Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Riadenie získavania experimentálnych údajov na účely vizualizácie

Diplomová práca

2022 **Bc. Dominik Matis**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Riadenie získavania experimentálnych údajov na účely vizualizácie

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	RNDr. Martin Vaľa, PhD.
Konzultant:	

Košice 2022 **Bc. Dominik Matis**

Abstrakt v SJ

Táto diplomová práca sa skladá z implementácie knižnice pre webovú aplikáciu a prostredie virtuálnej reality, implementácie aplikačného programového rozhrania pre spúšťanie úloh a implementácie sprostredkovateľa pre dávkovacie systémy. Knižnica má slúžiť pre používateľov na jednoduchšiu prácu s klastrami a dávkovacími systémami, pričom nie je potrebné používať prostredie príkazového riadku, ale napríklad webovú aplikáciu, či virtuálnu realitu. Úvodná časť práce sa zaoberá analýzou viacerých dávkovacích systémov, opisom fyziky vysokých energií, opisom technológií, výberom technológie pre rozhranie na spúšťanie úloh a tiež opisom infraštruktúry GRID. Nasledujúca časť sa zaoberá implementáciou knižnice, implementáciou rozhrania a tiež implementáciou sprostredkovateľa pre dávkovací systém. Záverečnou časťou je vyhodnotenie, ktoré obsahuje požiadavky a zisťuje, či boli splnené. Tiež je tu opísaný spôsob testovania a opis predností projektu.

Kľúčové slová v SJ

dávkovacie systémy, knižnica, webové rozhranie, virtuálna realita

Abstrakt v AJ

This thesis consists of the implementation of a library for a web application and a virtual reality environment, the implementation of an application programming interface for job launching, and the implementation of a broker for batching systems. The library is intended to make it easier for users to work with clusters and batching systems without the need to use a command line environment, but for example a web application or a virtual reality environment. The introductory part of the thesis deals with the analysis of several batching systems, the description of high energy physics, the description of technologies, the selection of the technology for the interface to run the jobs and also the description of the GRID infrastructure. The following section deals with the implementation of the library, the implementation of the interface and also the implementation of the broker for the batching system. The final part is the evaluation, which contains the requirements and finds out whether these have been met. It also describes the method of testing and describes the strengths of the project.

Kľúčové slová v AJ

batch systems, library, web interface, virtual reality

Bibliografická citácia

MATIS, Dominik. *Riadenie získavania experimentálnych údajov na účely vizualizácie*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2022. 70s. Vedúci práce: RNDr. Martin Vaľa, PhD.

64302

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Riadenie získavania experimentálnych údajov na účely vizualizácie

Experimental Data Acquisition Management for Visualization Purposes

Študent: **Bc. Dominik Matis**
Školiteľ: **RNDr. Martin Vaľa, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:
1. Analyzovať súčasné prístupy k riadeniu získavania údajov z fyzikálnych experimentov na účely ich vizualizácie.
2. Analyzovať súčasný stav webového softvérového systému SALSA pre takéto riadenie získavania údajov.
3. Na základe analýzy navrhnúť a implementovať vybrané rozšírenia systému SALSA.
4. Implementované riešenie overiť na vizualizačných úlohách využívajúcich údaje z reálnych fyzikálnych experimentov.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce a štandardov školiaceho pracoviska.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 22.04.2022
Dátum zadania diplomovej práce: 29.10.2021

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 22.4.2022 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce.

Rovnako by som sa rád poďakoval svojim rodičom, priateľke a kamarátom za ich podporu a povzbudzovanie počas celého môjho štúdia.

Obsah

Zoznam obrázkov

Úvod

V dnešnej dobe sa praktizuje veľké množstvo pokusov, ktoré sa týkajú časticovej fyziky, či fyziky vysokých energií. Pri vykonávaní týchto pokusov vznikne obrovské množstvo údajov, ktoré obsahujú aj niektoré užitočné dáta, takže je potrebné všetky tieto údaje rozanalyzovať a nájsť tie dôležité. Týchto údajov je také množstvo, že ak by človek chcel použiť osobný počítač alebo notebook pre analýzu, trvalo by mu veľmi veľa rokov, kým by sa mu podarilo zanalyzovať všetky údaje. Je to z dôvodu, že osobné počítače v dnešnej dobe poskytujú obrovský výkon, no v porovnaní s profesionálnymi počítačovými centrami slúžiacimi výlučne na prácu s vedeckými údajmi, je to skutočne zanedbateľný výkon. Vo výskumných organizáciách sa nachádzajú výkonné počítačové centrá, pričom tieto je potrebné istým spôsobom organizovať. Na tieto účely slúžia práve dávkovacie systémy. Úlohou dávkovacích systémov je zabezpečiť správu úloh a práce v rámci centra, pričom je potrebné sledovať, či sú všetky úlohy vykonávané, tiež je potrebné zbierať čiastkové a celkové výsledky z týchto úloh. Tu nastáva ďalší problém a to ukladanie týchto medzivýsledkov a výsledkov. Každé počítačové centrum na svete musí zabezpečiť, aby obsahovalo úložný priestor. Vo chvíli, keď počítačové centrum obsahuje výpočtové zdroje a úložné zdroje, je potrebné ich nejakým spôsobom prepojiť. Tu prichádza na rad tzv. gridovský systém, resp. gridovská infraštruktúra. Táto zabezpečí správne prepojenie týchto dvoch častí. Nič na svete nie je stopercentné a to isté platí aj pri veľkých počítačových centrách. Niekedy sa totiž môže stať, že niektorá úloha zahltí celý výpočtový klaster, či už z pohľadu sieťovej komunikácie, využitia CPU alebo vstupno-výstupnej operácie, ako zápisy na disk, resp. čítania z disku. Aby sa zamedzilo alebo aspoň obmedzilo vzniku takýchto situácií, používajú sa rôzne monitorovacie služby či systémy. Tieto monitoringy majú zväčša podobu webovej aplikácie, no nájdu sa tiež riešenia, ktoré pracujú výlučne v prostredí príkazového riadku.

Mnoho ľudí v dnešnej spoločnosti si určite všimlo rapídny vzostup informačných technológií, najmä v oblasti VR (*Virtual Reality*) a AR (*Augmented Reality*). Veľké množstvo spoločností začína vývoj v tejto oblasti, keďže existujú podozre-

nia, že práve odvetvie virtuálnej reality bude čoraz viac využívané v reálnom svete. V oblasti časticovej fyziky a fyziky vysokých energií nevidíme použitie virtuálnej reality a myslíme si, že by bolo vhodné vytvoriť práve takýto systém, ktorý by umožňoval pracovníkom ovládať výpočtové klastre pomocou virtuálnej reality. Vytvorenie takéhoto prepojenia si vyžaduje vybudovanie prostredia pre virtuálnu realitu a spôsob komunikácie medzi týmto prostredím a samotnými výpočtovými klastrami.

Formulácia úlohy

Ako už bolo spomenuté v úvode, pre vytvorenie prostredia virtuálnej reality pre manipuláciu s klastrami, je potrebné vybudovať prostredie a spôsob komunikácie. Táto diplomová práca má za úlohu práve túto druhú úlohu, a teda vytvoriť spôsob, akým môže používateľ vo virtuálnej realite komunikovať s výpočtovým klastrom. Okrem tejto úlohy sme sa rozhodli pripraviť taktiež knižnicu, ktorá môže byť vložená napríklad do nástroja Cockpit, ktorý slúži na správu Linux serverov pomocou webového prehliadača. Táto knižnica bude slúžiť na začatie vykonávania rôznych úloh na klastroch a tiež bude slúžiť na získavanie informácií z týchto klastrov. Výhodou bude možnosť monitorovať viacero typov klastrov, teda rôzne dávkovacie systémy za pomoci jedinej aplikácie.

Ďalšou úlohou v rámci tejto diplomovej práce je pripraviť server, ktorý bude schopný spúšťať zadané úlohy z HTTP (*HyperText Transfer Protocol*) požiadavky vytvorenej používateľom. Tento server bude potrebné vytvoriť tak, aby bolo možné spúšťať úlohy na rôznych dávkovacích systémoch. V tomto prípade to budú dávkovacie systémy SALSA a SLURM. Pri tvorbe tohto serveru bude potrebné taktiež otestovať viacero možností, ktoré ako vývojári máme pri zvolení jazyka a knižnice, či rámca na tvorbu HTTP servera.

Okrem tohto HTTP API (*Application Programming Interface*) servera bude potrebné tiež vytvoriť sprostredkovateľa pre dávkovací systém SLURM, keďže tento primárne poskytuje informácie len cez príkazový riadok. Našťastie, však existuje dokumentácia pre tento systém, ktorý obsahuje informácie o REST API, ktorý je možné použiť.

1 Analytická časť

1.1 Úvod do analýzy dát vo fyzike vysokých energií

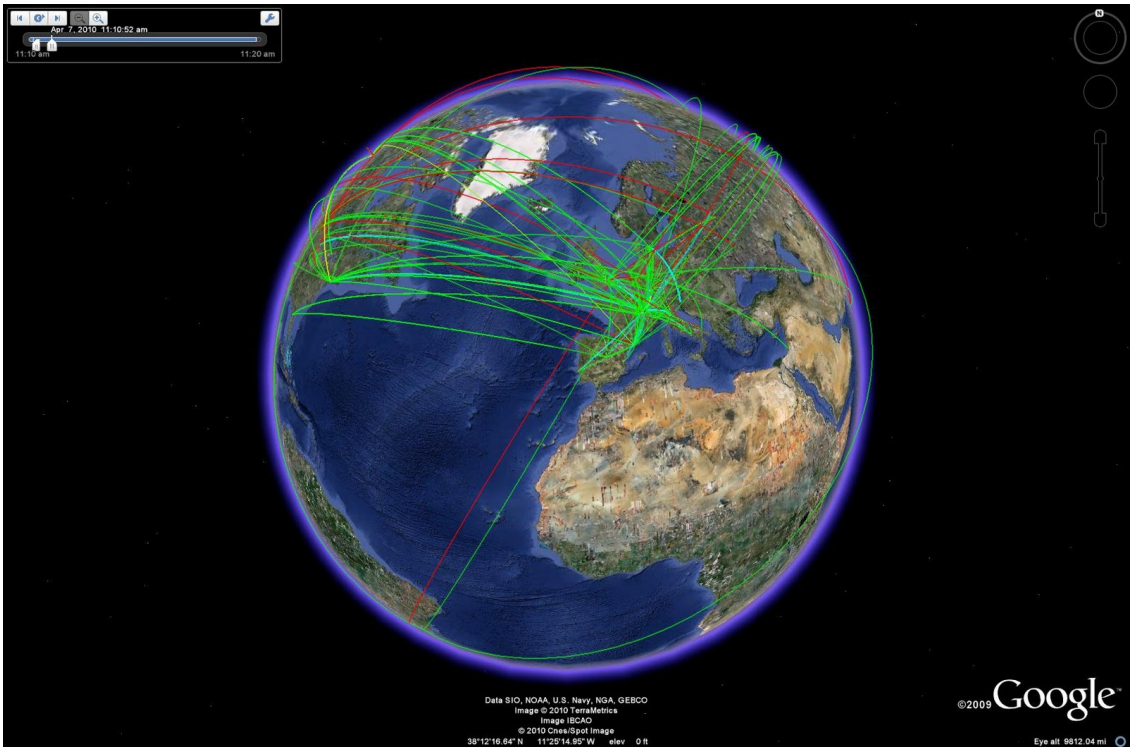
Každá organizácia zaoberajúca sa výskumom potrebuje istý spôsob, ako pracovať s veľkým množstvom dát, to platí najmä pri časticovej fyzike. Fyzika vysokých energií [1], skr. HEP (*High Energy Physics*), je potrebná kvôli tomu, že časticová fyzika sa venuje elementárnym zložkám hmoty. Tieto zložky sú nazývané elementárne kvôli tomu, že nemajú známu štruktúru, teda vyzerajú ako bod. Na zistenie štruktúry je potrebné zaistiť, aby boli dva body od seba, pričom lúče s vysokou hybnosťou majú krátku vlnovú dĺžku a môžu mať vysoké rozlíšenie. Pri vysokej hodnote hybnosti (napr. 10 GeV), ktorá je ľahko dosiahnuteľná pomocou súčasných urýchľovacích lúčov, poskytuje priestorové rozlíšenie o hodnote 10^{-16} m, čo je 10-krát menšie ako známy polomer náboja a hmotnosti protónu. Ďalším dôvodom vzniku fyziky vysokých energií je fakt, že v experimentálnej časticovej fyzike je mnoho elementárnych častíc, ktoré sú extrémne ťažké a energia na ich vytvorenie je korešpondujúco vysoká. Keďže klasické SI jednotky sú pre HEP veľmi veľké, používajú sa napr. *femtometre* miesto *metrov*. (1 femtometer = 1^{-15} m).

Jedno z najvýznamnejších centier je CERN [2], laboratórium založené v roku 1954, nachádzajúce sa na francúzsko-švajčiarskej hranici neďaleko mesta Ženeva. Bolo jedným z prvých európskych spoločných podnikov a v súčasnosti je jej členmi 23 krajín. Fyzici a inžinieri v tomto laboratóriu používajú najväčšie a najzložitejšie vedecké prístroje na svete na štúdium elementárnych zložiek hmoty. Tieto prístroje sú špeciálne skonštruované urýchľovače a detektory častíc. Urýchľovače urýchľujú zväzky častíc na vysoké energie a potom sa zväzky zrážajú navzájom alebo so stacionárnymi cieľmi. Detektory pozorujú a zaznamenávajú výsledky týchto zrážok. V roku 1951 bola na zasadnutí UNESCO prijatá prvá rezolúcia o zriadení Európskej rady pre jadrový výskum (po fran. *Conseil Européen pour la Recherche Nucléaire* z čoho vznikla skratka CERN). Hlavnou oblasťou výskumu laboratória CERN je časticová fyzika. Z tohto dôvodu sa laboratórium, ktoré prevádzkuje CERN, často označuje ako Európske laboratórium pre časticovú fyziku.

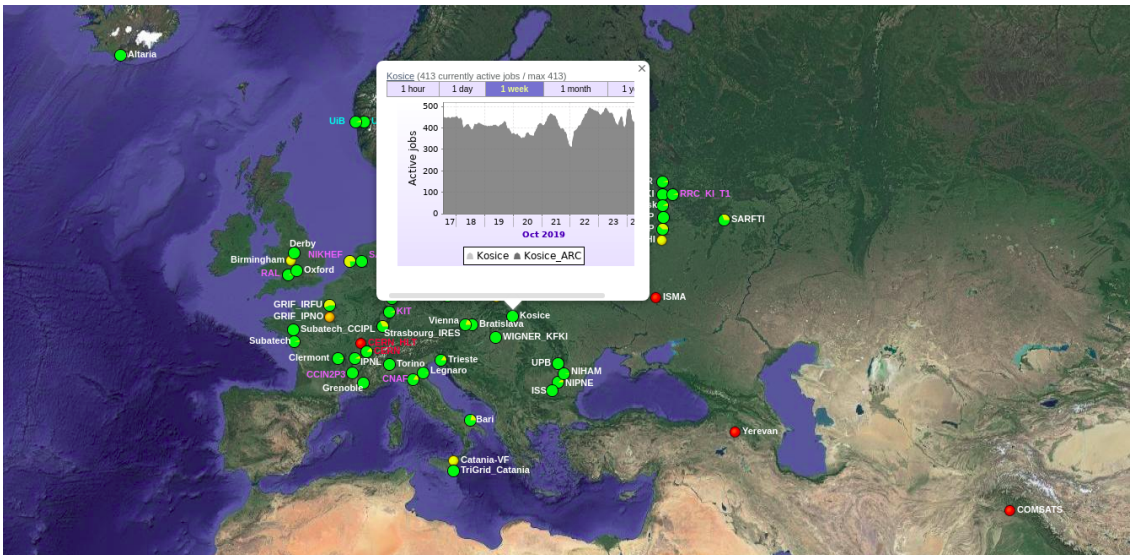
Druhým veľkým inštitútom je JINR [3], medzinárodná medzivládna organizácia založená na základe dohovoru podpísaného v roku 1956 jedenástimi zakladajúcimi štátmi a zaregistrovaná v Organizácii Spojených národov v roku 1957. JINR je skratkou pre Spoločný inštitút pre jadrový výskum (angl. *Joint Institute for Nuclear Research*), nachádza sa v meste Dubna v Moskovskej oblasti v Ruskej federácii. JINR je svetoznáme vedecké centrum, ktoré je jedinečným príkladom integrácie základného teoretického a experimentálneho výskumu s vývojom, aplikáciou špičkových technológií a univerzitného vzdelávania. JINR má v súčasnosti 19 členských štátov, pričom patria medzi ne krajiny ako Azerbajdžan, Arménsko, Bielorusko, Kórejská republika, Mongolsko, Poľsko či Kuba, a dokonca medzi ne patrí aj Slovensko. Hlavnými oblasťami teoretického a experimentálneho výskumu v JINR sú časticová fyzika, jadrová fyzika a fyzika kondenzovaných látok. Výskumný program JINR je zameraný na získanie veľmi významných výsledkov vedeckej hodnoty. Zároveň však experimentálna základňa JINR poskytuje možnosti na vykonávanie nielen pokročilého základného výskumu, ale aj aplikovaných štúdií v oblasti fyziky kondenzovaných látok, biológie, medicíny, materiálových vied, geofyziky a technickej diagnostiky, ktoré sú zamerané na výskum štruktúry a vlastností nanosystémov a nových materiálov, biologických objektov a na vývoj a rozvoj nových elektronických, bio- a informačných technológií. Ako sa ďalej spomína, JINR obsahuje sedem laboratórií, Vekslerovo a Baldinovo laboratórium fyziky vysokých energií, Dželepovovo laboratórium jadrových problémov, Bogoliubovovo laboratórium teoretickej fyziky, Frankovo laboratórium neutrónovej fyziky, Flerovovo laboratórium jadrových reakcií a laboratórium radiačnej biológie. Inštitút zamestnáva vyše 4000 ľudí.

V 90. rokoch 20. storočia bol objavený termín Grid [4], pričom tento označoval technológie, ktoré umožňovali používateľom získať výpočtový výkon na požiadanie. Úlohou infraštruktúry GRID, zobrazenej na Obr. 1.1, je spojiť rôzne systémy, akými sú napríklad dávkovacie a úložné systémy. V rámci Európy je veľké množstvo miest, kde sa nachádzajú klastre, ktoré obsahujú GRID implementáciu. Rozmiestnenie týchto klastrov je možné vidieť na Obr. 1.2. Z mnohých miest v Európe sa niekoľko klastrov nachádza aj na Slovensku, pričom jeden je dokonca aj v Košiciach. Tento klaster je na Obr. 1.2 vyznačený.

GRID, ako už bolo spomennuté, spája viaceré druhy systémov. Medzi dávkovacími systémami sú známe rôzne systémy, pričom niektoré budú spomenuté ďalej v tejto práci. Experiment ALICE v organizácii CERN nachádzajúcej sa vo Švajčiarsku napríklad využíva sadu nástrojov nazývanú AliEn, pričom AliEn nie je len dávkovací systém, no je to sada rôznych typov služieb a tiež nástrojov, pri-



Obr. 1.1: Prepojenie GRID



Obr. 1.2: Klastre v Európe

čom tieto spolu zabezpečujú správne fungovanie infraštruktúry GRID.

Keďže experimenty zvyčajne generujú veľké množstvo dát, tieto je potrebné niekde ukladať. Práve na to slúži druhý typ systémov v rámci infraštruktúry, tými sú úložné systémy. Rovnako ako v prípade dávkovacích systémov, aj úložných systémov existuje veľké množstvo. Spomenutý experiment ALICE vo švajčiarskej organizácii CERN využíva popri sade nástrojov AliEn úložný systém nazývaný EOS.

Úložný systém EOS [5] je distribuovaný úložný systém, zložený iba z diskov, ktoré sú schopné podpory rôznych protokolov na komunikáciu. Tento systém je vyvíjaný priamo v tejto organizácii a okrem experimentu ALICE je používaný na ďalšie experimenty, medzi ktorými sú napr. LHC, LHCB alebo ATLAS. Keďže najdôležitejšia je dostupnosť dát, bolo potrebné vytvoriť duplicitu dát, a to za pomoci duplikácie a distribúcie medzi hlavné švajčiarske a maďarské počítačové stredisko. Obr. 1.3 a Obr. 1.4 zobrazujú úložný priestor v rámci košického klastra, resp. klastrov používaných v rámci ALICE.

