallel systems. *Journal of Parallel and Distributed Computing* [online]. 2016, s. 69–77 [cit. 2022-03-04]. ISSN 0743-7315. Dostupné z DOI: [10.1016/j.jpdc.2016.06.013](https://doi.org/10.1016/j.jpdc.2016.06.013).
10. RAND, David. *HPC as a service: High-performance computing when you need it* [online] [cit. 2022-03-10]. Dostupné z: <https://www.hpe.com/us/en/insights/articles/hpc-as-a-service-high-performance-computing-2012.html>.
11. WIGGERS, Kyle. *What is HPC-as-a-service and why does it matter?* [Online] [cit. 2022-03-10]. Dostupné z: <https://venturebeat.com/2022/01/20/what-is-hpc-as-a-service-and-why-does-it-matter/>.

12. *HEAPPE MIDDLEWARE: About HEAppE* [online] [cit. 2022-03-01]. Dostupné z: https://heappe.eu/web/about_heappe/.
13. HPE GreenLake for High Performance Computing brochure. In: [online]. [B.r.] [cit. 2022-03-15]. Dostupné z: <https://assets.ext.hpe.com/is/content/hpedam/documents/a50001000-1999/a50001688/a50001688enw.pdf>.
14. SHOWALTER, Steve. *HPE GreenLake Central* [online] [cit. 2022-03-15]. Dostupné z: <https://developer.hpe.com/blog/hpe-greenlake-for-private-cloud/>.
15. *Extreme Factory: Innovation at your fingertips* [online] [cit. 2022-03-15]. Dostupné z: <https://atos.net/en/solutions/high-performance-computing-hpc/bull-extreme-factory>.
16. GRANT, Andy. Delivering on the promise of hybrid multi cloud for HPC [online]. [B.r.] [cit. 2022-03-15]. Dostupné z: https://atos.net/wp-content/uploads/2021/10/Hybrid_multi_cloud_HPC_ATOS.pdf.
17. *Json* [online] [cit. 2022-03-01]. Dostupné z: <https://www.json.org/json-en.html>.
18. *Docker overview* [online] [cit. 2022-03-04]. Dostupné z: <https://docs.docker.com/get-started/overview/>.
19. *How to submit a job using qsub* [online] [cit. 2022-03-01]. Dostupné z: https://bioinformatics.mdc-berlin.de/intro2UnixandSGE/sun_grid_engine_for_beginners/how_to_submit_a_job_using_qsub.html.
20. *Denial of service* [online] [cit. 2022-03-02]. Dostupné z: https://cs.wikipedia.org/wiki/Denial_of_service.
21. *Deadlock* [online] [cit. 2022-03-01]. Dostupné z: https://cs.wikipedia.org/wiki/Deadlock%5C#/media/Soubor:Process_deadlock.svg.
22. PADUA, David; SCHULZ, Martin. Checkpointing. In: *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011, s. 264–273. ISBN 978-0-387-09766-4. Dostupné z DOI: 10.1007/978-0-387-09766-4_62.
23. *Co je to parser?* [Online] [cit. 2022-03-01]. Dostupné z: <https://it-slovník.cz/pojem/parser>.
24. *Jq* [online] [cit. 2022-01-12]. Dostupné z: <https://stedolan.github.io/jq/>.
25. *Sep 9, 1947 CE: World's First Computer Bug* [online] [cit. 2022-03-01]. Dostupné z: <https://www.nationalgeographic.org>thisday/sep9/worlds-first-computer-bug/>.
26. PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80-722-6636-5.
27. *Spiral model: Spiral model (Boehm, 1988)* [online] [cit. 2022-03-01]. Dostupné z: [https://commons.wikimedia.org/wiki/File:Spiral_model_\(Boehm,_1988\).svg](https://commons.wikimedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg).

28. MITCHELL, Jamie L.; BLACK, Rex; BARABAS, Michael. *Advanced Software Testing - Vol. 3*. 2nd Edition. USA: Rocky Nook, 2015. ISBN 9781457189104.
29. *Robot Framework* [online] [cit. 2022-01-19]. Dostupné z: <https://robotframework.org/>.
30. *Smoke test* [online] [cit. 2022-03-01]. Dostupné z: https://cs.wikipedia.org/wiki/Smoke_test.
31. *Continuous Integration with Bamboo* [online] [cit. 2022-03-04]. Dostupné z: <https://www.automation-consultants.com/continuous-integration-2/>.