	Kosice,CERN																						
	AliEn SE		Catalogue statistics						Storage-provided information						Functional tests					Last day add tests		Demotion	IPv6
SE Name	AliEn name	Tier	Size	Used	Free	Usage	No. of files	Type	Size	Used	Free	Usage	Version	EOS Version	add	get	rm	3rd	Last OK add	Successful	Failed	factor	add
1. CERN - EOS	ALICE::CERN::EOS	0	41.11 PB	29.45 PB	11.66 PB	71.64%	689,473,435	FILE	41.11 PB	29.61 PB	11.5 PB	72.03%	Xrootd v4.12.8	4.8.78					12.04.2022 13:01	24	0	0	
2. CERN - EOSALICEO2	ALICE::CERN::EOSALICEO2	0	75.61 PB	12.18 PB	63.43 PB	16.11%	27,130,713	TEST	75.53 PB	30.8 PB	44.73 PB	40.78%	Xrootd v4.12.8						12.04.2022 13:11	24	0	0	
3. CERN - OCDB	ALICE::CERN::OCDB	0	63.66 TB	10.53 TB	53.13 TB	16.55%	7,134,825	FILE	63.66 TB	3.544 TB	60.12 TB	5.566%	Xrootd v4.12.8						12.04.2022 13:01	24	0	0	
4. Kosice - EOS	ALICE::Kosice::EOS	2	824.1 TB	460.3 TB	363.8 TB	55.86%	13,634,750	FILE	796.7 TB	456.4 TB	340.3 TB	57.29%		4.8.69					12.04.2022 13:06	24	0	0	
5. Kosice - SE	ALICE::Kosice::SE	2	420.2 TB	178.6 TB	241.6 TB	42.5%	3,839,916	FILE	420.2 TB	182.6 TB	237.6 TB	43.45%	Xrootd v5.0.3						12.04.2022 12:55	24	0	0	
Total			118 PB	42.26 PB	75.73 PB		741,213,639		117.9 PB	61.04 PB	56.85 PB				5	5	5	5					5

Obr. 1.3: Využitie úložiska v Košiciach a CERN-e

51. Troitsk - SE	ALICE::Troitsk::SE	2	112.7 TB	46.81 TB	65.89 TB	41.53%	922,378	FILE	112.7 TB	58.04 TB	54.64 TB	51.51%
52. UNAM_T1 - EOS2	ALICE::UNAM_T1::EOS2	2	458.4 TB	110.8 TB	347.6 TB	24.17%	3,340,100	FILE	458.4 TB	122.2 TB	336.2 TB	26.66%
53. UPB - EOS	ALICE::UPB::EOS	2	4.618 PB	3.581 PB	1.038 PB	77.53%	73,245,378	FILE	4.406 PB	3.636 PB	787.6 TB	82.54%
54. Vienna - EOS	ALICE::Vienna::EOS	2	227.4 TB	27.12 TB	200.3 TB	11.93%	485,675	FILE	227.4 TB	27.13 TB	200.2 TB	11.93%
55. ZA_CHPC - EOS	ALICE::ZA_CHPC::EOS	2	348.8 TB	158.4 TB	190.4 TB	45.4%	6,947,359	FILE	348.8 TB	189.6 TB	159.2 TB	54.35%
Total			**204.1 PB**	**100.1 PB**	**104 PB**		**1,992,818,941**		**185.6 PB**	**107.2 PB**	**78.43 PB**	

Obr. 1.4: Využitie niektorých úložísk z klastrov pre experiment ALICE

1.2 Urýchľovače

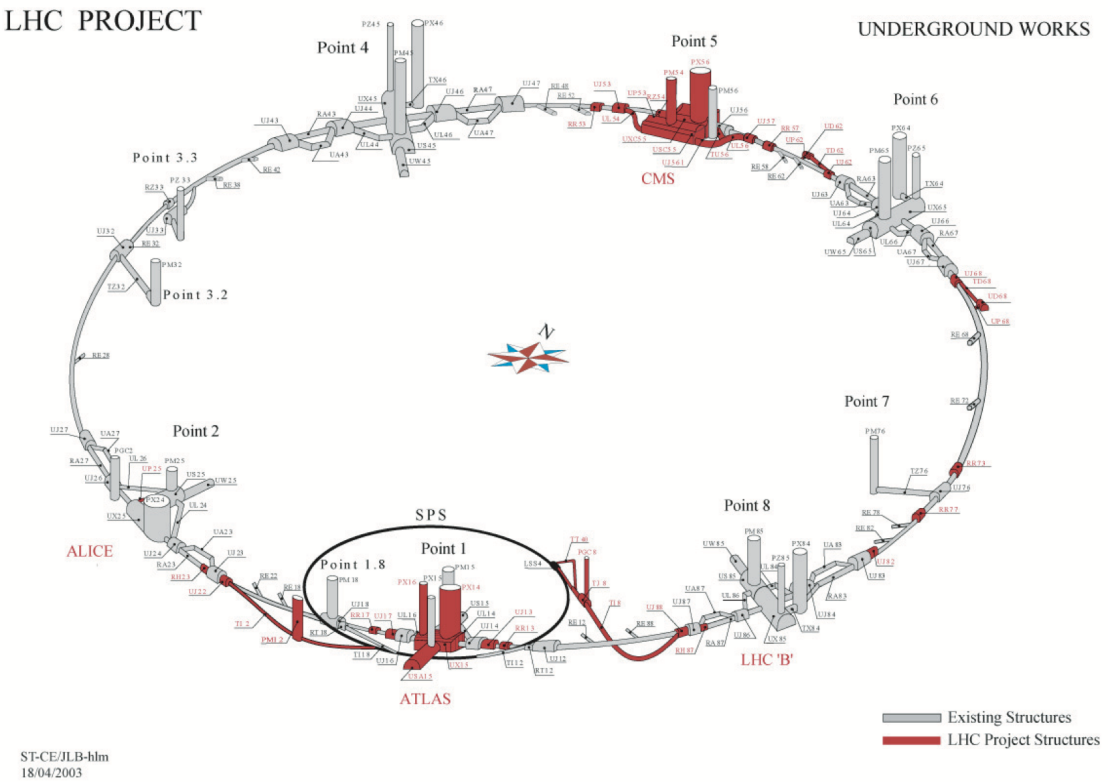
Aby bolo možné experimenty uskutočňovať, je potrebné vytvoriť prostredie, v ktorom môžu nastávať zrážky častíc vo vysokej rýchlosti. Práve pre tento účel slúžia urýchľovače. Urýchľovače sú zariadenia, ktoré dokážu zrýchliť častice, pričom na zrýchlenie používa elektrické pole. V tejto časti budú spomenuté niektoré najznámejšie urýchľovače na svete.

1.2.1 LHC

Veľký hadrónový urýchľovač (skr. LHC, angl. *Large Hadron Collider*) [6] bol zavedený do prevádzky v roku 2010 v organizácii CERN a jeho účelom sú zrážky typu protón-protón. Tento urýchľovač umožňuje posunúť hranice ľudského poznania a umožňuje tiež prekročiť hranice štandardného modelu pre fyzikov. V roku 2012 sa dokonca za pomoci urýchľovača objavil nový bozón, Higgsova častica, čo je jedným z najzásadnejších objavov. V oblasti fyziky vysokých energií je Európa vďaka LHC vo vedúcom postavení. V nasledujúcich rokoch je plánované vytvorenie nového prototypu, testovanie a realizácia. Nová konfigurácia sa

má volať LHC s vysokou svietivosťou (skr. HL-LHC, angl. *High Luminosity LHC*), pomocou ktorej bude možné niekoľkonásobne zvýšiť svietivosť a tým aj rýchlosť zrážok.

Ďalej sa spomína [7], že LHC je dvojprúdový supravodivý urýchľovač umiestnený v 27-kilometrovom tuneli, ktorý bol predtým vybudovaný pre veľký elektrónový urýchľovač (skr. LEP, angl. *Large Electron Positor collider*). Základné usporiadanie LHC zodpovedá geometrii tunela LEP a je znázornené na Obr. 1.5. Tento urýchľovač má osem oblúkov a rovných úsekov. V štyroch rovných úsekoch sa nachádzajú detektory urýchľovača, pričom ostatné sa používajú len na obsluhu. Detektor tiež obsahuje mnoho magnetov, ktorých počet sa pohybuje nad číslom 7000, pričom ich dĺžka môže byť od 10 centimetrov (korektory), až po 15 metrov (dipóly).



Obr. 1.5: Pohľad na rozloženie urýchľovača LHC [7]

1.2.2 NICA

Od roku 2007 sa buduje nový urýchľovací komplex NICA [8] v inštitúte JINR, Dubna, Rusko. Zariadenie je zamerané na experimenty s ťažkými iónmi (až po urán) v rozsahu energií od 4 do 11 GeV/u. Počíta sa taktiež aj so zrážkami polarizovaných deuterónov. Zariadenie obsahuje dva injektorové reťazce, nový supra-

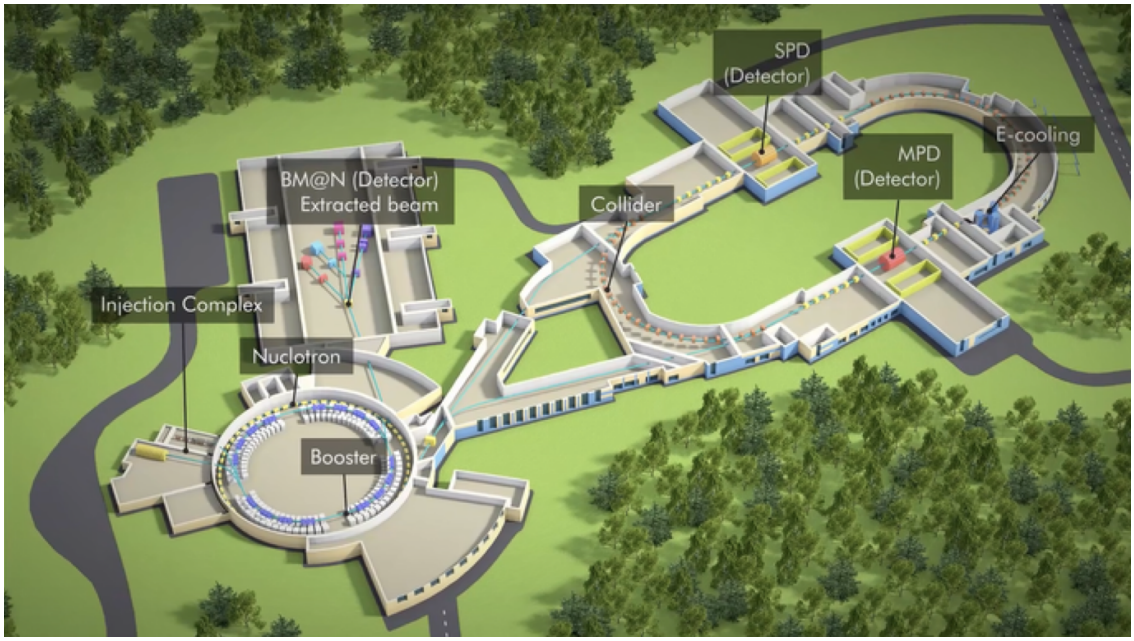
vodivý urýchľovací synchrotrón, existujúci supravodivý synchrotrón Nuclotron a nový supravodivý zrážač pozostávajúci z dvoch prstencov, každý s obvodom približne 500 m. Urýchľovací synchrotrón a urýchľovač NICA sú založené na magnetoch s dominanciou železa a s dutým supravodivým vinutím, ktoré je analogické magnetu nuklotrónu. Magnet je dlhý 2,2 m a má polomer zakrivenia približne 14 m.

Hlavným cieľom projektu NICA [9] je štúdium horúcej a hustej baryónovej hmoty v energetickom rozsahu maximálnej čistej baryónovej hustoty a výskum spinovej štruktúry nukleónov a polarizačných javov. Tento projekt je tiež vhodný pre skúmanie prechodu medzi hadrónovou a kvarkovo-gluónovou fázou pri najvyššej baryónovej hustote. Experimenty projektu NICA, ktoré sú určené pre tieto štúdie sú Baryonová hmota na nuklotróne (skr. BM@N), čo je experiment s pevným terčom v zväzku extrahovanom z nuklotrónu a tiež viacúčelový detektor (skr. MPD, angl. *MultiPurpose Detector*), čo je experiment vykonávaný na urýchľovači. Na to, aby bolo možné uskutočňovať tieto experimenty, je potrebné ďalej rozvíjať existujúci urýchľovač v JINR, Nuclotron, a tiež výstavbu ďalších detektorov a urýchľovačov. Na Obr. 1.6 je znázornený pohľad na hlavné zariadenia NICA. Pre lepšie pochopenie je na Obr. 1.7 znázornená štruktúra a prevádzkový režim tohto komplexu. Spomínaný detektor MPD, zobrazený na Obr. 1.8, pozostáva z hlavňovej časti, v ktorej sa nachádza hlavný sledovací detektor (skr. TPC, angl. *Time Projection Chamber*), ktorý má priemer 2,7m a dĺžku 3,4m, pričom po jej oboch stranách je umiestnených viacero komôr určených na čítanie. Okrem sledovacieho detektora TPC obsahuje hlavňová časť aj systém času letu (skr. TOF, agl. *Time of Flight*) a tiež elektromagnetický kalorimeter (skr. ECal).

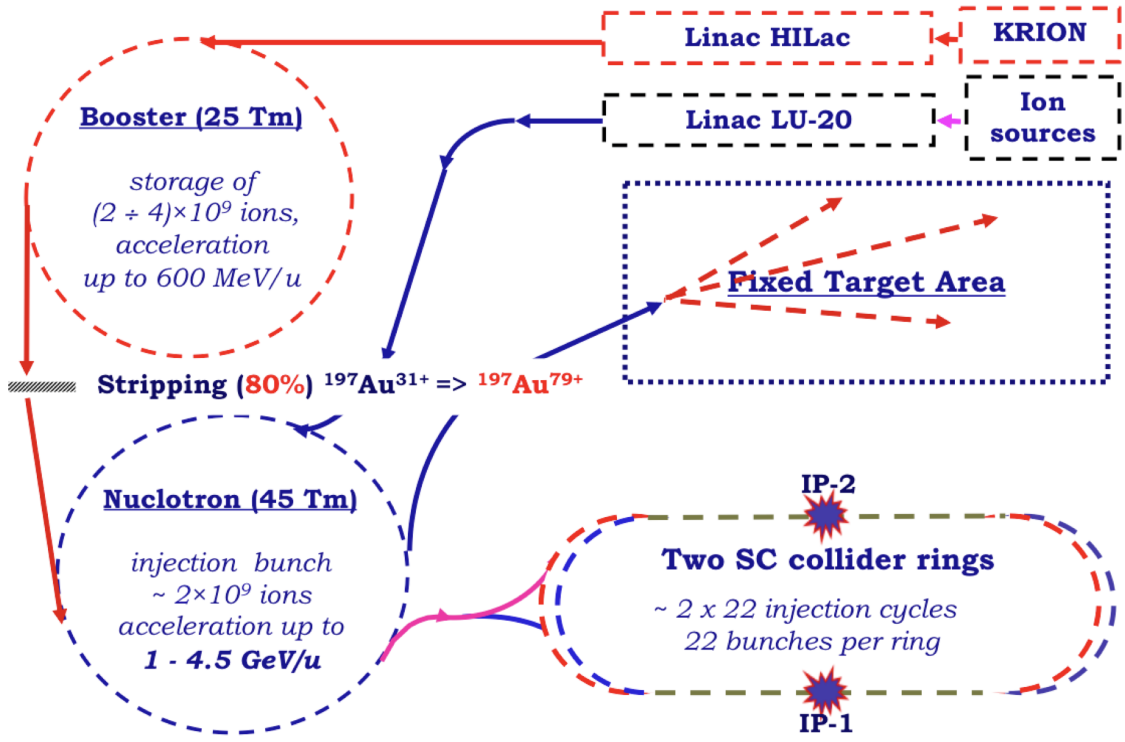
1.3 HEP experiment

Spúšťací systém ľubovoľného HEP experimentu disponuje dvomi simultánnymi režimami, pričom jeden slúži pre detektory so spúšťaním a druhý režim slúži pre detektory, ktoré nepretržite čítajú a zapisujú v obrovských objemoch dát, ktoré dosahujú rýchlosť v stovkách Gbps. Jedným z najväčších producentov dát je HEP experiment ALICE (*A Large Ion Collider Experiment*) [11] na urýchľovači LHC (*Large Hadron Colliden*), ktorého úlohou je štúdium kvarkovo-gluónovej plazmy. V súčasnosti má tento experiment uložené dáta o veľkosti viac ako 100 PB na diskových úložiskách rozložených po celom svete.

Ako je ďalej napísané [5], celé úložisko je rozdelené do šiestich nezávislých oblastí zlyhania (*failure domains*), štyri LHC experimenty (ALICE, ATLAS, CMS

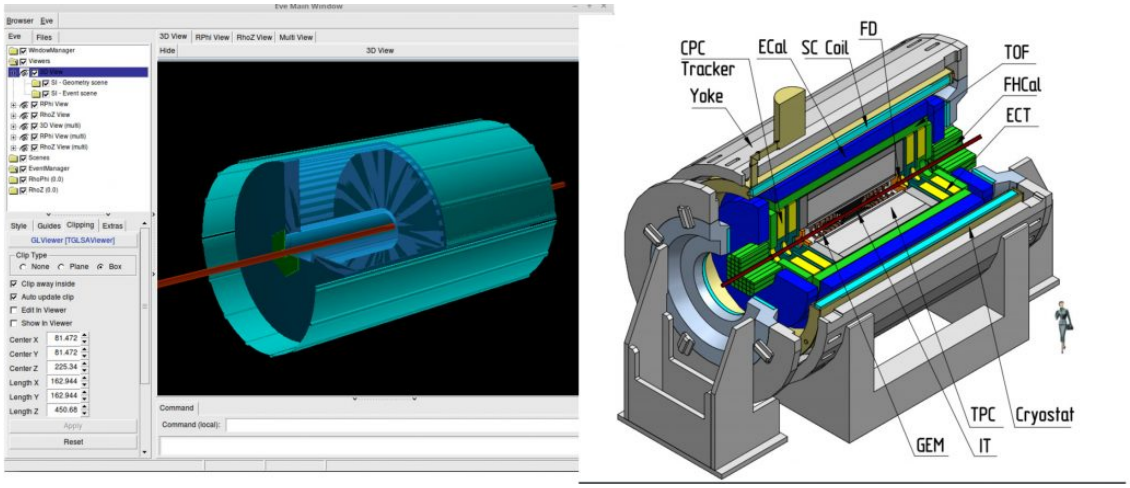


Obr. 1.6: Pohľad na hlavné zariadenia NICA [9]



Obr. 1.7: Štruktúra komplexu NICA [9]

a LHCb), ďalšia zdieľaná inštancia pre menšie experimenty a generická používateľská inštancia pre všetkých CERN používateľov. Medzi hlavné rozdiely patrí veľkosť súboru, pričom pre používateľskú inštanciu sú to najmenšie súbory. Úložisko EOS taktiež rozdeľuje cestu I/O na prístup k metadátam prostredníctvom špeciálnej služby MGM (služba metadát) a prístup k údajom prostredníctvom



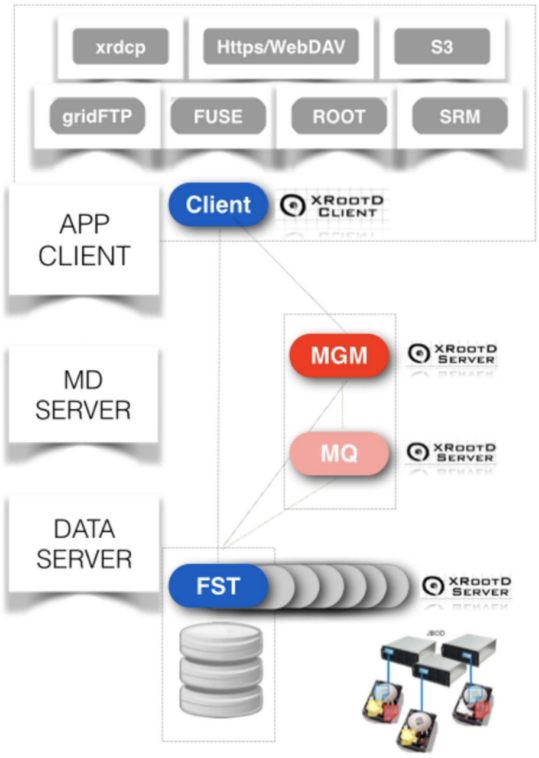
Obr. 1.8: Detektor MPD [10]

služieb I/O súborov FST. Na Obr. 1.9 je možné vidieť architektúru systému EOS. Zvyčajne sú súbory replikované na dvoch diskoch JBOD (*Just a Bunch of Disks*). EOS dodatočne podporuje kódovanie vymazávania súborov (*erasure encoding*) s dvoma alebo tromi redundančnými pruhmi. Aktuálny stav a konfigurácia klastra sú reprezentované zdieľaným hash kódom. Stav a konfigurácie sa vymieňajú medzi úložnými a metadátovými servermi pomocou fronty správ (MQ, *Message Queue*). Všetky tri služby (MGM, FST a MQ) boli implementované za pomoci XRootD klient-server rámca. Služby systému EOS sú distribuované v dvoch počítačových centrách, pričom jedno sa nachádza vo Švajčiarsku a druhé sa nachádza v Maďarsku. Metadátové služby sú nasadené ako šesť párov aktívno-pasívnych párov s možnosťou prepínania v reálnom čase na uzloch s pamäťou o veľkosti 256 GB. Služby ukladania súborov sú nasadené na cca 1400 serverových uzloch s dodatočne pripojeným úložiskom, pričom počet pripojených diskov na uzol môže dosiahnuť až 50. V súčasnosti je uložených vyše 200 miliónov súborov, čo je viac ako pol miliardy replík daných súborov. Systém EOS je taktiež rozšírený o funkcie synchronizácie a zdieľania. Prístup k protokolu HTTP systému EOS je rozšírený tak, aby bol kompatibilný s webovou službou OwnCloud, ktorá je základom pre službu CERNBox, ktorá slúži na synchronizáciu a zdieľanie súborov.

1.4 GRID infraštruktúra WLCG

Infraštruktúra WLCG GRID [12] sa skladá zo štyroch úrovní počítačových centier Tier 0, Tier 1, Tier 2 a Tier 3.

Počítačové centrum úrovne Tier 0 sa nachádza priamo vo švajčiarskej organizácii CERN a slúži na príjem dát priamo z experimentov pri plnej rýchlosti pre-



Obr. 1.9: EOS architektúra [5]

nosu a tiež sa musí postarať o okamžité zapisovanie týchto dát na páskové úložisko. Kvôli striktnej politike zabezpečenia spoľahlivého ukladania nespracovaných údajov je nutné, aby boli všetky údaje uložené v organizácii CERN a druhá kópia dát je okamžite rozdelená medzi počítačové centrá úrovne Tier 1. Aby bol možný rýchly prenos dát, bolo zriadené pripojenie k úrovni Tier 0 s rýchlosťou dosahujúcou 10 GB/s.

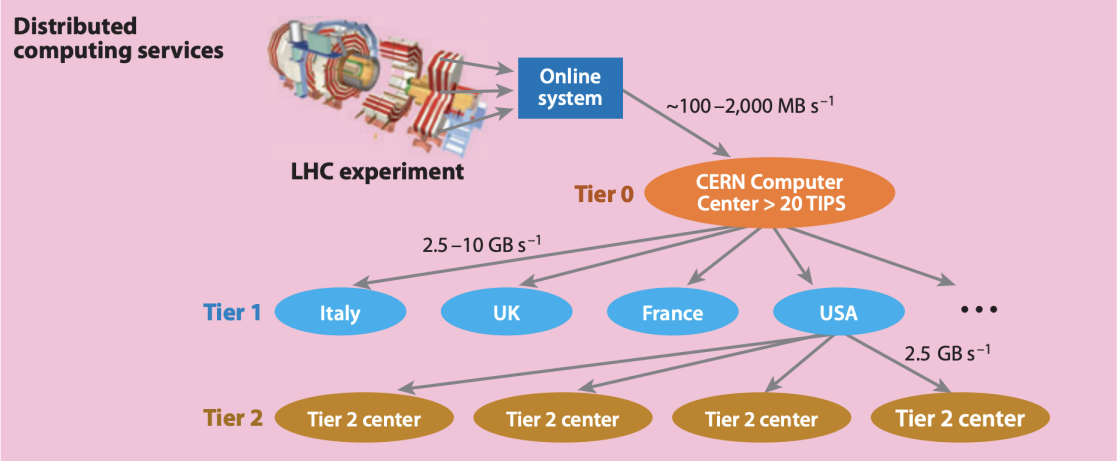
Úlohou počítačového centra úrovne Tier 1 je prijatie údajov z Tier 0, pričom musí zabezpečiť aj dlhodobú archiváciu týchto údajov. Tieto počítačové centrá zvyčajne prevádzkujú aj počítačové klastre, ktoré sú zväčša používané na prácu s nespracovanými dátami s vylepšenou kalibráciou. Taktiež obsahujú zariadenia na analýzu údajov, ktoré sa používajú na rozsiahlu analýzu. Väčšina počítačových centier úrovní Tier 1 podporuje niekoľko počítačových centier úrovne Tier 2, pričom tieto sú umiestnené najmä v krajine, kde sa nachádza aj počítačové centrum úrovne Tier 1, no nemusí to byť vždy. Okrem samotného poskytovania dát, Tier 1 poskytuje podporu v prípade porúch a problémov.

Ďalšou úrovňou sú počítačové centrá úrovne Tier 2. Organizácia týchto centier sa líši od krajiny ku krajine. Príkladom môže byť USA, kde úroveň Tier 2 podporuje len jeden experiment. V rámci Európy je bežné, že počítačové centrum úrovne Tier 2 podporuje niekoľko alebo dokonca všetky experimenty. Tieto centrá po-

skytujú tiež výpočtový výkon pre simulácie Monte Carlo, ktoré niektoré experimenty potrebujú. Počítačové centrá úrovne Tier 2 poskytujú prevažne výpočtové a diskové úložné zdroje, no ich úlohou nemusí byť ich archivácia. Zväčša sú len poskytovateľmi gridových služieb, ktoré sú potrebné na prístup ku ich zdrojom.

Poslednou spomenutou úrovňou sú počítačové centrá úrovne Tier 3, pričom tieto nie sú opísané v memorande o porozumení (*MoU*). Tieto centrá sú dôležité a slúžia ako klienti pre úrovne Tier 2, od ktorých sa očakáva schopnosť distribúcie dát do úrovne Tier 3 na analýzu pre koncových používateľov. Veľkosťou sa počítačové centrá úrovne Tier 3 veľmi líšia, pričom ich veľkosť môže byť od niekoľkých CPU až po veľké národné analytické zariadenia.