Příloha A

Příklad jednotlivých segmentů specifikace pro lokální HPC

```
"Clusters": [
    {
        "Id": 1,
        "Name": "{LocalComputingClusterName}",
        "Description": "{ClusterDescription}",
        "MasterNodeName": "host.docker.internal",
        "DomainName": "localhost",
        "Port": 49500,
        "LocalBasepath": "/home/heappeclient/heappetests",
        "TimeZone": "CET",
        "SchedulerType": 1,
        "ConnectionProtocol": 2,
        "ServiceAccountCredentialsId": 1,
        "UpdateJobStateByServiceAccount": true
    }
]
...
"ClusterAuthenticationCredentials": [
    {
        "Id": 1,
        "Username": "{UserName}",
        "Password": null,
        "PrivateKeyFile": "{PrivateKeyFile}",
        "PrivateKeyPassword": "{PrivateKeyPassword}",
        "ClusterId": 1
    },
    ...
]
```

```

{
    "Id": 2,
    "Username": "{UserName}",
    "Password": null,
    "PrivateKeyFile": "{PrivateKeyFile}",
    "PrivateKeyPassword": "{PrivateKeyPassword}",
    "ClusterId": 1
}
]

...
```
"ClusterNodeTypes": [
{
 "Id": 1,
 "Name": "{Name}",
 "Description": "{Description}",
 "NumberOfNodes": "{NumberOfNodes}",
 "CoresPerNode": "{CoresPerNode}",
 "Queue": "{Queue}",
 "RequestedNodeGroups": null,
 "MaxWalltime": "{MaxWalltime}",
 "ClusterId": 1,
 "FileTransferMethodId": 3,
 "JobTemplateId": 1,
 "TaskTemplateId": 1,
 "ClusterAllocationName": null
}
]

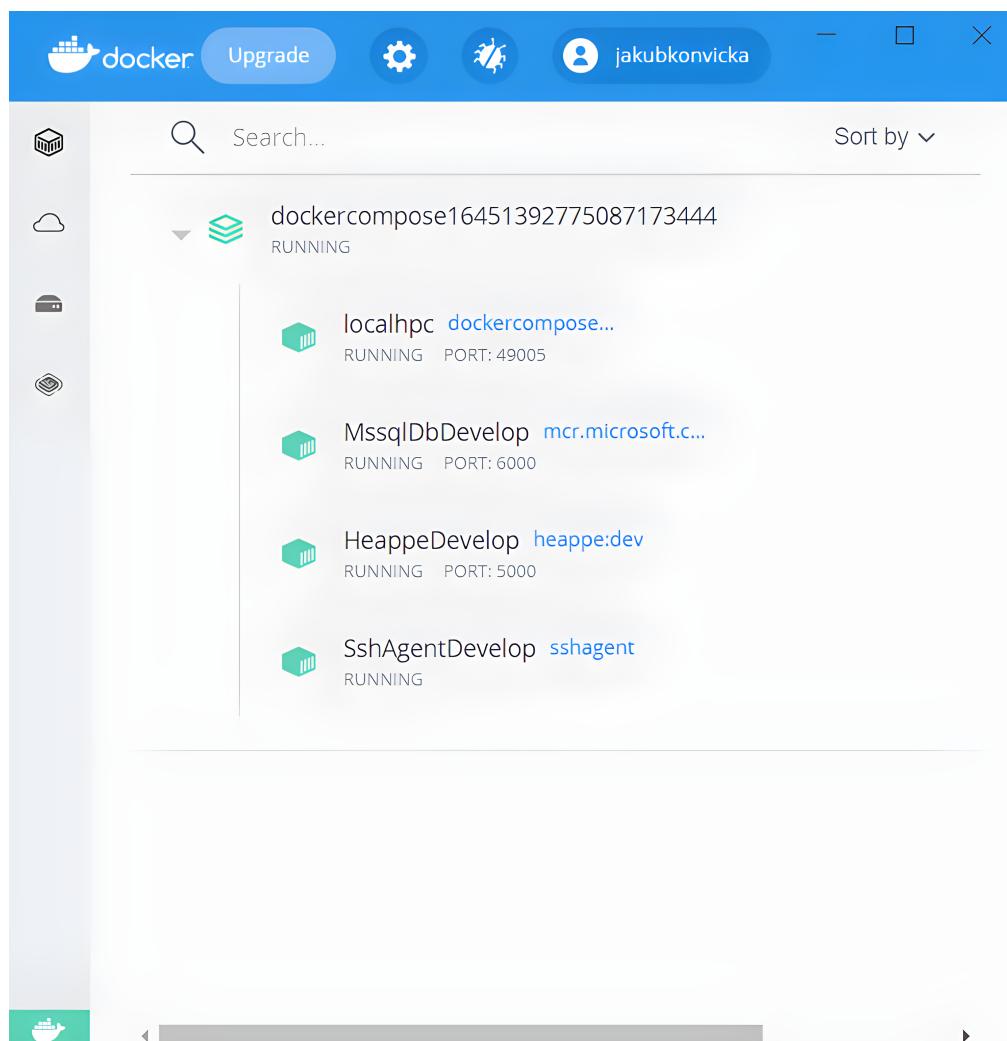
```

---

Listing A.1: Struktura segmentů konfigurace pro lokální HPC cluster

## Příloha B

### Spuštěný kontejner HEAppE



Obrázek B.1: Ukázka aplikace Docker Desktop se spuštěným kontejnerem

## Příloha C

# Příklad specifikace testovacího plánu pro Robot Framework

---