Na Obr. 1.10 je možné vidieť infraštruktúru WLCG, pričom tento obrázok aj zobrazuje prenosovú rýchlosť medzi jednotlivými vrstvami infraštruktúry. Taktiež je uvedené [12], že v súčasnosti je v rámci infraštruktúry WCLG vyše 200 000 výpočtových jadier a tiež okolo 100 PB diskov.



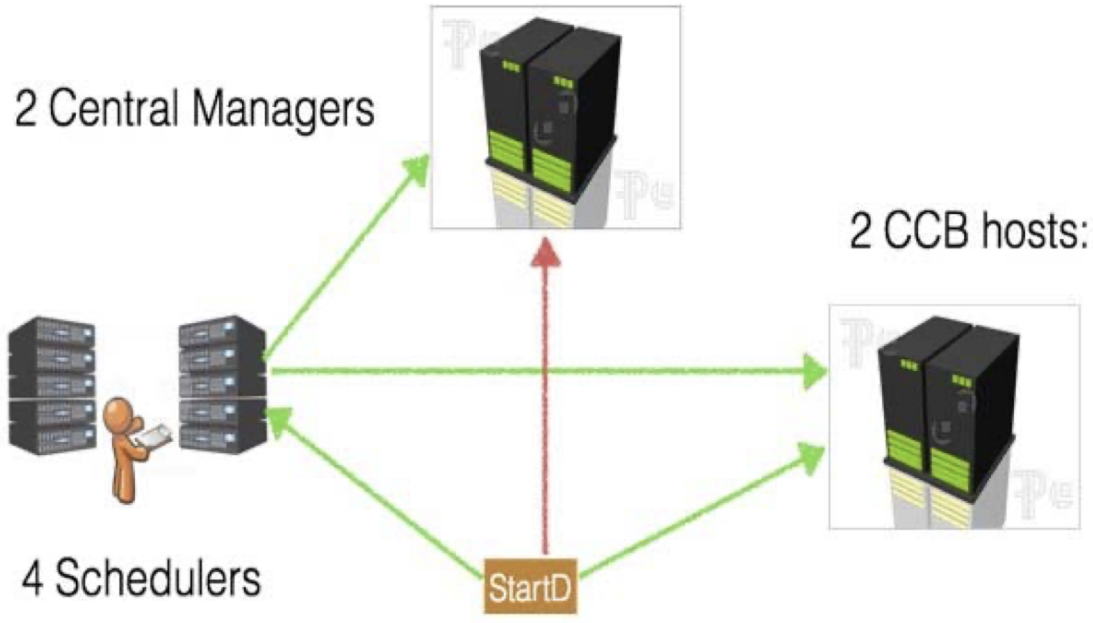
Obr. 1.10: WLCG Infraštruktúra

1.5 Dávkovacie systémy

Ak chceme, aby výkonné klastre riešili rôzne úlohy, ktoré používateľ potrebuje, je potrebné určitým spôsobom orchestrovať tieto klastre. Na tento účel slúžia dávkovacie systémy. Majú za úlohu práve orchestrovať rozdeľovanie úloh medzi uzly a monitorovať stav týchto uzlov. Nasledujúca časť hovorí o najznámejších dávkovacích systémoch používaných vo fyzike vysokých energií.

1.5.1 HTCondor

HTCondor [13] je distribuovaný, vysoko výkonný výpočtový systém, ktorý poskytuje dávkové služby. Komunita okolo fyziky vysokých energií ho vo veľkej miere zaviedla na mnohých počítačových klastroch po celom svete. HTCondor sa skladá z viacerých dôležitých častí, medzi ktoré patria *centrálny manažér*, jeden alebo viacero *vykonávacích uzlov* a tiež jeden alebo viacero *odosielacích uzlov*. Zvyčajne je postup pri odosielaní úloh taký, že používateľ sa prihlási na odosielací uzol a používa nástroje príkazového riadku, ktoré komunikujú s lokálnymi démonmi na zadávanie úloh, získavanie stavu úloh, ukončovanie úloh a pod. Démon *Shedd*, ktorý je spustený na odosielacích uzloch spravuje frontu úloh na vykonanie a slúži tiež ako plánovač. V prípade, že tento démon nemá dostatok vykonávacích uzlov, pošle požiadavky k centrálnemu manažérovi, kde si vypýta viac zdrojov. Tohto démona je možné horizontálne škálovať. Ďalší démon nazývaný *Negotiator*, ktorý beží na centrálnom manažérovi, prijíma tieto požiadavky a pokúša sa nájsť ďalšie vykonávacie uzly. Okrem tohto démona beží na centrálnom manažérovi aj démon *Collector*, ktorý zhromažďuje informácie o všetkých démonoch v rámci systému HTCondor. Odporúčanú architektúru systému HTCondor pre viac ako 100 000 paralelne vykonávajúcich úloh je možné vidieť na Obr. 1.11.

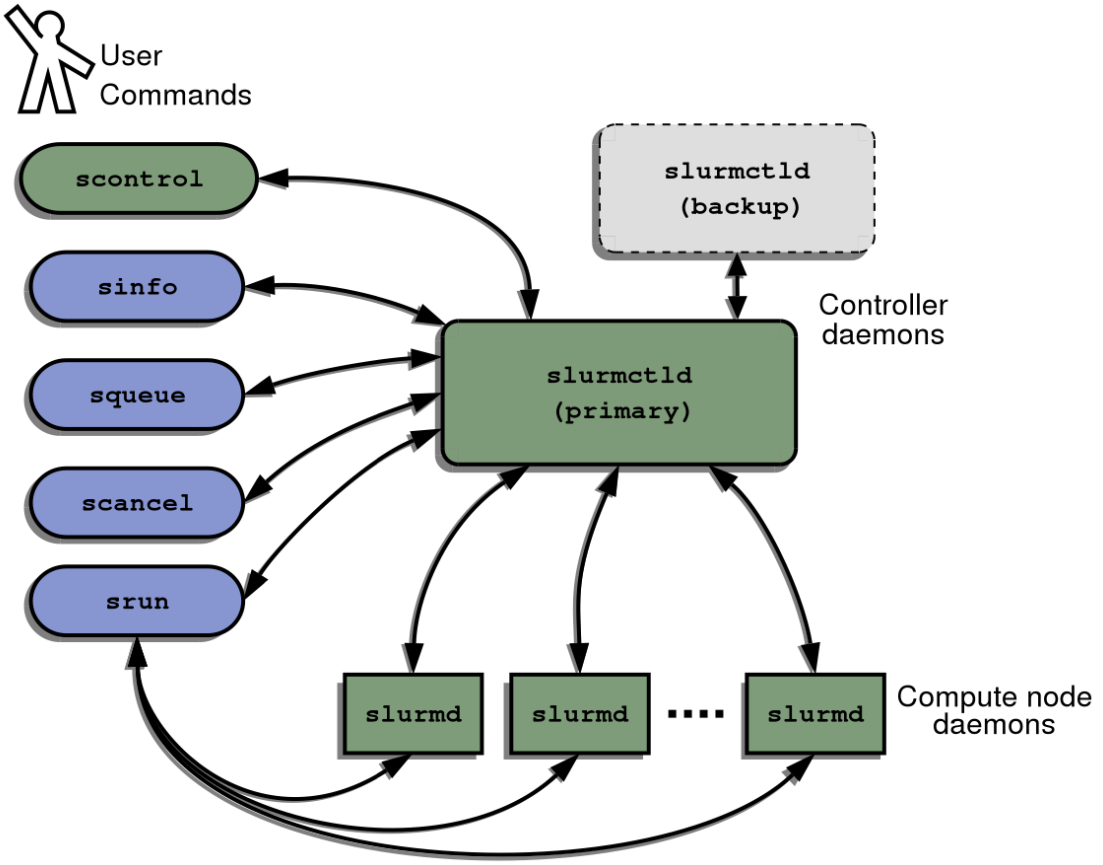


Obr. 1.11: Architektúra HTCondor pre viac ako 100k paralelných úloh [13]

1.5.2 Slurm

Slurm [14], je preferovaný názov voči SLURM, ktorý bol používaný v minulosti na označenie správcu zdrojov. SLURM je skratkou pre *jednoduchý linuxový nástroj na správu zdrojov* (angl. *Simple Linux Utility for Resource Management*).

SLURM systém [15] je vhodný na správu zdrojov a na použitie vo veľkých a malých linuxových klastroch. Tento systém je dostatočne jednoduchý na to, aby umožnil koncovým používateľom porozumieť jeho zdrojovému kódu a pridávať rôzne vlastné funkcie. Okrem toho, že tento systém nie je zložitý, má aj otvorený kód a je licencovaný pod GNU GPL. Ďalšou charakteristikou je škálovateľnosť, teda tento systém je nadizajnovaný na škálovanie na klastre s tisíckami uzlov. SLURM ovládač pre klaster s tisíc uzlami zaberá miesto v pamäti o veľkosti len 2 MB. SLURM má ako správca prostriedkov tri hlavné funkcie. Prvou je prideľovanie exkluzívnych a/alebo neexkluzívnych prístupov k prostriedkom (napríklad výpočtovým uzlom) na určitý čas. Po druhé poskytuje rámec na spustenie, vykonávanie a monitorovanie úloh. A po tretie, rozhoduje o konfliktných požiadavkách na zdroje. Architektúru systému SLURM je možné vidieť na Obr. 1.12.



Obr. 1.12: Architektúra SLURM [15]

SLURM bol primárne vyvíjaný pre klastre založené na Linuxe, no podporuje

taktiež aj IBM BlueGene a IBM SP systémy [16]. SLURM podporuje aj pluginy, čo sú dynamicky pripojené objekty načítané počas behu. Medzi takéto pluginy môžeme radiť napríklad autentifikačný plugin pomocou AuthD alebo Munge. Samotný SLURM sa skladá z niekoľkých entít, pričom tieto delíme na uzly, čo sú individuálne počítače, partície, čo sú fronty s úlohami, úlohy, ktorým sú prideľované zdroje a kroky úloh, čo je sada úloh, ktoré sú zvyčajne paralelné. Uzly sa môžu nachádzať v niekoľkých stavoch, pričom poznáme neznámy stav, alokovaný stav, vykonávajúci, nečinný a vypnutý. Okrem uzlov majú stavy aj samotné úlohy, a tie sú čakajúci, bežiaci, pozastavený, zrušený, dokončený, zlyhaný, zlyhaný uzol, vypršanie limitu. SLURM je založený na démonoch, pričom základným démonom je *slurmctld* (*SLURM Control Daemon*), ktorý orchestruje všetky aktivity SLURM-u naprieč klastrom. Je možné spustiť aj záložného démona. Démon *slurmd* monitoruje stav jednotlivých uzlov, pričom taktiež riadi úlohy a kroky úloh na danom uzle. Tento démon je nenáročný na zdroje a podporuje hierarchickú komunikáciu. Posledným dôležitým démonom je *slurmstepd*, ktorý je nasadený démonom *slurmd* pri inicializácii kroku úlohy. Tento démon spravuje krok úlohy a spracováva jeho vstupy a výstupy. Je aktívny len počas doby, kým je daný krok úlohy aktívny.

1.5.3 Torque

Nástroj Torque [17], je správca zdrojov s plánovačom, ktorý mu zadáva požiadavky. Hlavnou úlohou správcov zdrojov je poskytovanie nízkoúrovňových funkcií na spúšťanie, podržanie, zrušenie a monitorovanie úloh. Tento nástroj je založený z kolekcie počítačov a iných zdrojov, napríklad siete, úložné systémy, licenčné servery a pod. Výhodou dávkovacích systémov je fakt, že tieto systémy zvyčajne znižujú technickú správu zdrojov a zároveň ponúka pre používateľov jednotný pohľad. Bežne sa dávkovacie systémy skladajú z *hlavného uzla*, *odosielacích uzlov*, *výpočtových uzlov* a samozrejme zo *zdrojov*. Torque disponuje jedným hlavným uzlom a mnohými výpočtovými uzlami. Na hlavnom uzle taktiež beží plánovač, ktorý prijíma rozhodnutia o politike využívania zdrojov a prideľuje uzly úlohám. Tento plánovač je typu *First In First Out* (FIFO). Používatelia posielajú úlohy na hlavný *pbs_server* pomocou príkazu **qsub**, následne bude informovaný plánovač, ktorý nájde uzly pre danú úlohu, pošle späť pokyny na spustenie úlohy so zoznamom uzlov. Server odošle úlohu prvému uzlu v zozname a dá mu pokyn na spustenie, pričom tento uzol sa nazýva nadriadený (*Mother Superior*). Ostatné uzly sa volajú sesterské MOM (*sister MOMs*).

1.6 GRID midlvéry

Midlvér (po angl. *middleware*) je softvér, ktorý poskytuje spoločné služby pre aplikácie, ktoré nie sú ponúkané operačným systémom. Medzi služby, ktoré zvládnu midlvéry riešiť, je možné zaradiť napr. správu údajov, posielanie správ, autentifikáciu, správu API a i. V tejto časti sú opísané dva najznámejšie GRID midlvéry, DIRAC a AliEn, pričom tieto môžu menežovať viaceré dávkovacie systémy (SLURM a i.). Ďalej je v tejto časti opísaný spôsob autentifikácie GRID midlvérov.

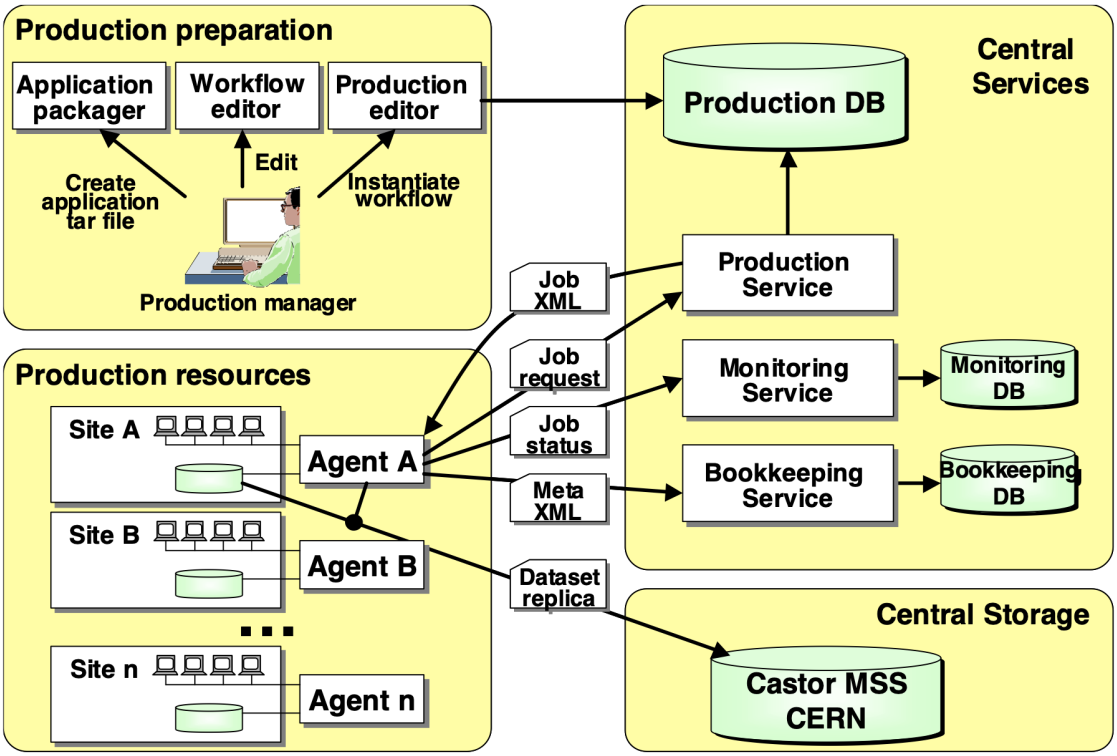
1.6.1 DIRAC

DIRAC je riešenie pre GRID, ktorý je využívaný ako distribuovaný zdroj. S týmto nástrojom je možné spravovať úlohy a tiež aj pracovné zaťaženie na samotnom GRID-e.

DIRAC [18] sa skladá z mnohých komponentov, pričom jedným z nich je komponent pre správu údajov, ktoré poskytujú prístup k štandardným systémom na ukladanie údajov na bežných FTP, SFTP alebo HTTP serveroch. Keďže DIRAC je vytvorený modulárne, tak to umožňuje výber podmnožiny funkcií vhodných pre konkrétne aplikácie alebo možnosť veľmi jednoducho pridať svoju vlastnú chýbajúcu funkcionalitu. Samotný DIRAC sa riadi paradigmou architektúry orientovanej na služby (SOA, angl. *Service Oriented Architecture*). Architektúru systému DIRAC je možné vidieť na Obr. 1.13.

Projekt DIRAC [20] je projekt s otvoreným kódom, ktorý bol spustený spolu s LHCb projektom a neskôr bol v roku 2009 otvorený vďaka záujmu o jeho prijatie zo strany inej komunity. Aktuálne je dostupný a umiestnený na platforme GitHub a je uvoľnený pod licenciou GPLv3. Jeho vývoj nie je financovaný, a tak sú vítané komunity, ktoré ho používajú, aby sa podieľali na vývoji tohto projektu. Ďalej tento zdroj uvádza niekoľko používateľov, či komunít, ktoré nástroj DIRAC používajú. Prvou známou komunitou je práve *LHCb*, ktorá je aj jej prvotným vývojárom a v dnešnej dobe je stále jeho hlavným správcom. LHCb používa DIRAC vo veľkej miere, pričom ho využíva takmer na všetky svoje potreby spojené s distribuovanými výpočtami, kde využíva rôzne výpočtové zdroje, súkromné klastre a dávkovacie systémy. Vďaka projektu DIRAC môže LHCb komunikovať s úložiskami prostredníctvom rôznych protokolov, napríklad DIP (interný DIRAC protocol). Druhou známou komunitou je ohľadom experimentu *Belle II*. Tento experiment je vykonávaný na asymetrickom energetickom urýchľovači v Japonsku.

DIRAC implementuje WMS (Workload Management System), ktorý dokáže simultánne spravovať viacero úloh pre danú komunitu používateľov, pričom nie



Obr. 1.13: Architektúra DIRAC [19]

je nutné, aby tieto úlohy boli rovnakej povahy [21]. Tieto úlohy sa môžu pohybovať v rozsahu od krátkych úloh s veľmi vysokou prioritou až po chaotické záťaže koncových používateľov. Okrem toho samozrejme zahŕňajú aj celospoločenské aktivity s rôznou prioritou, akými sú napríklad simulácie Monte Carlo a simulácie v reálnom čase. S cieľom optimalizovať prístup k výpočtovým zdrojom sa na začiatku vývoja projektu DIRAC zvolilo riešenie s jediným centrálnym serverom, na ktorom sa nachádza celý rad čakajúcich úloh na vykonanie.

1.6.2 Rozdiely medzi dávkovacími systémami

Ako už bolo spomenuté, cieľom tejto práce je schopnosť riadiť získavanie experimentálnych údajov, pričom tieto dáta chceme získavať a aj odosielať na rôzne typy dávkovacích systémov. Problémom, ktorý nastáva pri tomto projekte je fakt, že všetky tieto spomenuté dávkovacie systémy majú rôznu dobu odozvy na vykonávanie úloh. Najrýchlejším systémom je SALSA, ktorý je schopný odpovedať na spustenie úloh rádovo v sekundách, o niečo pomalšie to trvá pri dávkovacom systéme SLURM, kde to prevažne zaberie rádovo desiatky sekúnd a najhoršie, čo sa odozvy týka, dopadol dávkovací systém DIRAC, ktorý je schopný odpovedať na spustenú úlohu niekedy až po 3 minútach. Tieto hodnoty boli otestované, pričom ide o spriemerované hodnoty, keďže prebehlo viacero testovaní v rámci

každého dávkovacieho systému. Ako je možné vidieť z výsledkov, niektoré časy odozvy sú nedostatočné a práve preto je potrebné vytvoriť spôsob, ktorým by sa mohli tieto časy zredukovať.

1.6.3 AliEn

AliEn [22] je súbor midlvér nástrojov a služieb, ktoré implementujú infraštruktúru Grid. Vývoj začal v roku 2000 a bol veľmi rýchlo použitý na distribuovanú produkciu Monte Carla na viacerých vzdialených počítačových pracoviskách. Najdôležitejšou funkciou AliEn sú rozhrania na iné Grid riešenia. Medzi hlavné komponenty AliEn je možné zaradiť katalóg súborov s funkciami metadát, nástroje na správu údajov na prenos a ukladanie údajov, autentifikácia a autorizácia, systém riadenia pracovného zaťaženia, rozhrania na iné Grid implementácie, rozhranie ROOT a monitoring.

1.6.4 Autentifikácia GRID midlvérov

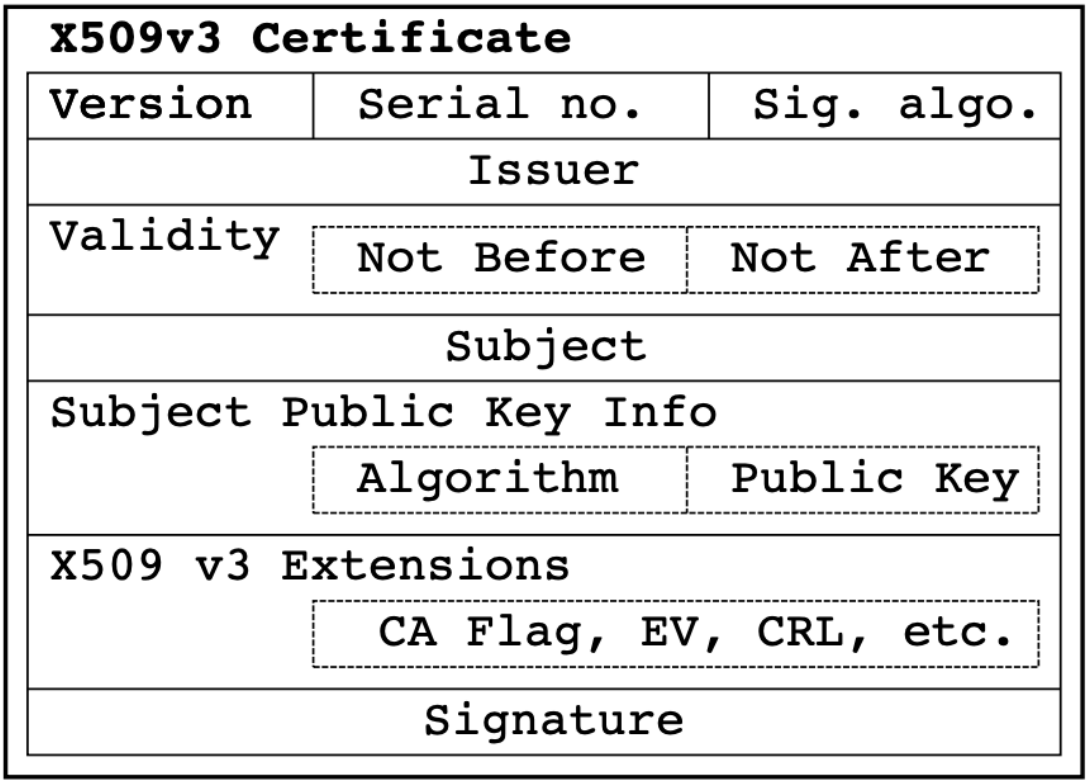
Keďže je potrebné zabezpečiť celú infraštruktúru GRID pred zneužitím treťou stranou, je nutné zabezpečiť prístup ku výpočtovým zdrojom a tiež ku samotným midlvérom. Vo fyzike vysokých energií sú výskumné skupiny bežne rozptýlené po inštitúciách a geografických oblastiach [23]. V súčasnosti sa v celosvetovej počítačovej sieti LHC (WLCG) poskytuje autentifikácia prostredníctvom X.509 cerifikátov, ktoré vydávajú akreditované certifikačné autority IGTF. Individuálny používateľ potom môže získať certifikát a prostredníctvom miestnej registračnej autority môže preukázať svoju totožnosť. To znamená, že tento vydaný certifikát predstavuje identitu daného používateľa.

V prípade dávkovacieho systému SLURM je možné okrem infraštruktúry verejného kľúča *PKI* (*Public Key Infrastructure*) použiť aj klasickú UNIX autentifikáciu, teda pomocou klasického prihlásenia s menom a heslom alebo taktiež pomocou LDAP (*Lightweight Directory Access Protocol*). LDAP je primárne určený na vytváranie požiadaviek a možnosť úprav adresárov v cieľovom systéme pomocou protokolu TCP/IP.

1.6.5 X.509

X.509 [24] je štandard, ktorý dáva možnosť autentifikácie v rôznych odvetviach, ako napr. nakupovanie, online bankovníctvo a i. Zvyčajne je možné vidieť zabezpečenie aplikácii za pomoci sady protokolov SSL a TLS (*Transport Layer Security*),

pričom sú založené na dobre známych kryptografických algoritmoch. Tieto algoritmy je možné matematicky analyzovať s ohľadom na bezpečnosť. Naproti tomu je infraštruktúra X.509 založená nie len na kryptografii, ale taktiež aj na rôznych organizáciách a subjektoch. Certifikačné autority, skr. CA (*Certification Authority*), certifikujú identity a verejné kľúče iných subjektov v X.509 certifikáte. Webové prehliadače obsahujú tzv. koreňový sklad (angl. *root store*), ktoré obsahujú zoznam certifikačných autorít, ktorým bude daný prehliadač dôverovať.



Obr. 1.14: Schematický pohľad na certifikát X.509 verzie 3 [24]

1.6.6 Certifikačná Autorita

Na to, aby mohol byť akýkoľvek X.509 certifikát používaný a bolo mu možné dôverovať, je potrebné aby ho nevytvárala súkromná osoba, ale verejne známa CA. Táto autorita zohráva v rámci PKI (*Infraštruktúra verejného kľúča*, *Public Key Infrastructure*) veľmi dôležitú úlohu [25]. Je možné povedať, že CA je autoritou, ktorej dôveruje jeden alebo viac používateľov, aby vytvárala a prideľovala certifikáty verejných kľúčov. Jej úlohou je byť treťou stranou, pričom osvedčuje identitu koncového subjektu. To je dosiahnuteľné tak, že daný subjekt poskytne CA dostatočný dôkaz o svojej totožnosti a následne autorita vygeneruje správu obsahujúcu totožnosť subjektu a verejný kľúč. Túto správu nazývame *certifikát* a je

kryptograficky podpísaná autoritou. Verejné kľúče certifikačných autorít musia byť distribuované všetkým tým, ktorí dôverujú danej autorite. CA je hierarchicky rozdelený, pričom najvyššie sa nachádza tzv. *koreňová certifikačná autorita*, ktorá je najdôveryhodnejšia a musí distribuovať svoje kľúče ako certifikáty podpísané vlastným podpisom s prijateľným formátom certifikátu a distribučným protokolom. Autorita musí tiež sprístupniť svoje verejné kľúče vo forme čitateľného textu pre prípad, aby spoliehajúce sa subjekty mohli rozlíšiť certifikáty podpísané vlastným podpisom a autoritou.

1.7 Virtuálna realita

Keďže táto práca má za cieľ sprostredkovať spôsob, ako použiť prostredie virtuálnej reality a webovej aplikácie na manipuláciu s dávkovacím systémom a samotnými klastrami, je potrebné objasniť, čo je virtuálna a rozšírená realita.

1.7.1 Čo je virtuálna realita

VR alebo *virtuálnu realitu* [26] je možné definovať ako spôsob oklamania mozgu a vnímania človeka na úrovni, že samotný mozog nevie rozlíšiť či to, na čo sa díva, je skutočné alebo umelé. Podobný princíp využívali napríklad kúzelníci a rôzni umelci, ktorí sa snažili pomocou rôznych trikov zmiasť oči a mysle. Počítačové systémy prinášajú presvedčivé obrázky, zvuky, pocity a iné súčasti, ktoré vytvárajú úplne imaginárne ale realisticky vyzerajúce svety, pričom v prípade AR sú fyzické predmety v reálnom svete pozmenené. Pri použití virtuálnej, či rozšírenej reality spolu s umelou inteligenciou sa dostávame do nepoznaného sveta možností.

S príchodom virtuálnej, rozšírenej či zmiešanej reality prichádzajú možnosti, ktoré predtým neboli možné. Medzi takéto možnosti môžu byť radené virtuálne obchodné stretnutia, rôzne nebezpečné športy, ako napríklad skok padákom, jazda na horskej dráhe alebo napríklad návšteva odľahlých ostrovov. V súčasnosti je tiež možné vidieť použitie VR a AR aj v iných priemysloch ako je herný. Jedným takýmto odvetvím je zdravotníctvo, kde môžu tieto technológie pomôcť lekárom pri diagnostike a pri liečení pacientov.

1.7.2 Existujúce technológie

Keďže virtuálna realita získava čím ďalej tým viac pozornosti, tak, samozrejme, existujú technológie, ktoré s ňou pracujú. V tejto sekcii budú opísané niektoré z

nich.

V rámci článku [27] je spomenutých vyše 50 hardvérových a tiež softvérových nástrojov, ktoré slúžia pre vývojárov na zlepšenie používateľského zážitku. Tento zoznam obsahuje nástroje ako **Unreal Engine** a **Unity**, ktoré sú zvyčajne používané v rámci herného priemyslu. Spoločnosť Valve vytvorila nositeľnú sadu, **SteamVR**, ktorá dokáže preniesť používateľa do virtuálnej reality a v nej môže hrať rôzne herné tituly. V rámci nositeľných technológií, v dnešnej dobe existuje niekoľko riešení, medzi ktorými sa nachádza **Oculus Rift**, čo je headset od spoločnosti Oculus, ktorý podobne ako *SteamVR* dokáže vtiahnuť používateľa do VR. Ďalšou firmou, ktorá vytvorila VR headset je spoločnosť Google, pričom ich nástroj sa volá **Google Daydream** a tento umožňuje vstup do virtuálnej reality pomocou mobilného telefónu. Na podobnom princípe funguje aj **Samsung Gear VR** od spoločnosti Samsung. Najjednoduchším riešením, ktorým je možné dosiahnuť prostredie VR je použitie **Google Cardboard**, ktorý sa doslova skladá len z kartónu a dvoch šošoviek.

Okrem hardvéru a softvéru pre desktopy sú vyvíjané tiež sady nástrojov pre vývoj softvéru, ktoré sú nazývané SDK (*Software Development Kit*). Tieto sady slúžia pre vývojárov, aby nemuseli vytvárať sami nástroje, ktoré by prepojili nimi vytváraný softvér s virtuálnou realitou. Jedným z najznámejších sád nástrojov pre webové rozhranie je **A-Frame**, ktorý je napísaný v jazyku JavaScript a je určený na použitie s jazykom HTML spolu s WebVR. Alternatívami pre sadu nástrojov *A-Frame* sú **Google VR** od spoločnosti Google, **VRWorks** od spoločnosti NVIDIA alebo **Cardboard** opäť od spoločnosti Google.

1.7.3 Čo chýba v analýze dát

Hlavným nedostatkom v dnešnej dobe je fakt, že nie je možné ovládať dávkovacie systémy a klastre len pomocou virtuálnej reality. Zmenou v tomto smere je projekt NDMVR [1]. Tento projekt má za úlohu sprístupniť správu klastrov pomocou webových technológií virtuálnej reality, ako napríklad A-Frame.

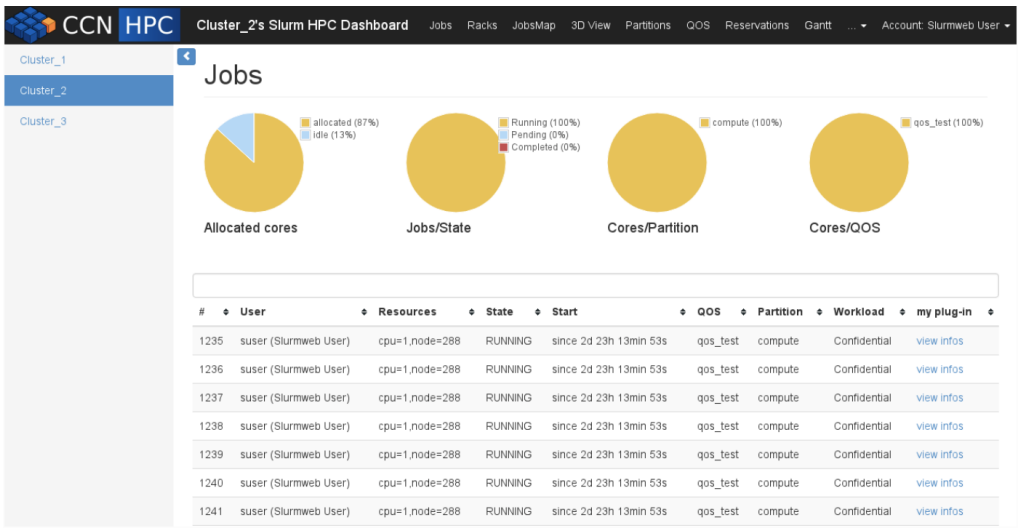
1.8 Súčasný prístup k riadeniu získavania dát

Keďže táto práca má za úlohu riadiť získavanie experimentálnych údajov na účely vizualizácie, pred samotnou implementáciou je nutné analyzovať, aké súčasné spôsoby existujú v dnešnej dobe. Pre potreby získavania údajov je potrebné ana-

[1] https://gitlab.com/ndmspc/ndmvr

lyzovať dávkovacie systémy, pretože vďaka nim sú údaje získavané.

V prípade dávkovacieho systému SLURM, existuje niekoľko spôsobov získavania údajov, a tiež možnosť spúšťať úlohy. Prvou možnosťou je použitie príkazového riadku, pričom táto možnosť podporuje všetky potrebné schopnosti, `scontrol` príkaz umožňuje získať údaje o stave dávkovacieho systému, `sbatch` a `srun` umožňujú spúšťať úlohy. Príkaz `squeue` povoľuje pozrieť informácie o úlohách, ktoré sa nachádzajú vo fronte v dávkovacom systéme. SLURM obsahuje aj tzv. účtovníctvo (*accounting*), pričom tento vytvára záznamy o vykonávaných úlohach a tiež o úlohách, ktoré už boli vykonané. Jedným z príkazov na použitie pri účtovníctve je `sacct`, ktorý zobrazuje aktuálny stav zdrojov pre bežiace alebo ukončené úlohy. Pre potreby nastavovania a riadenia účtovníctva bol vytvorený aj príkaz `sacctmgr`, ktorý umožňuje zobraziť a meniť SLURM informácie v rámci účtovníctva. Okrem samotného prostredia príkazového riadku vytvorili vývojári aj webového klienta pre SLURM, nazývaný **slurm-web** [2], ktorý dokáže poskytovať údaje o dávkovacom systéme SLURM pomocou webového rozhrania. Problémom však je, že táto webová aplikácia dokáže získavať dáta len z dávkovacieho systému a nie je možné s ňou spúšťať úlohy. Táto aplikácia používa ako backend PySLURM knižnicu spolu s rámcom Flask, pričom samotná webová aplikácia bola vytvorená pomocou knižnice jQuery a rámca Bootstrap. Ukážka panelu je možné vidieť na Obr. 1.15

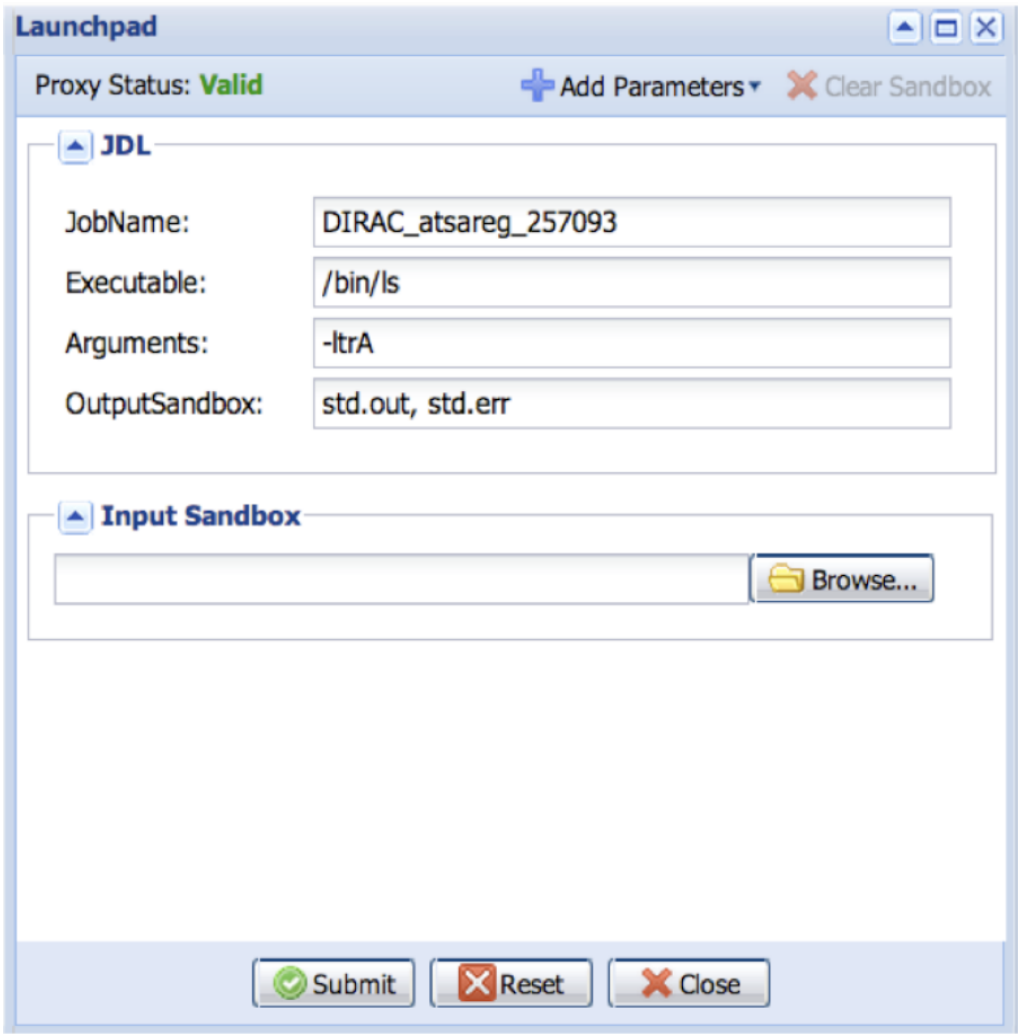


Obr. 1.15: Slurm-web panel

V prípade dávkovacieho systému DIRAC, existuje podobne ako pri systéme SLURM rozhranie príkazového riadku, pomocou ktorého je možné manipulovať s dávkovacím systémom, teda získavať údaje a odosielať úlohy. Avšak narozdiel od

[2] https://edf-hpc.github.io/slurm-web/

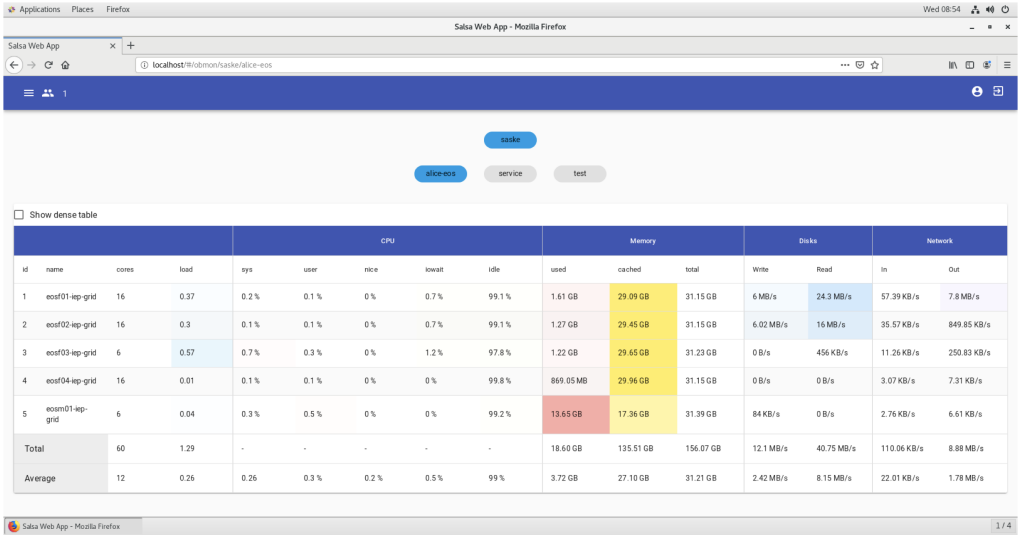
SLURM-u, DIRAC poskytuje **Web Job Launchpad**, čo je webová aplikácia zahrnutá v rámci *DIRAC Web Portal*. Jej ukážka je na Obr. 1.16. Táto aplikácia umožňuje používateľovi formulovať a zadávať jednoduché úlohy. V prípade, že vývojári by si chceli vytvoriť vlastné implementácie, je možné použiť Python modul **DIRAC**, ktorý obsahuje funkcionalitu na vytvorenie a spustenie úlohy. DIRAC Web Portal je webová aplikácia, ktorá poskytuje prístup ku všetkým aspektom systému DIRAC. Umožňuje monitorovať a kontrolovať úlohy na tomto dávkovacom systéme. Webová aplikácia bola vytvorená s použitím prvkov grafického rozhrania, ktoré napodobňujú desktopové aplikácie. Nevýhodou tejto aplikácie však je, že v prípade, ak si chce používateľ zobraziť viac ako 100 riadkov tabuľky s údajmi, je táto webová aplikácia veľmi pomalá.



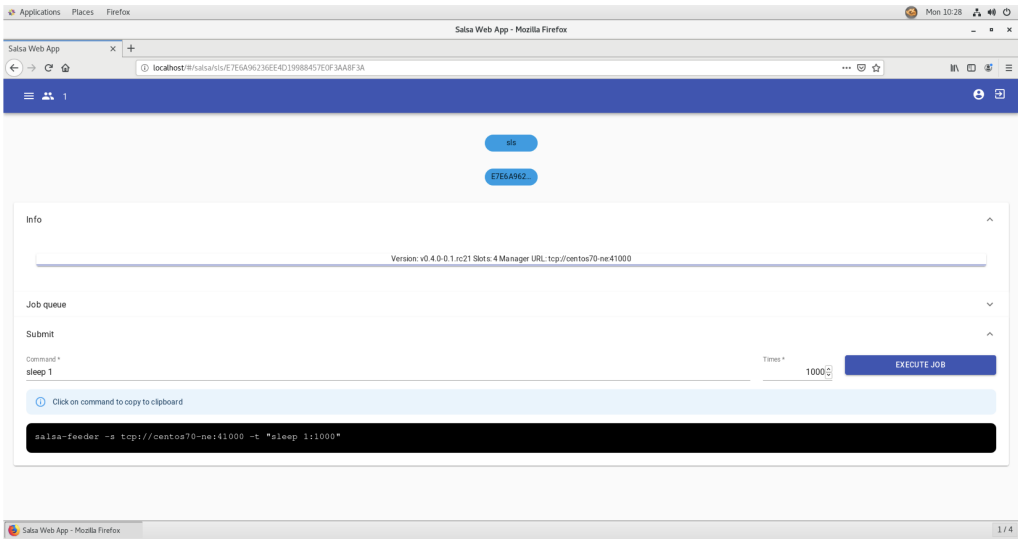
Obr. 1.16: Web Job Launchpad

Pre dávkovací systém SALSA rovnako existuje rozhranie príkazového riadku, ktorým je možné tento dávkovací systém ovládať. Okrem príkazového riadku vytvorili vývojári systému SALSA aj terminálové používateľské rozhranie (TUI) po-

mocou knižnice Curses, ktoré graficky zobrazuje priebeh vykonávania úlohy v termináli. V rámci tohto dávkovacieho systému nebolo priamo implementované žiadne webové prostredie, ktoré by mohlo slúžiť pre používateľov na interakciu so systémom. Avšak práca [28] sa zaoberá práve implementáciou takého webového rozhrania, ktoré by slúžilo pre prácu s úlohami v rámci dávkovacieho systému SALSA. Jej hlavnými časťami sú zobrazenie práve prebiehajúcich úloh v dávkovacom systéme a tiež zaisťovanie odosielania úloh. Súčasný stav tejto webovej aplikácie povoľuje prácu so systémom SALSA. Monitorovanie úloh je možné vidieť na Obr. 1.17 a spúšťanie úloh vo webovom rozhraní je na Obr. 1.18.



Obr. 1.17: SALSA webové rozhranie - monitorovanie



Obr. 1.18: SALSA webové rozhranie - spúšťanie úloh

Okrem samotných dávkovacích systémov, niektoré spoločnosti taktiež vytvorili webové aplikácie či služby, ktoré pomáhajú vysokovýkonnej výpočtovej tech-

nike (HPC). Firma Atos ponúka službu **Extreme Factory**, pričom táto je označovaná ako HPC a AI ako služba (*HPC & AI as-a-Service*). Táto služba bola vytvorená v roku 2010 spoločnosťou Bull, ktorá bola neskôr odkúpená spoločnosťou Atos. O tento produkt je možné požiadať na súkromné využitie alebo na požiadanie (*on demand*). Tento produkt v sebe zahŕňa tiež webový portál **Extreme computing studio v3** (*XCS3*), responzívny portál, ktorým je možné ovládať viaceré dávkovacie systémy, medzi ktorými je aj SLURM. Nevýhodou tohto riešenia však je, že tento produkt je propietárny, teda má zatvorený zdrojový kód, čo znemožňuje vytváranie vlastných rozšírení. Taktiež je platený, čo pri ostatných riešeniach neplatí a najdôležitejšie je, že tento produkt nepodporuje veľké množstvo dávkovacích systémov. Ďalším verejne dostupným nástrojom je **Bull Efficiency Manager** (skr. *BEM*), pričom tento nástroj je tiež webová aplikácia bežiaca nad dávkovacím systémom SLURM, pričom tento poskytuje len detailné informácie o klastroch. Tieto informácie sú zobrazované v grafoch a tabuľkách pre aktuálne a minulé dáta o zdrojoch klastrov. Problém s týmto nástrojom však je, že tento nástroj neslúži na priamu manipuláciu s úlohami, či už monitorovanie ich stavu alebo odosielanie nových úloh.

Po analýze všetkých možností, ktoré sú definované vyššie, je možné povedať, že webové nástroje, ktoré sú dostupné priamo v rámci dávkovacích systémov, je možné použiť len s danými systémami, čo znamená, že ak je požiadavka na webovú aplikáciu, aby cez ňu bolo možné ovládať viaceré druhy dávkovacích systémov, tieto webové aplikácie nie je možné použiť. Ďalšími možnosťami, ktoré existujú, sú propietárne produkty, ktoré majú nevýhodu, že tieto nedovoľujú modifikáciu a tiež nepodporujú všetky dávkovacie systémy. Výsledkom tejto analýzy je fakt, že je potrebné vytvoriť vlastné webové rozhranie, resp. aplikáciu, ktorá bude podporovať všetky potrebné dávkovacie systémy, popr. bude možné jednoducho pridať implementáciu pre ďalšie systémy.