```
*** Settings ***
Resource keywords.resource
Default Tags positive

*** Test Cases ***
Login User with Password
 Connect to Server
 Login User ironman password
 Verify Valid Login Tony Stark
 [Teardown] Close Server Connection

Denied Login with Wrong Password
 [Tags] negative
 Connect to Server
 Run Keyword And Expect Error * Invalid Password Login User ironman
 drowssap
 Verify Unauthorised Access
 [Teardown] Close Server Connection
```

---

Listing C.1: Specifikace Robot Framework testu

## Příloha D

# Robot Framework - výstup testu

### completeHeappeTest LOCALCOMPUTING Report

Generated  
20220305 10:37:38 UTC+01:00

#### Summary Information

|               |                       |
|---------------|-----------------------|
| Status:       | All tests passed      |
| Start Time:   | 20220305 10:35:22.907 |
| End Time:     | 20220305 10:37:38.758 |
| Elapsed Time: | 00:02:15.851          |
| Log File:     | log.html              |

#### Test Statistics

| Total Statistics                  | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                        |
|-----------------------------------|-------|------|------|------|----------|---------------------------------------------------------------------------|
| All Tests                         | 14    | 14   | 0    | 0    | 00:02:15 | <div style="width: 100%; background-color: #2e7131; height: 10px;"></div> |
| Statistics by Tag                 |       |      |      |      |          |                                                                           |
| No Tags                           |       |      |      |      |          | <div style="width: 0%; background-color: #cccccc; height: 10px;"></div>   |
| Statistics by Suite               | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                        |
| completeHeappeTest LOCALCOMPUTING | 14    | 14   | 0    | 0    | 00:02:16 | <div style="width: 100%; background-color: #2e7131; height: 10px;"></div> |

Obrázek D.1: Report Robot Frameworku

# completeHeappeTest LOCALCOMPUTING Log

Generated  
20220305 10:37:38 UTC+01:00

## Test Statistics

| Total Statistics                  | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                        |
|-----------------------------------|-------|------|------|------|----------|---------------------------------------------------------------------------|
| All Tests                         | 14    | 14   | 0    | 0    | 00:02:15 | <div style="width: 100%; background-color: #2e7131; height: 10px;"></div> |
| Statistics by Tag                 | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                        |
| No Tags                           |       |      |      |      |          | <div style="width: 0%; background-color: #cccccc; height: 10px;"></div>   |
| Statistics by Suite               | Total | Pass | Fail | Skip | Elapsed  | Pass / Fail / Skip                                                        |
| completeHeappeTest LOCALCOMPUTING | 14    | 14   | 0    | 0    | 00:02:16 | <div style="width: 100%; background-color: #2e7131; height: 10px;"></div> |

## Test Execution Log

|                                                                                               |              |
|-----------------------------------------------------------------------------------------------|--------------|
| - <b>SUITE</b> completeHeappeTest LOCALCOMPUTING                                              | 00:02:15.851 |
| <b>Full Name:</b> completeHeappeTest LOCALCOMPUTING                                           |              |
| <b>Source:</b> C:\GIT\ApiTests\robotTests\completeHeappeTest_LOCALCOMPUTING.robot             |              |
| <b>Start / End / Elapsed:</b> 20220305 10:35:22.907 / 20220305 10:37:38.758 / 00:02:15.851    |              |
| <b>Status:</b> 14 tests total, 14 passed, 0 failed, 0 skipped                                 |              |
| + <b>TEST</b> /heappe/ClusterInformation/ListAvailableClusters                                | 00:00:00.092 |
| + <b>TEST</b> /heappe/UserAndLimitationManagement/AuthenticateUserPassword                    | 00:00:00.029 |
| + <b>TEST</b> /heappe/UserAndLimitationManagement/GetCurrentUsageAndLimitationsForCurrentUser | 00:00:00.084 |
| + <b>TEST</b> /heappe/ClusterInformation/CurrentClusterNodeUsage                              | 00:00:00.202 |
| + <b>TEST</b> /heappe/JobReporting/ GetUserResourceUsageReport                                | 00:00:00.119 |
| + <b>TEST</b> /heappe/JobReporting/ GetUserGroupResourceUsageReport                           | 00:00:00.122 |
| + <b>TEST</b> /heappe/JobManagement/CreateJobSIMPLE                                           | 00:00:20.860 |
| + <b>TEST</b> /heappe/JobManagement/CreateJobGENERIC                                          | 00:00:30.921 |
| + <b>TEST</b> /heappe/JobManagement/CreateJobEXTRA-LONG                                       | 00:00:20.840 |
| + <b>TEST</b> /heappe/JobManagement/CreateJobJOBARRAYS                                        | 00:00:30.949 |
| + <b>TEST</b> /heappe/JobManagement/CreateJobDEPENDENCY                                       | 00:00:31.038 |
| + <b>TEST</b> /heappe/Management/CreateCommandTemplate                                        | 00:00:00.049 |
| + <b>TEST</b> /heappe/Management/ModifyCommandTemplate                                        | 00:00:00.048 |
| + <b>TEST</b> /heappe/Management/RemoveCommandTemplate                                        | 00:00:00.032 |

Obrázek D.2: Log Robot Frameworku

## Příloha E

### Výstup testování přenosu souborů

---