1.9 Technológie pre vytvorenie knižnice

Táto časť analýzy opisuje technológie, ktoré budú použité pri vytváraní knižníc pre webové rozhrania. Opísaná tu bude knižnica React, protokol WebSocket a tiež rámec PatternFly.

1.9.1 React

Knižnica React je napísaná v skriptovacom jazyku JavaScript, pričom sa využíva na vytváranie webových stránok či webových aplikácií. Tvorcom knižnice je sve-

tovo známa spoločnosť Meta, ktorá ju vytvorila v roku 2013.

React [29] je využívaný na vývoj znovupoužiteľných komponentov používateľského rozhrania. Táto knižnica umožňuje vývoj veľkých a komplexných webových aplikácií, v ktorých je možné meniť údaje bez toho, aby bolo nutné obnoviť stránku. Z pohľadu MVC modelu je možné zaradiť knižnicu ako V, teda pohľad (angl. *view*) [30]. React abstrahuje objektový model dokumentu (DOM), vďaka čomu ponúka jednoduchý vývoj. Viazanie údajov medzi komponentmi je možné vďaka jednosmernému toku dát, ktoré táto knižnica implementuje. Primárnym jazykom využívaným v React-e je jazyk JSX. JSX je jazyk, ktorý je veľmi podobný klasickému jazyku HTML. Používanie tohto jazyka nie je povinné, no jeho použitie má výhody, medzi ktorými je napríklad písanie značiek pre komponenty spolu s ich príslušnými udalosťami. React je veľmi výkonný najmä vďaka virtuálnemu objektovému modelu dokumentov, ktorý slúži na to, aby neboli všetky zmeny aplikované na reálny DOM, ale aby bolo vyhodnotené, ktoré zmeny je potrebné reálne vykonať, a tým je možné ušetriť výpočtový výkon. Jediným rozdielom medzi reálnym objektovým modelom dokumentu a virtuálnym je ten, že druhý spomenutý je uložený v pamäti a nie je potrebné ho vykresľovať, pričom reálny model je nutné vykresľovať, čo je pomalá operácia. Vyhodnocovanie zmien je pozorovateľné napríklad pri udalostiach, kedy je zmenené veľké množstvo dát.

Spôsobov ako začať vývoj s knižnicou React je niekoľko, pričom najjednoduchším spôsobom je možnosť importovať samotnú knižnicu priamo do HTML kódu. V tomto prípade je v dnešnej dobe možné použiť už aj ESM alebo *EcmaScript Modules*, ktoré nahrádzajú pôvodné importovania pomocou kľúčového slova **require**. Miesto neho je možné importovať moduly za pomoci kľúčového slova **import**.

1.9.2 WebSocket

WebSocket je protokol [31] schopný posielať dáta v oboch smeroch, teda od klienta ku serveru a tiež od servera ku klientovi. Samotný protokol pozostáva z úvodného nadviazania spojenia (*handshake*), po ktorom nasledujú rámce so správami cez protokol TCP. Cieľom tohto protokolu je poskytnúť možnosť pre webové aplikácie komunikovať obojsmerne so servermi, pričom nie je potrebné na otváranie viacerých spojení cez protokol HTTP (napríklad za pomoci *XMLHttpRequest*).

1.9.3 PatternFly

PatternFly [32] je predstavený ako návrhový systém vytvorený na podporu konzistentnosti a zjednotenie tímov s otvoreným kódom. Základom PatternFly sú

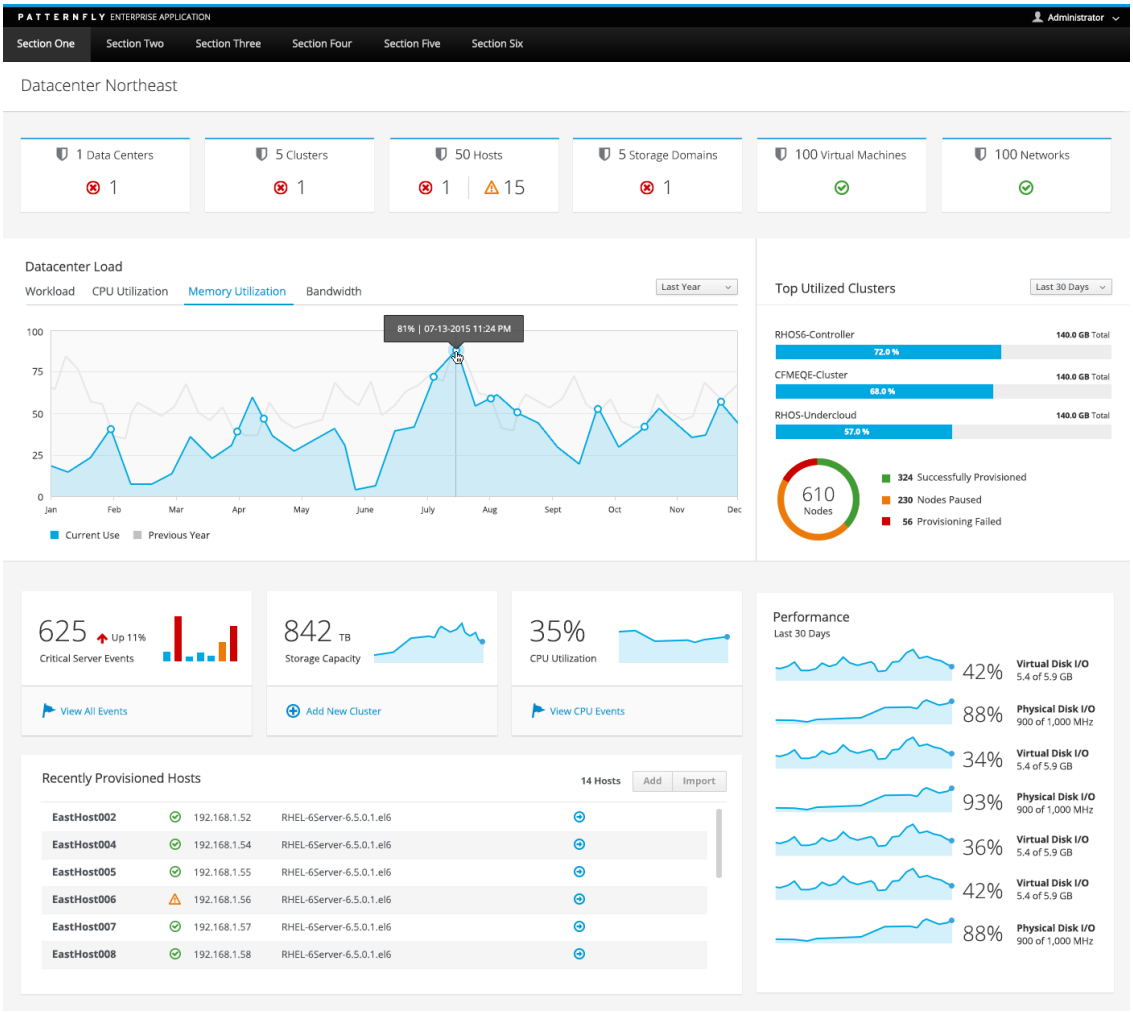
usmernenia pre návrh. Tento systém poskytuje plnú podporu pre tvorbu responzívnych stránok, pričom pomáha udržiavať komponenty usporiadané a zarovnané bez ohľadu na veľkosť obrazovky. Komponenty sú navrhnuté tak, aby boli flexibilné a modulárne, takže je možné ich kombinovať a vytvoriť riešenie akéhokoľvek problému s používateľským rozhraním. Tento systém je vytvorený a spravovaný spoločnosťou Red Hat, ktorá stojí aj za známymi Linux distribúciami, akými sú napríklad Fedora, RHEL, CentOS.

Okrem toho je tiež spomenuté, že základnú štruktúru návrhového systému PatternFly tvoria **komponenty**, pričom sú navrhnuté tak, aby ich bolo možné použiť na takmer akýkoľvek UI (*User Interface*) problém [33]. Ďalšou zložkou štruktúry sú rozloženia, pričom tieto poskytujú plne responzívne štruktúry stránok, ktoré udržujú komponenty organizované a zarovnané bez ohľadu na veľkosť obrazovky používateľa. Demá sú taktiež súčasťou štruktúry, pričom tieto demá zobrazujú, ako môžu byť viaceré komponenty použité v jednom dizajne. Samotný systém ponúka aj CSS premenné, ktoré môže vývojár použiť a taktiež aj meniť, pričom tieto premenné sa delia na globálne a komponentové. Ukážku stránky s použitým PatternFly je možné vidieť na Obr. 1.19.

1.9.4 Cockpit

Cockpit [34] je projekt, ktorý poskytuje ľahko použiteľné, integrované, prehľadné a otvorené grafické používateľské rozhranie vo forme webovej aplikácie na správu serverov. Pomocou tohto projektu je možné spravovať servery s Linuxom bez toho, aby bolo nutné písanie a spúšťanie príkazov do príkazového riadku. Cockpit používa rovnaké systémové nástroje, aké by bolo možné použiť z príkazového riadka. Tento projekt má dokonca integrovaný terminál, čo je užitočné v prípade, že sa používateľ alebo správca pripája zo zariadenia, ktoré nie je založené na Linuxe. Cockpit používa už existujúce používateľské rozhranie systému, pričom v predvolenom nastavení používa bežné prihlasovacie údaje a oprávnenia používateľov daného systému. Cockpit nevyužíva zdroje a ani nebeží na pozadí v prípade, keď nie je využívaný. Je spúšťaný na požiadanie vďaka aktivácii *systemd soketov*.

Vyvinula ho spoločnosť Red Hat, pričom samotný projekt má otvorený kód, teda *open-source*. Jeho rozranie je veľmi používateľsky prívetivé a jednoduché na použitie, pričom je pomocou neho možné monitorovať, upravovať serverové konfigurácie. Samotný server môže pracovať na jednej z mnohých distribúcií založenej na GNU/Linux jadre. Okrem týchto vlastností Cockpit umožňuje pristupovať ku viacerým serverom či klastrom z toho istého rozhrania.



Obr. 1.19: PatternFly [32]

Je dôležité podotknúť, že aj keď je zdrojový kód projektu otvorený, tak ho tvorí skupina vývojarov zo spoločnosti Red Hat, pretože daný projekt je dosť zložitý a rozsiahly.

Pri nasadzovaní Cockpitu na server či klaster je nutné povedať, že samotné nasadenie je jednoduché, pričom projekt samotný používa rovnaké nástroje, ktoré sa dajú použiť z príkazového riadku. Kedykoľvek je možné spravovať server buď cez Cockpit, alebo cez terminál. V rámci webového grafického rozhrania projektu sa nachádza aj vstavaný terminál, takže je možné zadávať príkazy kedykoľvek a odkiaľkoľvek.

V rámci našej práce využijeme ďalšiu možnosť, ktorú nám tento projekt poskytuje a to je **rozšíriteľnosť**. V kontexte projektu Cockpit rozšíriteľnosť znamená, že je možné nainštalovať rôzne dodatočné aplikácie tretích strán, či dokonca si vytvoriť a nainštalovať vlastnú aplikáciu. V tomto prípade vytvárame aplikáciu napísanú pomocou JavaScript knižnice React, ktorá slúži na tvorbu webových stránok a webových aplikácií. Tím vývojárov pripravil **Starter Kit** pomocou ktorého

je možné veľmi jednoducho pripraviť vlastnú aplikáciu.

Každý projekt, ktorý je určený na použitie v reálnych podmienkach musí spĺňať niekoľko kritérií [35]. Prvým je oddelenie kódu HTML, CSS a JS. Oddelenie týchto kódov zaručuje, že je možné používať bezpečné CSP (angl. *Content Security Policy*). Tieto politiky sú odporúčané pre stránky tretích strán a vyžadované pre vlastné stránky projektu Cockpit. Druhým kritériom je využitie moderných softvérových rámcov na tvorbu obsahu stránok. Je výhodnejšie použiť rámec alebo knižnicu, ako napríklad React, miesto skladania reťazcov jazyka HTML. V prípade použitia knižnice React spolu s návrhovým systémom PatternFly, je možné, aby stránka zapadla do dizajnu projektu Cockpit. Ďalšími kritériami sú použitie nástrojov Babel a ESLint, ktoré umožňujú písanie kódu v modernej verzii jazyka JavaScript, resp. odhalenie chýb v kóde. Ďalej sú to kritériá na systémy zostavovania, ako napríklad Webpack, potom je to vytváranie tarballov, RPM balíkov, testovanie a následné nasadenie. *Cockpit Starter Kit* slúži práve na vytvorenie projektu, ktorý spĺňa všetky kritériá spomenuté vyššie. Tento nástroj poskytuje jednoduchú stránku napísanú pomocou knižnice React, ktorá používa aplikačné programové rozhranie projektu Cockpit a tiež spriedovný test, ktorý overuje túto stránku.

Okrem toho je Cockpit opísaný ako nástroj na správu serverov sponzorovaný spoločnosťou Red Hat, ktorého úlohou je poskytovanie moderne vyzerajúceho a používateľsky prívetivého rozhrania na správu a administráciu serverov [36]. Prvou verziou operačného systému Fedora, ktorý obsahoval Cockpit je verzia Fedora 21. V Red Hat Enterprise Linux (RHEL) verzie 7 bol tento nástroj zahrnutý ako voliteľný a od verzie 8 ho obsahuje už štandardne. Medzi najdôležitejšie funkcie Cockpit-u je možné zaradiť moderný dizajn, modularita, teda je možné ho rozšíriť inštaláciou alebo vytváraním vlastných modulov. Ďalej je to podpora viacerých serverov z jedného ovládacieho panela, využívanie **systemd** soketov, pričom ak nie sú používané, tak nevyužíva žiadnu pamäť. Taktiež využíva rovnaké aplikačné programové rozhranie ako príkazový riadok, nespúšťa sa ako hlavný používateľ **root** a je kompletne zdarma. Ako ďalej píše, na vytvorenie vlastných modulov je možné použiť predpripravený "Starter Kit".

1.10 Technológie pre vytvorenie API

Podobne ako v prípade technológií pre vytvorenie knižníc pre webové rozhrania, je potrebné opísať technológie, ktoré sú používané pri vývoji API, teda *bekend*. Táto časť zabezpečuje spracovanie a odosielanie dát klientovi, resp. webovému

rozhraniu. Tiež opisuje tri z najviac rozšírených rámcov a knižníc pre tvorbu API, **FastAPI**, **Flask** a **Express**. Okrem opisu týchto technológií tu bude spomenutý výber vhodnej technológie.

1.10.1 FastAPI

Keďže je potrebné v rámci práce taktiež vytvoriť API server, je potrebné vybudovať API, ktoré bude umožňovať používateľovi zadať svoju úlohu cez HTTP požiadavku. Prvou možnosťou je FastAPI, moderný, výkonný webový rámec, ktorý slúži na vytváranie API v jazyku Python. Medzi hlavné vlastnosti FastAPI je možné radiť rýchlosť, ktorá je porovnateľná s rýchlosťou NodeJS alebo jazyka Go. To je umožnené vďaka použitiu Pydantic modulu, rýchlemu vývoju, jeho jednoduchosti, ktorá ho predurčuje na úlohy, kedy je potrebné veľmi rýchlo pripraviť server. Tiež je možné povedať, že rámec FastAPI je robustný a je vhodný na použitie v produkcii. Okrem týchto základných vlastností ponúka FastAPI aj niektoré nadštandardné služby, či vlastnosti, ktorými sa odlišuje od iných platforiem a rámcov, a to je použitie špecifikácie OpenAPI (*Swagger*).

1.10.2 Flask

Druhou možnosťou, tiež veľmi známou medzi vývojármi v jazyku Python, je Flask mikrorámec. Slovíčko *mikro* v slove mikrorámec značí, že jadro rámca Flask sa snaží byť jednoduché, ale rozšíriteľné. Základom Flasku je možnosť vývojára rozhodnúť sa, čo daný vývojár potrebuje bez toho, aby Flask použil akékoľvek predvolené nastavenia. V predvolenom nastavení napríklad Flask neobsahuje akúkoľvek abstraktnú vrstvu nad databázou, validácie a i. Na pridanie týchto možností je možné pridať rozšírenie pre Flask, ktoré to spraví. Výhodou je zabezpečenie proti cross-site scripting útokom (*XSS útoky*). Toto je zabezpečené pomocou šablonovacieho nástroja Jinja2.

1.10.3 Express

Za zmienku taktiež stojí využitie webového rámca Express, ktorý je vytvorený pre Node.js. Je to flexibilný rámec slúžiaci pre webové aplikácie, pričom poskytuje robustnú sadu funkcií pre webové a mobilné aplikácie. V tomto rámci je možné vytvoriť rôzne midlvéry, ktoré môžu slúžiť na rôzne účely, pričom sú volané pri príchode požiadavky na server ešte pred tým, ako sa táto požiadavka posunie na správnu funkciu. Medzi takéto midlvéry môžeme radiť napríklad overenie autentifikácie a autorizácie, vytváranie logov, počítadlo požiadaviek a i. Medzi dobré

zvyky pri používaní rámca Express (platí to aj pri ostatných rámcoch), je používanie TLS (teda poskytovanie služby cez protokol HTTPS). Priamo v prípade Express rámca je dobré vypnúť poskytovanie hlavičky **X-Powered-By**, ktorý vie ponúknuť útočníkom informáciu o tom, aký rámec bol použitý pri vývoji a na ktorom beží samotný API server.

1.10.4 Výber technológie

Dôležitou voľbou je výber servera, ktorý bude poskytovať API. Medzi nami zvažované technológie radíme Python a Node.js. V rámci jazyka Python sú to softvérové rámce Flask a FastAPI a v rámci Node.js to je knižnica Express.js. Keďže požadujeme, aby bol API server dostupný (*uptime* bol vysoký, najlepšie 100 %) a aby bol schopný spracovať mnoho používateľov a ich požiadaviek, je potrebné otestovať každý z týchto troch rámcov, resp. knižníc na záťaž (*load testing*). Pri hľadaní možností, ako overiť schopnosť práce serverov so záťažou, sme narazili na knižnicu napísanú v jazyku JavaScript, nazývanú Artillery. Tento nástroj poskytuje záťažové a dymové skúšky, pričom je možné vytvoriť milióny požiadaviek za sekundu. Dymové skúšky (angl. *smoke testing*) je druh testovania, pri ktorom sa zisťuje, či daný softvér je stabilný alebo nie. Väčšinou je potvrdením pre QA tím, čo povoľuje ďalší vývoj na softvéri. Výhodou tohto nástroja je fakt, že obsah testu môže byť spísaný v rámci konfiguračného súboru v jazyku YAML.

Medzi ďalšie faktory, ktoré môžu vplývať na výber technológie pre server, radíme rýchlosť vývoja a dodatočne ponúkané výhody. V prípade týchto výhod môžeme zaradiť pri FastAPI automatické generovanie dokumentácie pomocou nástroja Swagger, veľmi jednoduché spracovanie chýb, ktoré je centralizované, jednoduché testovanie a tiež rýchly vývoj. Na druhej strane, výhodou rámca Flask je fakt, že existuje dlhšie a je overený v praxi, čo znamená, že je viac dôveryhodný.

Pomocou nástroja Artillery, ktorý slúži na testovanie záťaže, sme otestovali všetky tri spomenuté rámce, aby sme si mohli zvoliť rámec, ktorý bude poskytovať rýchlu odpoveď a bude schopný spracovávať väčšie množstvo požiadaviek v jednom čase.

Prvým otestovaným rámcom je FastAPI. Zo všetkých 3000 požiadaviek zvládol spracovať všetkých 3000, teda 100 %, pričom časy odozvy boli v rozmedzí 1 ms a 47 ms, kde medián bol na úrovni 3 ms a 99. percentil (99 % všetkých požiadaviek) je na úrovni 7.9 ms.

Druhým je Flask, druhý z Python rámcov. V tomto prípade sa podarilo tiež spracovať 100 % požiadaviek (všetkých 3000). Časy odozvy sú v rozmedzí od 0 ms do 19 ms, pričom medián hodnôt je 3 ms a 99. percentil je 8.9 ms.

Posledným je Node.js webový rámec Express, jeden z najznámejších rámcov pre tvorbu API v jazyku JavaScript. Rovnako ako predošlé dva rámce, Express bol schopný spracovať všetkých 3000 požiadaviek. Minimálny čas odozvy na požiadavku bol 0 ms a maximálny čas 16 ms, medián 1 ms a 99. percentil 5 ms.

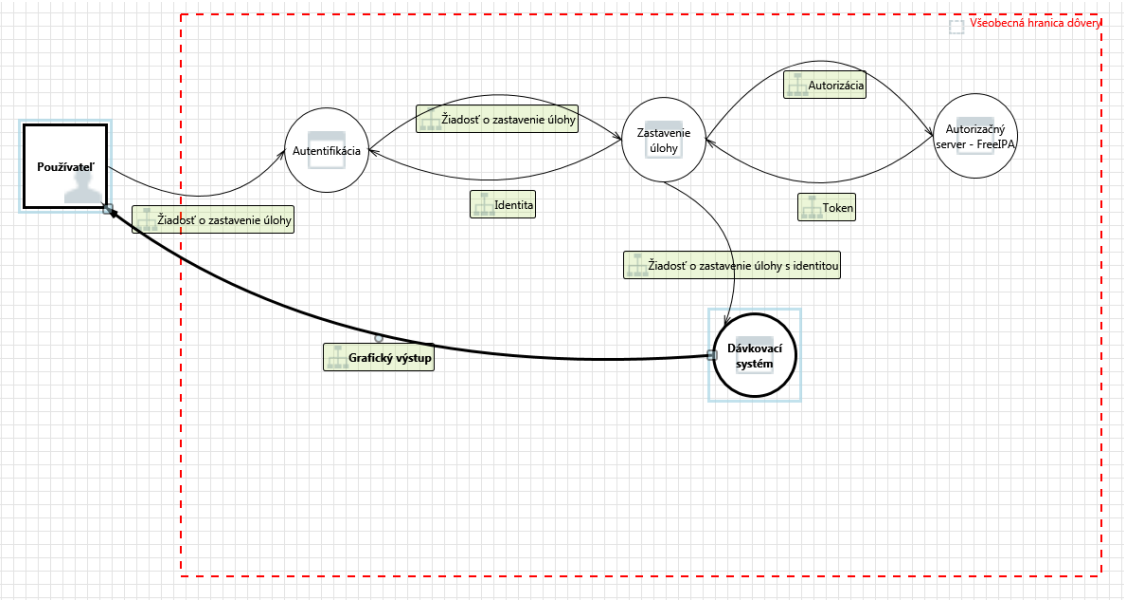
Z uvedených výsledkov vyplýva, že najlepší rozptyl výsledkov časov odozvy mal rámec Express, pričom najlepší medián (stredná hodnota) dosiahol tiež Express a rovnako tak aj 99. percentil. To znamená, že pri 3000 požiadavkách dosiahol najlepšie výsledky rámec Express. Na druhú stranu však treba spomenúť, že Express vo svojom predvolenom nastavení neposkytuje také možnosti, ako napríklad FastAPI a tiež vývoj serveru nie je taký rýchly ako pri FastAPI. Po dlhom premýšľaní nad voľbou rámca sme sa rozhodli použiť **FastAPI** aj napriek tomu, že testovanie na 3000 požiadavkách ukázalo o trochu pomalšie časy odozvy.

1.11 Modelovanie hrozieb

V rámci predmetu Bezpečnosť informačných a komunikačných systémov, bolo potrebné zhotoviť správu o modelovaní hrozieb, ktorá slúži ako bezpečnostná analýza pre daný projekt či systém. V tomto prípade bola vyhotovená správa pre túto diplomovú prácu, pričom na vytvorenie bol použitý nástroj Threat Modelling Tool od spoločnosti Microsoft. Samotný vygenerovaný diagram z tohto nástroja obsahuje niekoľko predmetov, pričom používatelia sú zvyčajne označení pomocou štvorcov, procesy a služby pomocou kruhov, prechody označujú atribúty, ktoré nastávajú pri prechode a červená prerušovaná čiara okolo niektorých predmetov označuje tzv. všeobecnú hranicu dôvery. Pri prechode touto hranicou je zvyčajne nutné zabezpečiť autentifikáciu a autorizáciu, pretože táto hranica rozdeľuje samotný systém na časť, ktorá môže byť všeobecne viditeľná pre každého a na časť, ktorá musí byť primárne skrytá a je dostupná len pre určitú časť používateľov. Na Obr. 1.20 je možné vidieť diagram vygenerovaný pre túto prácu. Medzi mnohé hrozby zistené týmto nástrojom patrí vyzdvihnutie privilégií (*elevation of privilege*), podvrhnutie (*spoofing*), odmietnutie (*repudiation*), odmietnutie služby (*DoS, Denial of Service*), falšovanie (*tampering*) a sprístupnenie informácií (*Information Disclosure*).

Proti hrozbe **podvrhnutia** je použité prihlasovanie a teda bežný neprihlásený používateľ nemá právo meniť cieľ, kam sa majú rôzne dáta poslať, čo by mohol využiť útočník. **Odmietnutie** je možné zmierniť tým, že používateľ aktualizuje stránku napríklad pomocou klávesy F5. Také isté riešenie je možné použiť v prípade hrozby **odmietnutia služby**. V prípade **vyzdvihnutia privilégií** je nutné

zabezpečiť prihlasovanie menom a heslom, keďže vyzdvihnutie privilégií hovorí o tom, že daný predmet sa môže vydávať za iný s cieľom získať ďalšie privilégiá. Prihlasovanie zmierňuje okrem iného aj hrozbu **podvrhnutia**, kde používateľ môže byť podvrhnutý útočníkom, čo by mohlo viesť ku neoprávnenému prístupu. Ďalšou dôležitou hrozbou, ktorú je potrebné eliminovať alebo aspoň zmierniť, je **sprístupnenie informácií**. Táto hrozba hovorí o tom, že existuje možnosť, že útočník by mohol odchytiť dôležité informácie a tieto použiť na získanie oprávnení a privilégií. Riešením pre túto hrozbu je zabezpečiť komunikáciu protokolu HTTP pomocou TLS, teda použitie HTTPS.



Obr. 1.20: Diagram zo správy o modelovaní hrozieb

Celkovo bolo zistených 19 možných hrozieb, pričom 16 z nich bolo zmiernených a 3 neboli aplikovateľné.

1.12 Zhrnutie analýzy

V tejto časti analýzy by bolo vhodné zhrnúť všetky pozorovania, ktoré boli spomentuné v rámci tejto kapitoly práce. Keďže existuje veľké množstvo dávkovacích systémov a tieto majú rozdielne implementácie, a teda samotné správanie, bolo potrebné analyzovať niektoré z nich a zistiť rozdiely medzi nimi.

Analýzou prešli dávkovacie systémy SALSA, SLURM a DIRAC. Ich rozdiely je možné pozorovať na čase odpovede od dávkovacieho systému. V tomto smere má prevahu dávkovací systém SALSA, keďže tento dokázal odpovedať na požiadavky klienta v jednotkách sekúnd. Horšie na tom boli dávkovacie systémy

SLURM a DIRAC, kde čas na získanie odpovede bol vyše desať sekúnd, resp. tri minúty.

Ďalším dôležitým poznatkom z analýzy je fakt, že v dnešnej dobe je možné ovládať dávkovacie systémy výlučne za pomoci prostredia príkazového riadku. Aj keď existujú implementácie webových aplikácií, tieto sú schopné iba zobrazovať aktuálny stav. V prípade veľkého množstva dát sa stane aplikácia pomalou, teda nepoužiteľnou.

Na základe tejto analýzy je potrebné vytvoriť webovú aplikáciu a tiež vytvoriť prepojenie medzi rôznymi dávkovacími systémami a klientom. V rámci ďalšej kapitoly budú spomenuté prípady použitia, vytvorenie sprostredkovateľa, nastavenie klastrov, implementácia grafického používateľského rozhrania vo webovom prehliadači s JavaScript-ovou knižnicou React, vytvorenie NPM (*Node Package Manager*) balíka a jeho následné nasadenie do nástroja Cockpit.

2 Syntetická časť

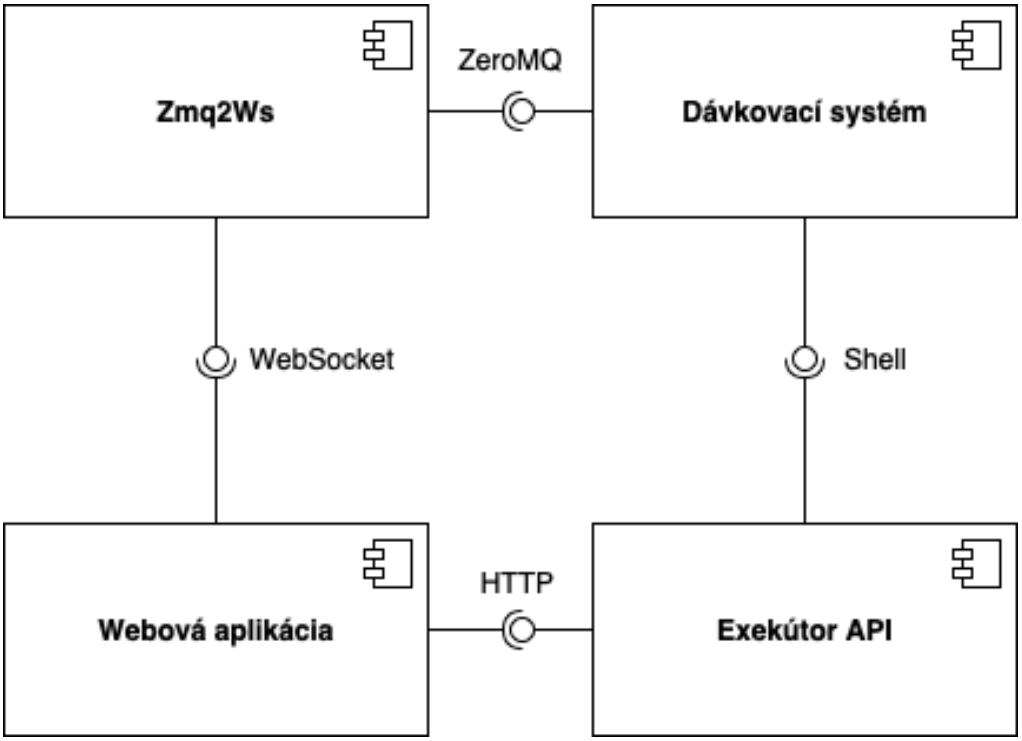
Úlohou tejto časti je opísať návrh riadenia získavania experimentálnych údajov. Kapitola je užitočná najmä pri určovaní, návrhu a implementácii všetkých požiadaviek, ktoré sú očakávané od používateľského rozhrania. Hlavným cieľom rozhrania je umožniť autentifikovaným používateľom manipuláciu s dávkovacími systémami, bez nutnosti práce s rozhraním príkazového riadku v termináli. Používateľ by mal mať možnosť manipulovať s rôznymi dávkovacími systémami, teda samotné rozhranie nie je orientované len na jeden špecifický typ.

2.1 Architektúra riešenia

V rámci konceptuálneho modelu je veľmi výhodné začať s navrhovaním architektúry systému za pomoci rôznych druhov UML diagramov. V súčasnosti poznáme veľké množstvo druhov diagramov, pričom každý slúži na iný účel. V prípade tejto práce je možné použiť sekvenčný diagram na prechod dát a informácií medzi webovým rozhraním a samotným dávkovacím systémom. Zistili sme, že najlepšie tento presun dát zobrazuje sekvenčný diagram, ktorý sa skladá z viacerých častí, stĺpcov, pričom každý predstavuje jeden subjekt, ako aplikáciu, nástroj a pod. Časová os prechádza zhora nadol, pričom jednotlivé interakcie častí je možné znázorniť pomocou šípok smerujúcimi medzi jednotlivými časťami. Návratové hodnoty sú zväčša označované prerušovanou čiarou, pričom je tiež zvykom označovať daný proces textovo nad danou šípkou.

Okrem toho, keďže naše riešenie je doplnením už existujúcich riešení, je výhodné zobraziť prepojenia medzi jednotlivými časťami pomocou komponentového diagramu. Celkové riešenie sa skladá z dávkovacieho systému, sprostredkovateľa Zmq2Ws, webového rozhrania a API exekútora, ktorý slúži na vykonávanie úloh. Spomínaný komponentový diagram je možné vidieť na Obr. 2.1. Najdôležitejšou časťou je dávkovací systém, ktorý je prepojený so sprostredkovateľom Zmq2Ws a API exekútorom, pričom pre prepojenie so Zmq2Ws používa protokol ZeroMQ, známy aj ako ZMQ a pre prepojenie s API ponúka príkazy, ktoré je

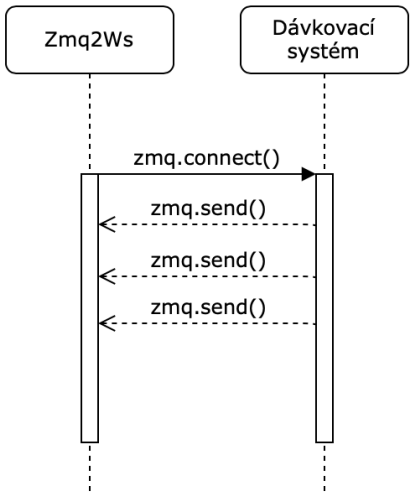
možné zadať do príkazového riadka. Aby bolo možné dáta prijímať na webovom rozhraní, prijaté dáta na sprostredkovateľovi sa odošlú cez protokol WebSocket, pričom tento sprostredkovateľ ponúka WebSocket pripojenie. Ak chce používateľ spustiť úlohu, vytvorí sa HTTP požiadavka, ktorá je odoslaná na API exekútora, ktorý počúva na HTTP požiadavky. Bližší opis spolupráce jednotlivých častí je obsahom nasledujúcej časti.



Obr. 2.1: Komponentový diagram systému

2.1.1 Prepojenie Zmq2Ws a dávkovacieho systému

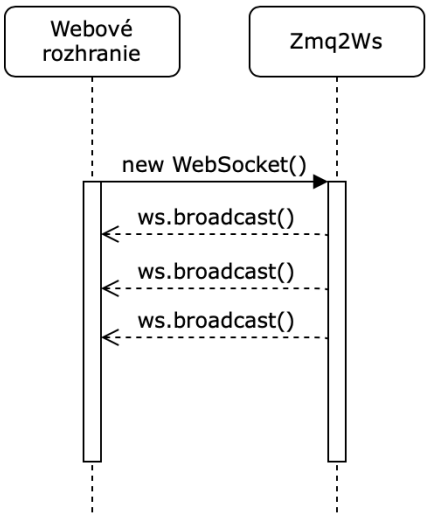
Prepojenie týchto dvoch častí je zabezpečené protokolom ZMQ, pričom sa využíva vzor vydavateľ-odberateľ (*publisher-subscriber pattern*). Výhodou tohto vzoru je fakt, že nie je potrebné žiadať dávkovací systém o nové údaje vždy, keď ich potrebujeme, ale stačí sa raz pripojiť na vydavateľa (v tomto prípade dávkovací systém) a prihlásiť sa na odber údajov v rámci istej skupiny správ, popr. odber všetkých správ. Prepojenie protokolom ZMQ je v dnešnej dobe implementované len v rámci dávkovacieho systému SALSA. Pre prepojenie iných dávkovacích systémov s už existujúcim sprostredkovateľom Zmq2Ws je potrebné doimplementovať špeciálneho prostredníka, ktorý sa postará o zber dát z dávkovacieho systému a následné vyslanie týchto dát pre všetkých odberateľov. Ukážkový sekvenčný diagram, ako komunikujú tieto časti, je možné vidieť na Obr. 2.2.



Obr. 2.2: Sekvenčný diagram prepojenia sprostredovateľa a dávkovacieho systému

2.1.2 Prepojenie Zmq2Ws a webového rozhrania

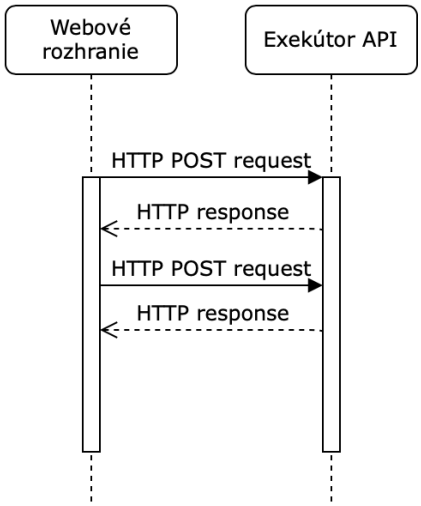
Pri prepojení týchto častí je použitý protokol WebSocket, keďže tento nám poskytuje možnosť posielať údaje zo sprostredkovateľa do webového rozhrania bez toho, aby bolo nutné robiť nové požiadavky. Principiálne sa tento prístup podobá na vzor vydavateľ-odberateľ z predošlej časti, avšak v tomto prípade to nie je tento vzor, keďže samotné webové rozhranie sa nemusí prihlasovať na odber. Údaje odosielané zo sprostredkovateľa sú vo formáte JSON. Ukážkový sekvenčný diagram je na Obr. 2.3.



Obr. 2.3: Sekvenčný diagram prepojenia sprostredovateľa a webového rozhrania

2.1.3 Prepojenie webového rozhrania a API exekútora

Ďalšou dôležitou časťou je prepojenie webového rozhrania a API exekútora, ktorý slúži na spúšťanie úloh. Toto API predstavuje HTTP server, ktorý počúva na požiadavky od používateľa zadané cez webové rozhranie. Ak pošle HTTP GET požiadavku na hlavný koncový bod, tak API mu vráti jednoduchú odpoveď vo forme JSON objektu s informáciou o úspešnej požiadavke. V prípade, že odošle HTTP POST požiadavku na hlavný koncový bod, exekútor najprv overí, či telo požiadavky obsahuje požadované atribúty. V prípade, že ich telo požiadavky neobsahuje, tak bude vrátená chyba s HTTP návratovým kódom 400, čo symbolizuje zlú požiadavku z pohľadu klienta. Ak telo požiadavky obsahuje všetky potrebné atribúty, exekútor sa pokúsi spustiť danú úlohu. Návratový kód úlohy, resp. návratová hodnota príkazu v príkazovom riadku je odoslaná užívateľovi v odpovedi. V prípade, že vznikli problémy pri spúšťaní úlohy, táto chyba sa odošle spolu s návratovou hodnotou. Zjednodušený priebeh komunikácie je možné vidieť na diagrame na Obr. 2.4.

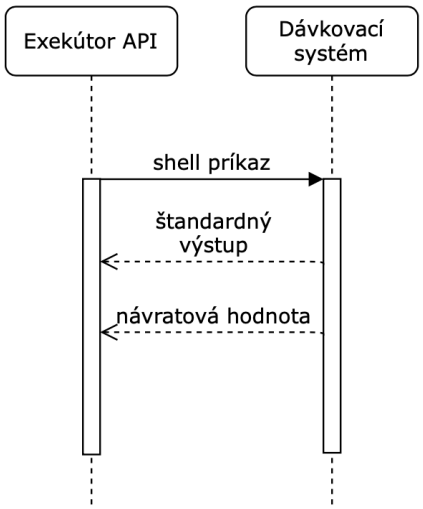


Obr. 2.4: Sekvenčný diagram prepojenia webového rozhrania a API exekútora

2.1.4 Prepojenie API exekútora a dávkovacieho systému

Poslednou časťou riešenia je prepojenie medzi exekútorom a samotným dávkovacím systémom. Toto je jediná komunikácia zo všetkých častí, ktorá neprebieha cez akýkoľvek sieťový protokol. Dôvodom je fakt, že väčšina dávkovacích systémov používa na manipuláciu s ním príkazový riadok. To znamená, že ak chce exekútor spustiť úlohu na dávkovacom systéme, je potrebné vykonať príkaz v príkazovom riadku, pričom príkaz je špecifický pre každý dávkovací systém. V

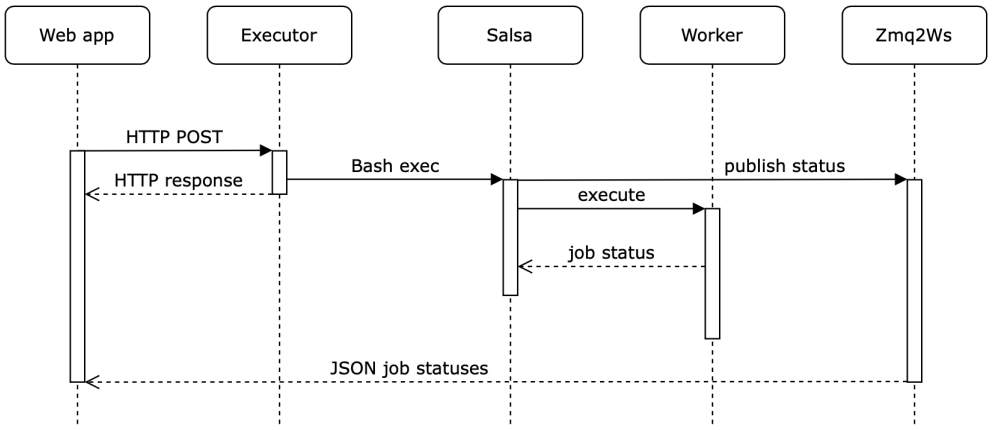
našom riešení je taktiež možné spúšťať úlohy v rámci Kubernetes, čo znamená, že dávkovací systém môže byť umiestnený v klastri Kubernetes a stále bude možné spustiť úlohy. Po vykonaní príkazu v príkazovom riadku získa API štandardný výstup programu (*stdout*) a tiež návratovú hodnotu. Ukážkový sekvenčný diagram komunikácie medzi exekútorom a dávkovacím systémom je možné vidieť na Obr. 2.5.



Obr. 2.5: Sekvenčný diagram prepojenia API exekútora a dávkovacieho systému

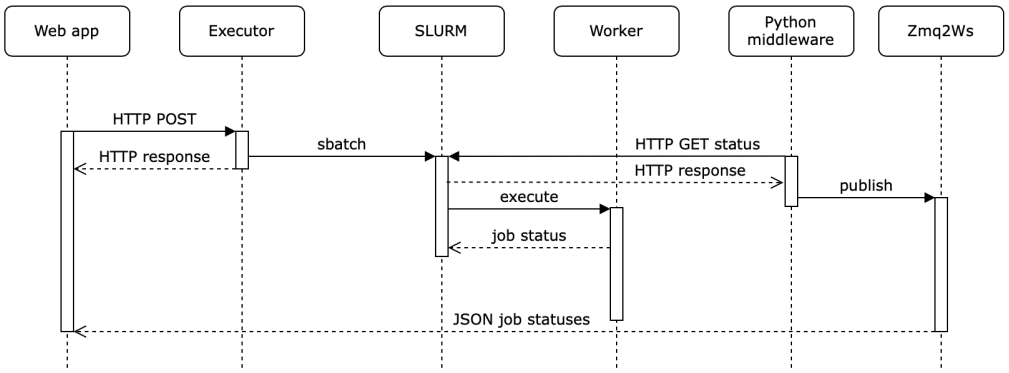
Celkový pohľad na spoluprácu jednotlivých komponentov riešenia ponúka Obr. 2.6, ktorý zobrazuje sekvenčný diagram pre nami vytváraný systém z dávkovacieho systému SALSA, pomocou ktorého je možné riadiť dávkovací systém a získavať z neho aktuálne informácie pomocou klienta. V tomto diagrame sa nachádza 5 častí, pričom prvý zľava predstavuje samotného klienta, teda webovú aplikáciu, druhý predstavuje nami vytvorený exekútor, teda server, ktorý slúži ako prostredník medzi dávkovacím systémom a klientom, ďalej sa tu nachádza dávkovací systém, ďalšou časťou je pracovný uzol a nakoniec je tu sprostredkovateľ Zmq2Ws, ktorý prijíma údaje z klastra cez ZeroMQ a preposiela ich cez WebSocket, pričom aplikuje vrstvu vyrovnávacej pamäte (*cache*). Na začiatku celého procesu je vytvorenie HTTP požiadavky metódou POST, v ktorej sa pošlú potrebné údaje pre spustenie úlohy v dávkovacom systéme na exekútor. Tento exekútor následne spustí daný príkaz v príkazovom riadku, čo zaregistruje dávkovací systém a zároveň spustí danú úlohu na dostupnom pracovnom uzle. Taktiež počas celého času dávkovací systém zverejňuje údaje o všetkých úlohách (*publish*), pričom na druhej strane pasívne počúva na tieto údaje Zmq2Ws (*subscribe*) v prípade dávkovacieho systému SALSA. Posledným krokom je zverejňova-

nie (*publish*) údajov zo Zmq2Ws cez WebSocket pre samotného klienta vo forme JSON objektov.



Obr. 2.6: Sekvenčný diagram systému (dávkovací systém SALSA)

V prípade dávkovacieho systému SLURM je potrebné použiť ešte dodatočný skript, ktorý využíva HTTP REST API SLURM-u, keďže samotný dávkovací systém neposkytuje posielanie údajov pomocou ZeroMQ a tieto posiela ďalej na Zmq2Ws rovnako ako v prípade dávkovacieho systému SALSA. Tento rozdiel je možné vidieť na Obr. 2.7. Pre dávkovací systém DIRAC bude platiť podobné ako pre dávkovací systém SLURM, a teda je potrebné vytvoriť podobný prostredník, pričom DIRAC taktiež obsahuje HTTP REST API.



Obr. 2.7: Sekvenčný diagram systému (dávkovací systém SLURM)

2.2 Prípady použitia

Pre navrhovaný systém bolo identifikovaných niekoľko prípadov použitia, ktoré budú spomenuté v rámci tejto časti práce.

2.2.1 Spustenie úlohy v dávkovacom systéme DIRAC, SLURM alebo SALSA

Autentifikovaný používateľ (pracovník vedeckej organizácie) musí otvoriť aplikáciu vo webovom prehliadači a musí sa pripojiť na sprostredkovateľa pomocou vyplnenia jeho adresy a následného kliknutia na tlačidlo *Connect*. Po pripojení pracovník vyberie klaster, s ktorým chce pracovať a potom zadá úlohu, ktorú chce vykonať do predprivaneného poľa pre zadávanie príkazov. Po zapísaní príkazu používateľ klikne na tlačidlo *Run command*, ktoré odošle príkaz na klaster, kde na následne vykoná.

2.2.2 Kontrola chodu úlohy

Autentifikovaný používateľ (pracovník vedeckej organizácie) otvorí aplikáciu vo webovom prehliadači a pripojí sa na sprostredkovateľa pomocou vyplnenia jeho adresy a následného kliknutia na tlačidlo *Connect*. Po pripojení pracovník vyberie klaster, s ktorým chce pracovať a potom sa mu zobrazí tabuľka s vykonávajúcimi sa úlohami v danom čase. V tejto tabuľke vidí používateľ svoje spustené úlohy, pričom vidí číselné štatistiky toho, koľko úloh je vykonaných, koľko sa vykonáva, koľko je priradených a koľko úloh ešte len čaká na priradenie.