```
JobSpecification created
JobSpecificationExt(name=ASBTestJob; project=ExpTests; waitingLimit=0; walltimeLimit=;
 notificationEmail=; phoneNumber=; notifyOnAbort=; notifyOnFinish=; notifyOnStart=;
 clusterId=3; fileTransferMethodId=3; environmentVariables=HEAppE.ExtModels.
 JobManagement.Models.EnvironmentVariableExt[]; tasks=HEAppE.ExtModels.JobManagement.
 Models.TaskSpecificationExt[])
Job 21 created
SubmittedJobInfoExt(id=21; name=ASBTestJob; state=Configuring; project=ExpTests;
 creationTime=07.03.2022 12:20:08; submitTime=; startTime=; endTime=;
 totalAllocatedTime=; tasks=HEAppE.ExtModels.JobManagement.Models.SubmittedTaskInfoExt
[])
All files uploaded
Job submitted
Uploading file: a.txt
File uploaded: a.txt
Job 21 state: Running
Canceling job 21 ...
Job 21 was canceled
```

---

Listing E.1: Log aplikace provádějící test HEAppE funkcionalit

---

```
INFO 2022-03-07 13:20:08 HEAppE.HpcConnectionFramework.SystemCommands.LinuxCommands -
 Create job directory result: "Created directory /home/heappeclient/heappetests/21.
 Created directory /home/heappeclient/heappetests/21/24."
INFO 2022-03-07 13:20:08 HEAppE.BusinessLogicTier.Logic.FileTransfer.FileTransferLogic -
 Getting file transfer method for submitted job info ID 21 with user testuser
INFO 2022-03-07 13:20:09 HEAppE.BusinessLogicTier.Logic.JobManagement.JobManagementLogic
 - User testuser is submitting the job with info Id 21
INFO 2022-03-07 13:20:09 HEAppE.HpcConnectionFramework.SchedulerAdapters.Generic.
 LinuxLocal.LinuxLocalSchedulerAdapter - Run prepare-job result:
INFO 2022-03-07 13:20:09 HEAppE.HpcConnectionFramework.SystemCommands.LinuxCommands -
 Allow file transfer result: "Public key inserted."
INFO 2022-03-07 13:20:09 HEAppE.BusinessLogicTier.Logic.FileTransfer.FileTransferLogic -
 Removing file transfer method for submitted job info ID 21 with user testuser
INFO 2022-03-07 13:20:09 HEAppE.HpcConnectionFramework.SystemCommands.LinuxCommands -
 Remove permission for direct file transfer result: "Public key removed for user
heappeclient."
```

---

Listing E.2: Log HEAppE Middleware - FileTransfer

---

```
/home/heappeclient/heappetests/21/24$ ls -la | grep a.txt
-rw-r--r-- 1 heappeclient root 16 Sep 9 2020 a.txt
```

---

Listing E.3: Výlistování adresáře na lokálním HPC clusteru s výběrem souboru a.txt