2.2.3 Ukončenie chodu úlohy

Autentifikovaný používateľ (pracovník vedeckej organizácie) otvorí aplikáciu vo webovom prehliadači a pripojí sa na sprostredkovateľa pomocou vyplnenia jeho adresy a následného kliknutia na tlačidlo *Connect*. Po pripojení pracovník vyberie klaster, s ktorým chce pracovať a potom sa mu zobrazí tabuľka s vykonávajúcimi sa úlohami v danom čase. V tejto tabuľke vidí používateľ svoje spustené úlohy, pričom vidí číselné štatistiky toho, koľko úloh je vykonaných, koľko sa vykonáva, koľko je priradených a koľko úloh ešte len čaká na priradenie.

2.3 Nastavenie klastrov

Po opise prípadov použitia je možné prejsť k samotnej implementácii. Prvým krokom však je nastavenie klastrov, keďže tie sú základom pre toto riešenie. V tomto prípade sa toto nastavenie týka dávkovacieho systému (klastru) SLURM. Spôsobov ako nastaviť klaster je hneď niekoľko. Prvým základným spôsobom je manuálne nastavovanie, teda manuálna inštalácia všetkých potrebných balíkov po-

mocou manažéra balíkov (ako napr. APT v operačných systémoch založených na Debiane, YUM, resp. DNF založené na operačnom systéme RHEL alebo *Red Hat Enterprise Linux*), manuálne spúšťanie démonov a služieb pomocou príkazu **systemctl** a tiež povoľovanie pripojení na určité služby cez firewall. Všetkým týmto manuálnym krokom sa dá jednoducho vyhnúť za pomoci rôznych nástrojov na automatizáciu práce na serveroch. Jedným z takýchto nástrojov je aj Ansible, nástroj na IT automatizáciu, ktorý vyvíja spoločnosť Red Hat. Ansible napríklad dokáže spravovať konfigurácie, nasadzovať aplikácie a orchestrovať služby na serveri. Tento nástroj nepoužíva žiadnych špeciálnych agentov, ani vlastnú bezpečnostnú infraštruktúru, pričom vo svojich konfiguráciách používa jazyk YAML, ktorý je pre človeka ľahko čitateľný. Princíp fungovania Ansible je, že tento nástroj sa napája k jednotlivým uzlom štandardne cez SSH a posiela do nich malé programy nazývané *Ansible moduly*. Tieto moduly sú písané ako požadovaný stav všetkých modelov zdrojov. Správcovi, ktorý používa Ansible je umožnené vytvoriť tzv. *Ansible Playbook*, ktorý obsahuje viacero inštrukcií, ktoré sa majú vykonať na daných uzloch. Samozrejmosťou pri nástroji Ansible je možnosť spravovať viacero uzlov v jednom momente, čo môže radikálne zrýchliť nastavovanie veľkého množstva serverov.

2.3.1 Nastavenie klastra SLURM

Pri nastavovaní klastra SLURM je potrebné začať vytvorením používateľa *slurm* a tiež skupiny *slurm*. Vytvorenie používateľa a skupiny je možné vykonať pomocou príkazu **useradd**, resp. **groupadd**. Nasledujúcim krokom je inštalácia balíkov radiča SLURM. V prípade operačných systémov založených na Debiane je potrebné nainštalovať balík **slurm-wlm** a v prípade operačných systémov založených na Red Hat-e je potrebné nainštalovať balíky **munge, slurm** a **slurm-slurmctld**. Po inštalácii balíkov radiča je potrebné vytvoriť priečinok, ktorý bude slúžiť ako úložisko stavu klastra SLURM. Pre tento účel je potrebné vytvoriť priečinok **slurmctld** v **/var/lib/slurm-llnl** v prípade OS založeného na Debiane a **/var/lib/slurm** v prípade OS založeného na Red Hat-e. Taktiež je potrebné nastaviť práva priečinku na 700, čo znamená, že vlastník má práva na všetko. Okrem ukladania stavu je potrebné aj ukladať logy, čo je možné vytvorením súboru **/var/log/slurm-llnl/slurmctld.log** v prípade Debianu. Ďalším krokom je vytvorenie priečinku pre konfiguráciu, pričom táto sa nachádza v **/etc/slurm-llnl** v prípade Debianu, inak v **/etc/slurm**. Po tomto kroku je úspešne nainštalovaný démon **slurmctld**. Nasleduje inštalácia a nastavovanie démona **slurmd**.

Podobne ako v prvom kroku, je potrebné nainštalovať balík **slurm-wlm** v De-

biane, resp. balíky **munge, slurm** a **slurm-slurmctld** v prípade Red Hat. A rovnako je potrebné vytvoriť súbor, ktorý bude slúžiť na logovanie aktivity, pričom cesta k tomuto súboru je rovnaká ako v prípade prvého kroku.

Pre inštaláciu databázy je potrebné nainštalovať balík **slurmdbd** pre Debian a balíky **munge** a **slurm-slurmdbd** pre Red Hat. Taktiež je potrebné pridať konfiguráciu *slurmdbd.conf* do **/etc/slurm-llnl** (Debian) alebo **/etc/slurm** (Red Hat).

2.4 Nasadenie klastra SALSA na Kubernetes

Okrem samotného klastra SLURM je potrebné tiež nastaviť klaster SALSA, pričom tento sme sa pre rýchlejšie nasadenie a správu rozhodli nasadiť na systém Kubernetes, niekedy označovaný ako k8s, pričom číslo 8 symbolizuje 8 písmen medzi prvým a posledným písmenom. Systém Kubernetes slúži na automatické nasadzovanie, škálovanie a manažment kontajnerizovaných aplikácii. Keďže Kubernetes požaduje kontajnerizované aplikácie, prvou úlohou je vytvorenie Docker obrazov klastra SALSA. Pre každú zo služieb v rámci Salsy je vytvorený vlastný obraz, teda *worker*, *redirector* a tiež *discovery*. Po vytvorení obrazov je možné prejsť na samotné vyskladanie konfiguračného súboru pre systém Kubernetes.

Princíp fungovania Kubernetes je taký, že klaster obsahuje jeden alebo viac uzlov, na ktorých sa nachádzajú tzv. *pods*, pričom tieto majú v sebe spustenú inštanciu Docker obrazu, t. j. kontajner. Tieto uzly je potrebné určitým spôsobom riadiť a na to slúži tzv. *control plane*, ktorý riadi a zabezpečuje, že požiadavky, ktoré boli definované v konfiguračnom súbore sú naplnené. Bez akéhokoľvek ďalšieho nastavovania sú pod-y neviditeľné pre ostatné pod-y a tiež nie sú viditeľné mimo Kubernetes klastra. Na to, aby ich bolo možné vidieť z vonku, resp. aby bolo možné komunikovať medzi podmi v rámci klastra, je potrebné nastaviť služby (*services*), ktoré sa starajú o presieťovanie. Medzi najznámejšie typy služieb patria *NodePort* a *ClusterIP*. Prvý menovaný typ slúži na to, aby bolo možné ku danému podu pristúpiť z vonku klastra. Druhý, naopak, slúži na možnosť komunikácie medzi podmi v rámci klastra.

Konfiguračný súbor je písaný v dobre známom jazyku YAML, ktorý je ľahko čitateľný pre ľudí. Prvou časťou v rámci konfigurácie je vytvorenie *Deploymentu*, ktorý služi sa informovanie klastra o tom, ako má vytvoriť pod-y s kontajnermi, v našom prípade potrebujeme *deployment* pre *worker* uzol, ktorý nazveme **slsw**, ďalej potrebujeme vytvoriť *deployment* pre *redirector*, ktorý bude mať názov **slsr** a ako posledný potrebujeme pre *discovery*, pričom jeho názov bude **slsd**. Po tom, ako sme v konfigurácii pripravili *deployments*, môžeme prejsť k nastaveniu služieb,

pretože potrebujeme, aby tieto časti systému SALSA mohli komunikovať medzi sebou. Pre **slsd** potrebujeme, aby bol viditeľný port 40000, na ktorom táto časť počúva. Keďže nepotrebujeme, aby **slsd** bolo viditeľné zvonku, nemusíme definovať typ služby, pretože predvolená možnosť pre typ je *ClusterIP*. V rámci časti redirector potrebujeme zaistiť, aby porty 5001 a 41000 boli viditeľné, pričom port 5001 slúži pre monitoring a 41000 na odosielanie úloh (*submitter*).

2.5 Tvorba knižnice v react-ndmspc

Táto knižnica bude pozostávať z React komponentov, pričom táto technológia je opísaná v časti 1.9 tejto práce. Cieľom tejto časti je opísať postup vytvorenia samotnej knižnice, jej komponentov. *React-ndmspc* je projekt, ktorý slúži ako frontend knižnica pre prácu s dávkovacími systémami pomocou webového rozhrania, pričom je možné ju importovať do iných projektov a používať.

2.5.1 Vytvorenie knižnice

Knižnica **react-ndmspc** je aplikovateľná pre viaceré platformy, medzi ktoré je možné zaradiť napríklad React alebo nástroj Cockpit. Táto knižnica je open-source, čo znamená, že jej kód je verejne dostupný a projekt je dostupný na platforme Gitlab[1].

Základom tohto projektu bolo vytvorenie počiatočnej štruktúry, pričom v tomto nám pomohol nástroj, resp. NPM balík na vytváranie knižníc v React-e, nazývaný *create-react-library*[2]. Výhodou tohto balíka je možnosť použitia *CommonJS*, *ESM* alebo *ES Modules* a tiež vytvorenie ukážkovej stránky pomocou *create-react-app*. Možností, ako použiť tento balík je hneď niekoľko. Prvou možnosťou je globálna inštalácia balíka a druhou okamžité spustenie, resp. použitie balíka pomocou nástroja **npx**, ktorý dokáže spúšťať binárne súbory v rámci balíkov. V tomto prípade sme sa rozhodli pre druhú možnosť, pričom vytvorenie knižnice je možné pomocou nasledujúceho príkazu:

```
npx create-react-library
```

Po spustení tohto príkazu sa v príkazovom riadku objaví výzva na zadanie názvu knižnice, popisu, autora a licencovania. Keď používateľ zadá všetky potrebné údaje, vytvorí sa priečinok s názvom zhodujúcim sa s názvom knižnice. Vytvorený priečinok s knižnicou sa skladá z niekoľkých ďalších priečinkov, kto-

[1] `https://gitlab.com/ndmspc/react-ndmspc`
[2] `https://www.npmjs.com/package/create-react-library`

rých význam bude opísaný nižšie. Okrem toho tu uvádzame aj stromovú štruktúru projektu pre lepšie porozumenie.

```
.
├── Dockerfile
├── README.md
├── config.yml
├── dist
├── example
├── node_modules
├── package-lock.json
├── package.json
├── scripts
└── src
```

Prvým je priečinok `dist`, tento priečinok obsahuje vygenerovanú knižnicu vo forme súborov s koncovkou `.js`, CSS súbory a tiež mapu zdrojov (*sourcemap*). V tomto priečinku sa budú nachádzať vždy najnovšie verzie knižnice, keďže pri vývoji je potrebné vždy vygenerovať knižnicu nanovo a použiť ju v príkladovom projekte.

Ďalším priečinkom je `example`, tento obsahuje práve základnú webovú aplikáciu vytvorenú pomocou React-u (pomocou príkazu `create-react-app`). Tento priečinok má jediný účel, a to priame otestovanie nami vytváranej knižnice bez toho, aby sme ju museli publikovať a testovať alebo prepájať ju s iným projektom pomocou príkazu `npm link`.

Nasleduje najdôležitejší priečinok, a to priečinok `src`, ktorý obsahuje všetky zdrojové kódy knižnice. Po samotnej inicializácii knižnice sa bude v tomto priečinku nachádzať len základný index súbor spolu s mapou zdrojov a CSS štýlmi. Samotný súbor `index.js` obsahuje jedine importy na React a CSS štýly a export jednoduchého komponentu. Väčšinu času sa vývoj bude vykonávať priamo v rámci tohto priečinku.

Okrem týchto spomenutých priečinkov sa v tomto projekte nachádzajú aj súbory, medzi ktorými je `README.md`, klasický súbor napísaný v jazyku Markdown, ktorý slúži zvyčajne na informovanie používateľa knižnice o samotnej knižnici, spôsobe práce s ňou a pod. Keďže sa samozrejme jedná o knižnicu napísanú v jazyku JavaScript, je tu použité prostredie Node.js, a teda sa tu nachádza súbor `package.json` a tiež `package-lock.json`. Tieto súbory disponujú informáciami o danom projekte, o jeho závislostiach a dostupných skriptoch na spúšťanie. Medzi týmito skriptami sa nachádzajú viaceré dôležité, napríklad pre testovanie, vygenerovanie knižnice, vývojárske spustenie knižnice určené pre vývoj a tiež nasadenie. V našom projekte sme sa rozhodli pridať niekoľko ďalších skriptov, ktoré nám vývoj uľahčia, keďže predvolene pre vývojárske spustenie knižnice a spus-

tenie ukážkovej webovej aplikácie sú potrebné dva rôzne terminály. Tento skript sme nazvali `dev`, pričom je možné ho spustiť ako `npm run dev` a slúži na paralelné spustenie knižnice a ukážkového projektu.

Keďže tento projekt je nasadený na platforme Gitlab, rozhodli sme sa využiť možnosti, ktoré nám táto platforma ponúka a pridali sme možnosť spúšťania tzv. *pipeline* pomocou Gitlab CI/CD. Táto funkcia dokáže napríklad vygenerovať knižnicu a poslať ju do NPM registra pre iných používateľov, otestovať a pod. Pre túto funkciu sme taktiež vytvorili priečinok `scripts`, ktorý obsahuje Bash skripty, ktorých funkciou je spustenie, konfigurácia a resetovanie knižnice. Tieto skripty sú práve použité v rámci súboru `.gitlab-ci.yml`, čo je konfiguračný súbor pre Gitlab CI/CD a obsahuje kroky, pričom prvý generuje knižnicu a nasadzuje ju do NPM registra a druhý krok vytvorí z priečinku `example` stránku, ktorá bude nasadená pomocou ďalšej funkcie Gitlab-u a to **Gitlab Pages**. Výsledná stránka je dostupná na tejto adrese[3].

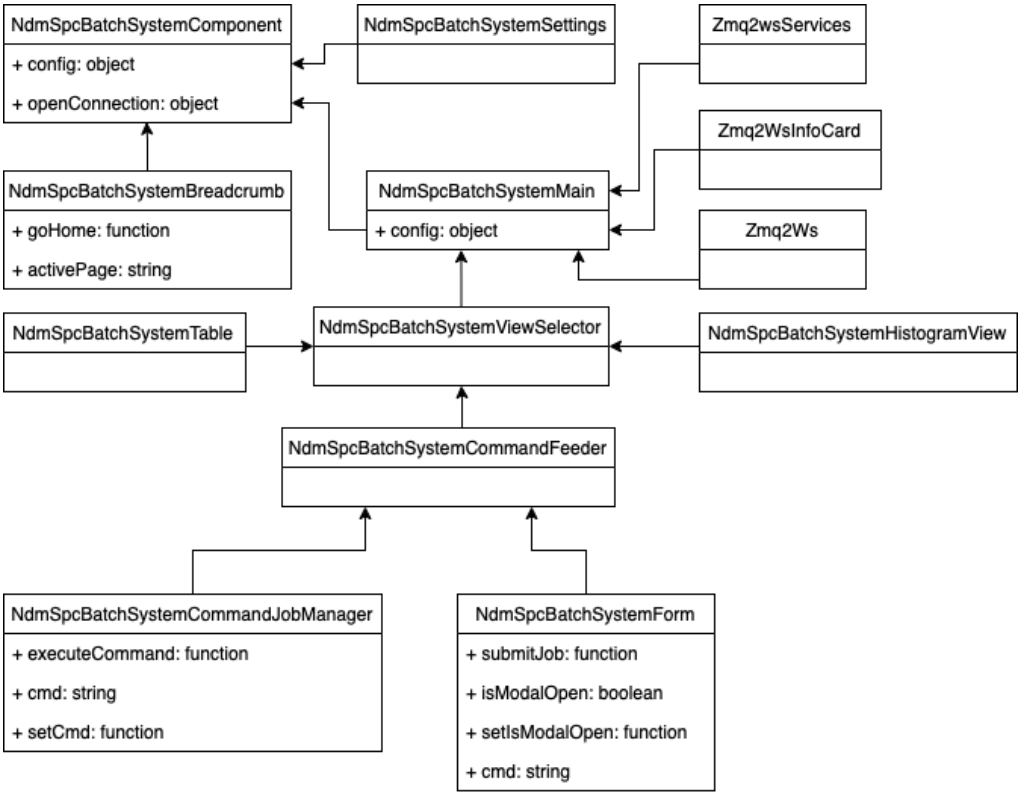
2.5.2 Tvorba komponentov knižnice

Pri vytváraní webovej aplikácie je veľmi dôležité rozhranie, s ktorým bude užívateľ pracovať. Keďže táto webová aplikácia bude nasadená ako plugin do nástroja Cockpit, ktorý využíva návrhový systém nazývaný PatternFly, taktiež vyvíjaný spoločnosťou Red Hat, rozhodli sme sa použiť práve tento systém pre zachovanie konzistencie celého nástroja Cockpit.

Dokumentácia návrhového systému PatternFly veľmi podrobne a jasne zobrazuje všetky kroky inštalácie a nasadenia tohto systému do projektu založenom ja knižnici React. Jedinou požiadavkou pre inštaláciu je mať nainštalovaného správcu balíkov NPM alebo Yarn. V našom prípade použijeme NPM, keďže ten sme použili aj na vytvorenie React projektu. Priamo v dokumentácii je zobrazený príkaz na inštaláciu a teda príkaz `npm install @patternfly/react-core --save` zabezpečí nainštalovanie tohto systému a tiež jeho uloženie medzi závislosti do súboru `package.json`. Ďalším krokom je skopírovanie CSS štýlov PatternFly do projektu a jeho nalinkovanie v hlavnom HTML súbore. Architektúru nami vytváranej knižnice vo forme UML diagramu je možné vidieť na Obr. 2.8. Na tomto obrázku je možné vidieť React komponenty, ktoré budú vytvorené.

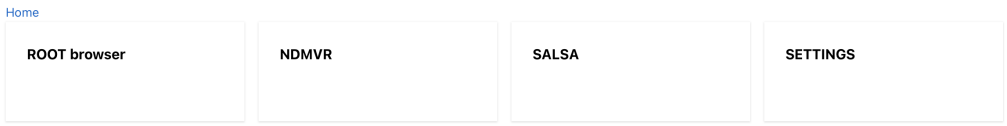
Ďalším krokom je vytvorenie komponentov slúžiacich pre používateľa na prácu s dávkovacím systémom, teda spúšťanie úloh a tiež získavanie údajov o vykonávajúcich úlohách. Pri príchode do webovej aplikácie používateľ bude mať mož-

[3] `https://ndmspc.gitlab.io/react-ndmspc/`



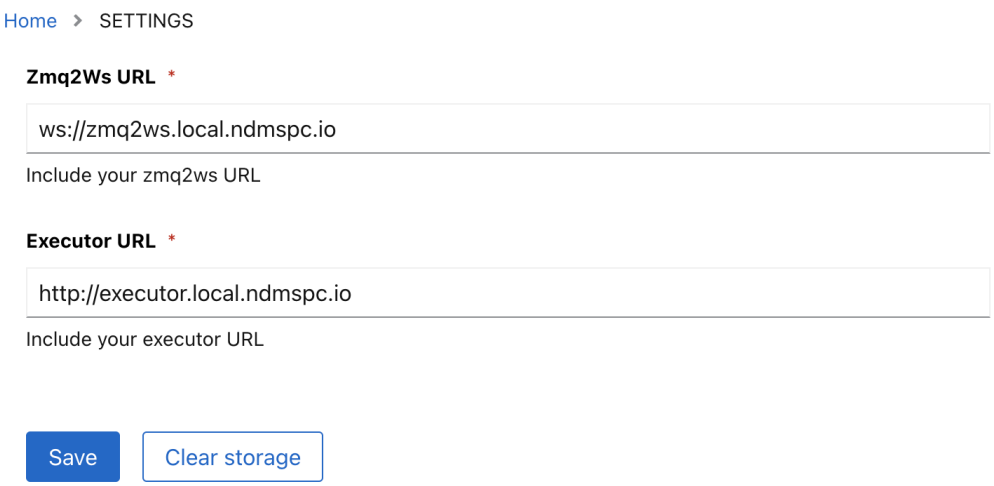
Obr. 2.8: Architektúra vytváranej knižnice

nosť výberu, čo chce s danou aplikáciou robiť, v aktuálnom prípade to budú možnosti pre zobrazenie ROOT prehliadača, spustenie NDMVR, teda prostredia virtuálnej reality, SALSA, čo bude možnosť práce s dávkovacím systémom a tiež možnosť nastavenia adries. Komponent bude obsahovať kartičky, na ktoré bude možné kliknúť, pričom tento komponent sa bude volať `NdmSpcBatchSystemComponent`. Všetky nami vytvorené komponenty pre tento projekt budú zdieľať rovnaký základ mena a to `NdmSpcBatchSystem`, čo zabezpečí, aby bolo v budúcnosti jednoduchšie odlíšiť prípadné iné časti projektu. Názornú ukážku týchto kartičiek je možné vidieť na Obr. 2.9.



Obr. 2.9: Kartičky v komponente NdmSpcBatchSystemComponent

Prvým krokom, pre prácu s dávkovacími systémami, je nastavenie správnych adries. Toto je možné dosiahnuť kliknutím na kartičku **Settings**. Po kliknutí sa zobrazí okno (Obr. 2.10). V rámci tejto kartičky je potrebné nastaviť správne adresy pre sprostredkovateľa Zmq2Ws a tiež pre nami vytvorené API. Po kliknutí na tlačidlo **Save** sa tieto hodnoty uložia do *LocalStorage*, odkiaľ budú ďalej čítané.



Obr. 2.10: Kartička s nastaveniami adries

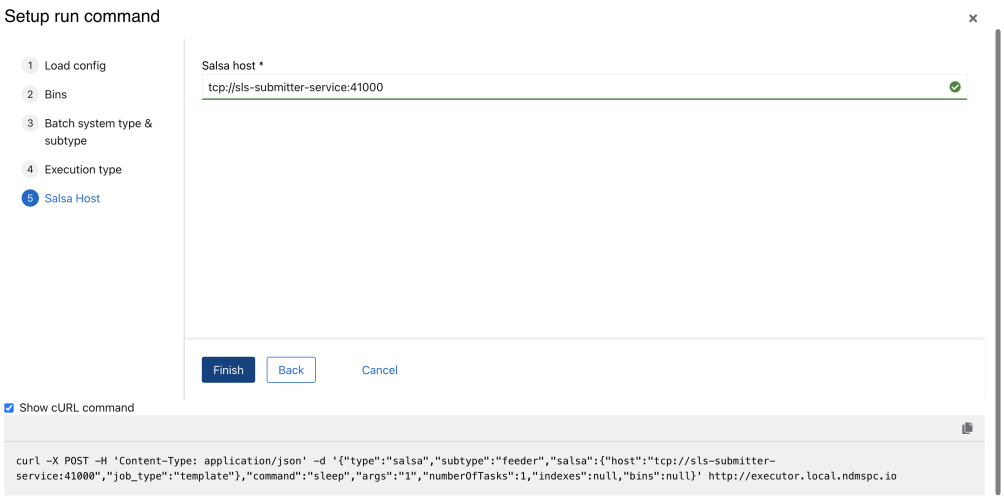
Po kliknutí na kartičku SALSA sa používateľ dostane ku kroku, kedy musí zadať adresu sprostredovateľa spolu s tlačidlom na pripojenie. Toto zadávacie pole sa nachádza v React komponente `NdmSpcBatchSystemMain`. Okrem pripojenia na sprostredkovateľa sa tento komponent stará aj o zvolenie požadovaného dávkovacieho systému, ktorého údaje sú posielané skrz daného sprostredkovateľa. To znamená, že v prípade, ak sa používateľ úspešne pripojil k sprostredkovateľovi, tak sa mu zobrazia záložky (*tabs*) so zdrojmi dát, medzi ktorými si používateľ môže vybrať. Existujú dva úrovne záložiek, čo je možné vidieť na Obr. 2.11, kde bola vybratá záložka **sls** a následne je možné vybrať podzáložku.

Obr. 2.11: Pripojený klient v komponente NdmSpcBatchSystemMain

V rámci tohto komponentu sa nachádzajú ďalšie komponenty, ktoré sú viditeľné až po zvolení dávkovacieho systému. Keďže je potrebné mať informácie ohľadom dávkovacieho systému, komponent `NdmSpcBatchSystemInfoCard` slúži na zobrazenie základných informácií, ako napríklad URL na odosielanie, počet dostupných slotov a tiež dátum a čas poslednej aktualizácie dát. Ďalším komponentom je komponent `NdmSpcBatchSystemViewSelector`.

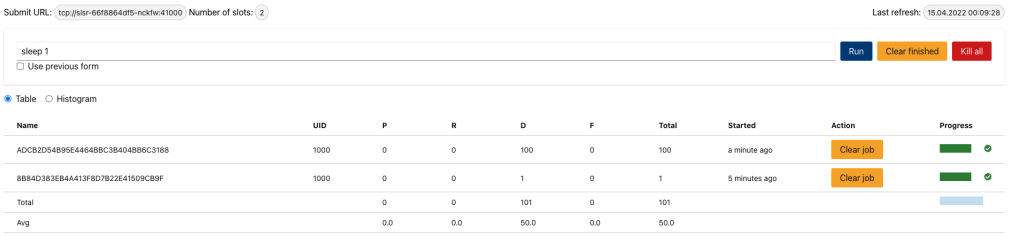
Tento komponent je pre používateľa najdôležitejší, keďže celá manipulácia s dávkovacím systémom sa odohráva v tomto komponente. Podobne ako iné, aj tento komponent je zložený v viacerých komponentov. Prvým komponentom je `NdmSpcBatchSystemCommandFeeder`. Tento komponent sa skladá zo zadávacieho poľa a troch tlačidiel, pričom prvé slúži na zadanie úlohy do dávkovacieho sys-

tému, druhé na odstránenie dokončených úloh a posledné na okamžité ukončenie všetkých úloh. Okrem týchto sa tu nachádza ešte zaškrtávací box, ktorý slúži pre prípad, ak chceme, aby daná úloha bola spustená podľa predošlej konfigurácie. Implementácia týchto tlačidiel nie je predvolene nastavená, keďže táto webová aplikácia je pripravená genericky, čo znamená, že samotný používateľ si bude môcť zadefinovať, akým spôsobom sa budú tieto tlačidlá správať. V zadávacom poli je prednastavená hodnota na ukážkovú úlohu pre dávkovací systém SALSA s príkazom **sleep 1**. Po kliknutí na tlačidlo **Run** a ak nie je zaškrtnuté pole na načítanie predošlej konfigurácie, otvorí sa modálne okno s konfiguráciou, v ktorej sa nastavujú viaceré parametre, medzi ktoré patria typ dávkovacieho systému, spôsob spustenia úlohy, počet vykonaných úloh a i. Toto modálne okno je možné vidieť na Obr. 2.12 tu a tiež je možné získať príkaz cURL, ktorý je možné spustiť v termináli. Toto konfiguračné modálne okno je vytvorené v komponente `NdmSpcBatchSystemForm`.

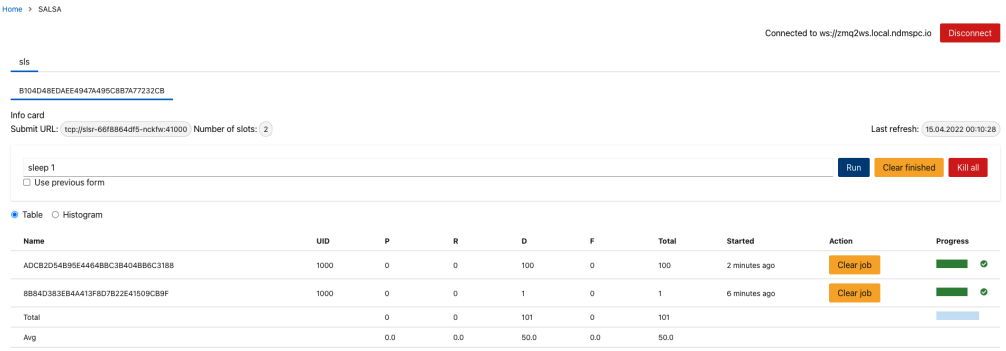


Obr. 2.12: Konfiguračné modálne okno

Druhým obsiahnutým komponentom je `NdmSpcBatchSystemTable`. Ako už názov napovedá, tento komponent obsahuje tabuľku, pričom táto obsahuje viacero dôležitých údajov, medzi ktoré patria *identifikátor úlohy*, *identifikátor používateľa*, *počet čakajúcich, bežiacich, dokončených a neúspešných krokov v rámci úlohy*, *relatívny čas začiatku úlohy*, *tlačidlo na zrušenie úlohy* a *ukazovateľ postupu*. Nami vytvorený komponent `NdmSpcBatchSystemViewSelector` je možné vidieť na obrázku. Výsledný pohľad na celú webovú aplikáciu je možné vidieť na Obr. 2.14.



Obr. 2.13: Komponent NdmSpcBatchSystemViewSelector



Obr. 2.14: Celkový pohľad na webovú aplikáciu

2.5.3 Aplikácia knižnice vo webovej aplikácii

Nástroj Cockpit umožňuje nasadiť vlastné webové aplikácie ako pluginy, čo je využité práve pre účel tohto rozhrania. Keďže sme sa rozhodli vytvoriť rozhranie za pomoci knižnice React, je nám umožnené vytvoriť projekt viacerými možnými spôsobmi.

Prvý spôsob je manuálny, kde je nutné vytvoriť priečinok, inicializovať ho pomocou `npm init -y`, nakonfigurovať Webpack a Babel (transpilér, ktorý zabezpečuje transformáciu kódu jazyka JavaScript s novšou špecifikáciou začínajúcou ECMAScript2015 do staršej verzie, ktorú podporujú webové prehliadače) a samotnou inštaláciou balíkov `react` a `react-dom`.

Druhý spôsob je jednoduchší a hlavne rýchlejší, pričom pri tomto je využitý nástroj určený pre príkazový riadok a volá sa `create-react-app`. Tento nástroj je možné nainštalovať do operačného systému pomocou NPM správcu balíkov alebo je ho možné jednoducho spustiť cez nástroj NPX, ktorý je od určitej verzie nainštalovaný spolu s týmto správcom balíkov. Keďže `create-react-app` bude použité len raz, nie je nutné ho inštalovať ale je možné ho jednoducho spustiť cez druhú spomenutú možnosť, teda NPX. `create-react-app` požaduje ako povinný argument názov projektu, pričom je možné pridať ďalšie možnosti, ako napríklad cieľový priečinok alebo šablónu (to je výhodné pri vytváraní progresívnych webových aplikácií).

Na samotné vytvorenie projektu je teda potrebné zadať a spustiť príkaz:

```
npx create-react-app name
```

Premenná `name` označuje názov projektu, ktorý bude možné vidieť aj v súbore `package.json`. Po spustení tohto príkazu bol vytvorený priečinok s menom projektu, čo je v tomto prípade `name`. Ak chceme otestovať, či je všetko pripravené správne, je možné vstúpiť do tohto priečinku a spustiť v príkazovom riadku príkaz `npm start`, ktorý spustí vývojársky server s možnosťou hot-reloading, čo zabezpečuje, že každá zmena v súbore, sa premietne do prehliadača okamžite. Po úspešnom otestovaní je možné pristúpiť ku samotnému vývoju.

Aby bolo možné použiť nami vytvorenú knižnicu v rámci tohto projektu, je potrebné nainštalovať túto knižnicu z NPM registra, čo je možné vykonať pomocou nasledujúceho príkazu:

```
npm install @npmspc/react-ndmspc
```

Vykonaním tohto príkazu sa okrem stiahnutia tejto knižnice do priečinku so závislosťami **node_modules** pridá daná knižnica do zoznamu závislostí v rámci súboru **package.json**. Spôsob, ako importovať a použiť túto knižnicu je uvedené v nasledujúcom bloku kódu, pričom tento je použiteľný v akomkoľvek projekte, ktorý používa *ES Modules* (*ESM*).

```
import { NdmSpcBatchSystemComponent } from '@ndmspc/react-ndmspc'

const App = () => {
  return (
    <React.Fragment>
      <NdmSpcBatchSystemComponent
        config={config}
        openConnection={openConnection} />
    </React.Fragment>
  );
}
```

Ako je vidno na predchadzájucom bloku kódu, používateľ si môže sám definovať funkciu **openConnection** a objekt **config**, keďže je očakávané, že tieto budú rozdielne medzi projektami. Objekt *config* obsahuje tri funkcie, ktoré korešpondujú s troma tlačidlami v rámci komponentu `NdmSpcBatchSystemViewSelector`, pričom ukážkový obsah tohto objektu je možné vidieť na nasledujúcom bloku kódu.

```
const config = {
```

```
    runCommand: () => window.alert('This is just an example'),
    clearFinished: () => window.alert('This is just an example'),
    killAll: () => window.alert('This is just an example')
}
```

Funkcia **openConnection** slúži na začatie komunikácie s nástrojom Zmq2Ws, ktorý je potrebný na získavanie údajov z dávkovacieho systému. Implementácie tejto funkcie budú taktiež rozdielne medzi rôznymi projektami, a tak sme sa rozhodli nechať implementáciu na samotného používateľa. Ukážkovú implementáciu je možné vidieť nižšie.

```
const openConnection = (
    protocol = 'ws',
    address = 'localhost',
    port = 8442,
    path = '/') => {
    const socket = new WebSocket(`${protocol}://${address}:${port}`);
    socket.parseMessage = data => JSON.parse(data.data)
    return [socket, "websocket"];
}
```

2.5.4 Aplikácia knižnice v Cockpite

Nami vytvráraná knižnica je použiteľná vo viacerých typoch projektov, pričom jedným z nich je napríklad v rámci Cockpitu. O tom, čo je Cockpit, hovorí nasledujúca sekcia.

2.5.5 Použitie knižnice

Prvým krokom po úspešnej inštalácii knižnice React a spustení projektu je nutné vytvoriť symbolickú linku z vygenerovaného projektu do špeciálneho priečinku určeného pre Cockpit. Tento nástroj požaduje, aby bol každý projekt, resp. plugin určený pre Cockpit prelinkovaný do špeciálneho priečinku v rámci systému a to:

```
~/.local/share/cockpit/<meno_projektu>
```

Na to, aby sme úspešne prelinkovali projekt do tohto priečinku je potrebné zadať príkaz počas toho, ako sa nachádzame v priečinku projektu:

```
ln -s `pwd`/dist ~/.local/share/cockpit/<meno_projektu>
```

pričom **meno_projektu** nahradíme za skutočný názov projektu. Okrem toho, `pwd`/dist je priečinok, v ktorom sa nachádza vygenerovaná webová aplikácia,

ktorá má byť nasadená do nástroja Cockpit. V prípade iných rámcov a knižníc ako React sa názov priečinku môže líšiť. V prípade nástroja React je to práve priečinok s názvom **dist**.

2.5.6 Nasadenie knižnice do Kubernetes

Keďže chceme, aby bolo možné celý klaster spustiť jediným príkazom v nástroji Kubernetes, je potrebné vytvoriť Docker obraz. V konfigurácii tohto obrazu je potrebné vygenerovať výslednú aplikáciu (v tomto prípade vygenerujeme aplikáciu dostupnú v priečinku `example`). Po vygenerovaní výsledného kódu aplikácie je tento kód prekopírovaný do servera NGINX, ktorý bude poskytovať túto stránku.

Po vytvorení obrazu je potrebné vytvoriť konfiguráciu pre Kubernetes, presnejšie je potrebné vytvoriť nasadenie a službu (*Deployment*, *Service*). Nasadenie slúži na vytvorenie šablóny, podľa ktorej budú vytvárané pody, ktoré v sebe obsahujú spustené kontajnery Docker obrazov. V tomto prípade používame nami vygenerovaný Docker obraz, pričom ten je dostupný v registri kontajnerov (*Container registry*) v rámci platformy Gitlab. Pri vytváraní šablóny je potrebné definovať okrem Docker obrazu tiež porty, ktoré budú dostupné z iných podov, poprípade z okolia Kubernetes klastra. V tomto prípade je to port 80, čo je klasický port pre protokol HTTP. Služba, na druhej strane poskytuje práve možnosť presieťovať viaceré pody, resp. povoliť pripojenie na daný port v rámci podu z vonkajšieho prostredia klastra.

Okrem týchto dvoch častí je potrebné nastaviť pre túto webovú aplikáciu ešte tzv. vstup (*Ingress*), ktorý umožní pristupovať k cestám webových služieb v rámci klastra z hosťovského počítača. Pre základnú cestu (`/`) na porte 80 sme sprístupnili našu webovú aplikáciu cez jej danú službu (*service*). Okrem toho, sme ingress použili aj ako obrátené proxy (*reverse proxy*), čo má za následok fakt, že môžeme pristupovať ku viacerým službám v rámci klastra. Cez toto proxy je potrebné preniesť sprostredkovateľa Zmq2Ws, ktorý bude dostupný na porte 8442 a tiež API, ktoré je spomenuté nižšie na porte 41001. Aby bolo možné ľahšie rozoznať rôzne služby, konfigurácia ingress ponúka aj možnosť nastavenia názvu hostiteľa. Po nastavení týchto mien je možné pripísať v súbore `/etc/hosts` tieto názvy, aby ukazovali na adresu `localhost`, resp. `127.0.0.1`.

2.6 Tvorba API pre spúšťanie úloh

Po tom, ako sme vytvorili knižnicu pre použitie v rôznych prostrediach, je nutné pokračovať ďalšou časťou práce, a to vytvorením aplikačného programového ro-

zhrania pre spúšťanie úloh, ktorý v našom prípade bude nazývaný exekútor. V rámci vyobrazenej architektúry na Obr. 2.1 je to komponent **Exekútor API**. Aplikačné programové rozhranie, tiež známe skrátene ako API, je potrebné na to, aby sme mohli z webového rozhrania posielať úlohy, ktoré majú byť vykonávané na klastri. Keďže našou úlohou je tiež podporovať viacero typov klastrov, je potrebné zabezpečiť správne prepojenie medzi nimi a webovou aplikáciou. V tejto časti práce budú opísané niektoré známe rámce na tvorbu API.

2.6.1 Vytvorenie servera

Prvým krokom je vytvorenie repozitára pre kód tohto servera. Rozhodli sme sa použiť platformu Gitlab[4], keďže poskytuje viaceré služby, ktoré sú pri vývoji užitočné. Pri vytváraní servera je samozrejme potrebné taktiež premyslieť, ktoré koncové body (**endpoints**) bude tento server potrebovať počúvať a tiež ktoré HTTP metódy je potrebné pripraviť. Keďže cieľom je primárne spúšťanie úloh, je potrebné nejakým spôsobom odoslať údaje na server z webovej stránky, poprípade z prostredia virtuálnej reality. Podľa špecifikácie HTTP je najrozumnejšie použiť metódu POST, ktorá slúži na odosielanie dát a zvykne sa používať na vytváranie objektov. Po zvolení metódy je potrebné navrhnúť spôsob a formu dát, ktoré budú odosielané z webovej aplikácie či prostredia VR. Dôležitými údajmi, ktoré je potrebné odoslať sú **typ**, **podtyp**, **príkaz** a **argumenty**. Tieto štyri údaje sú základné, keďže požadujeme, že bude možné spúšťať úlohy na viacerých typoch dávkovacích systémov. Ďalším údajom, ktorý bude možné posielať, je **run**, pričom tieto budú slúžiť na to, aby bolo možné spúšťanie príkazov v rámci kontajnerov, resp. v rámci Kubernetes, napríklad pomocou nástroja Docker alebo Podman, resp. pomocou príkazu `kubectl`. Pre validáciu, že používateľ zadá všetky údaje, je možné použiť typovanie v rámci jazyka Python pomocou modulov **typing** a **pydantic**.

```
class Pod(BaseModel):
    name: str
    image: str
    labels: str

class Run(BaseModel):
    type: str
    pod: Pod

class Salsa(BaseModel):
```

[4] `https://gitlab.com/ndmspc/api`

```
    host: str
    job_type: str # either template or command (-t or -c)

class Dirac(BaseModel):
    host: str

class RequestBody(BaseModel):
    type: str
    subtype: str
    command: str
    args: str
    salsa: Optional[Salsa] = None
    dirac: Optional[Dirac] = None
    run: Optional[Run] = None
    bins: Optional[List[dict]] = []
    numberOfTasks: Optional[int] = 1
    indexes: Optional[str] = '1'
```

V prípade, že používateľ nepošle všetky požadované údaje, FastAPI mu okamžite pošle HTTP chybu 400, čo značí, že klient nedodal niektoré údaje, pričom FastAPI mu dokonca zašle informáciu o údajoch, ktoré neboli odoslané. Nami vytváraný server požaduje v tele požiadavky údaje v tvare modelu **RequestBody**. V rámci tohto modelu je atribút *run*, ktorý je zadefinovaný pomocou modelu **Run**. Tento je použitý v prípade, že používateľ chce spustiť danú úlohu v rámci Kubernetes. V rámci tohto modelu je potrebné zaslať údaj typu, ktorý bude označovať, či je potrebné vytvoriť vlastný konfiguračný súbor (*k8s*) alebo priame zavolanie v príkazovom riadku (*k8s-cmd*). Druhým atribútom je *pod*, ktorý je zadefinovaný modelom s názvom **Pod**, pričom tento obsahuje informácie o použitom pode, v našom prípade názov podu, použitý Docker obraz a tiež štítky. V rámci základného modelu **RequestBody** sú zadefinované aj ďalšie voliteľné atribúty, pričom *bins* slúži pre používateľa na odoslanie viacerých úloh, kde používateľ vie vytvoriť šablónu pre úlohu. To je možné dosiahnuť špeciálnou syntaxou ${...}, kde ... bude názov premennej, ktorú používateľ zadefinuje v rámci atribútu **bins**. Atribút **numberOfTasks** slúži na definovanie počtu úloh na vykonanie a atribút **indexes** slúži pre dávkovací systém SLURM, kde bude označovať indexy v prípade, že úloha bude spúšťaná ako pole. V našom prípade sme testovali funkcionalitu nášho servera pomocou rozšírenia pre editor VSCode nazývaný **Thunder Client**. Pre parsovanie tela požiadavky sme sa rozhodli vytvoriť špeciálnu triedu **Executor**, ktorá rozparsuje získané údaje a pripraví samotný príkaz, popr. príkazy na

spustenie. Na rozparsovanie požiadavky slúži metóda **parse** a na získanie samotného príkazu je vytvorená metóda **execute**.

Po vytvorení API servera je potrebné ďalej vytvoriť Docker obraz, keďže chceme, aby tento server mohol byť spustený v rámci klastra Kubernetes. Na to, aby bolo možné vytvoriť Docker obraz je potrebné, aby existoval konfiguračný súbor **Dockerfile**. Tento súbor obsahuje inštrukcie, ktoré vedú k samotnému vytvoreniu obrazu. Keďže potrebujeme, aby v rámci Docker obrazu exekútora boli binárne spustiteľné súbory pre dávkovací systém SALSA a SLURM, bolo potrebné nainštalovať balíky pre tieto systémy. Dávkovací systém SALSA má vlastný vygenerovaný Docker obraz, čo je možné použiť tak, že nami vytvorený obraz bude použitý ako základný. Okrem toho je potrebné nainštalovať balík **slurm**, ktorý je možné nainštalovať pomocou príkazu `dnf install slurm`. FastAPI rámec je založený na jazyku Python, a tak je potrebné, aby bol aj Python verzie 3 dostupný. Ten je možné nainštalovať pomocou príkazu `dnf module install python39`. V tomto prípade, nie je potrebné dodatočne vytvárať virtuálne prostredie, keďže samotný obraz bude obsahovať len potrebné moduly. V rámci Docker obrazu je potrebné prekopírovať len tri súbory, pričom prvým je **requirements.txt**, ktorý obsahuje zoznam modulov, ktoré sú potrebné pre chod servera, **server.py** a **executor.py**, pričom prvý menovaný je samotný server a druhý je nami vytvorená trieda *Executor*. Po vytvorení konfiguračného súboru je možné otestovať funkčnosť vygenerovaním daného obrazu. Takého generovanie je možné uskutočniť pomocou príkazu:

```
docker build -t <name> .
```

pričom `<name>` je potrebné nahradiť za meno obrazu.

Po overení, že konfiguračný súbor je správny a samotný obraz je vygenerovaný správne, je možné nastaviť automatické vygenerovanie Docker obrazov v rámci Gitlab CI/CD, kde samotná platforma Gitlab dokonca ponúka možnosť *Container registry*, kde je možné nami vygenerované obrazy uschovať. Keďže nepotrebujeme, aby sa obrazy vytvárali po každom nahratí nového kúsku kódu, ale len po vytvorení novej značky (*tag*), je možné v rámci konfiguračného súboru *.gitlab-ci.yml* použiť atribút **only** a nastaviť ju na možnosť **tags**. Okrem toho je tiež možné spúšťať *pipeline* manuálne, čo je možné dosiahnuť pridaním atribútu **when: manual** v rámci kroku.

V tomto kroku už máme pripravené generovanie Docker obrazov v rámci Gitlab CI/CD a tým pádom ich môžeme zahrnúť do konfigurácie pre klaster Kubernetes, kde ich bude možné spustiť. V rámci našej konfigurácie pre Kubernetes používame kustomizáciu (*Kustomize*), čo je nástroj v rámci Kubernetes, ktorý

povoľuje meniť samotnú konfiguráciu klastra Kubernetes. V rámci tohto nástroja máme niekoľko konfigurácií, ktoré slúžia na vytvorenie nasadení (*deployments*) a služieb (*services*). Nasadenia slúžia na automatickú tvorbu podov, teda spustiteľných Docker kontajnerov, ktoré bežia na pracovných uzloch. Služby, na druhej strane, sú časťou klastra, ktorá dovoľuje sieťovanie medzi podmi a vonkajším svetom mimo klastra. Pre konfiguráciu nášho exekútora sme vytvorili vlastný YAML konfiguračný súbor, v ktorom sme zadefinovali, ako má byť nasadený tento server a tiež, aké porty majú byť dostupné. Nasadenie obsahuje šablónu, podľa ktorej sa budú vytvárať pody, pričom tam je definované meno podu a tiež kontajner, ktorý bude použitý, pričom v tomto prípade to bude kontajner z Gitlab repozitáru projektu nášho API a bude mať dostupný port 3000, na ktorom tento FastAPI server počúva.

2.6.2 Nasadenie rozhrania na Kubernetes

Podobne ako v prípade webovej aplikácie, aj pri API sme sa rozhodli pre nasadenie do klastra Kubernetes. Samozrejme, prvou úlohou bolo vytvorenie Docker obrazu. V našom prípade sme vytvorili obraz, ktorý je založený na Docker obraze dávkovacieho systému SALSA, pričom sme tam doinštalovali balíček pre dávkovací systém SLURM, príkaz `sudo` a tiež Python verzie 3.9. Po vytvorení je potrebné taktiež vytvoriť konfiguráciu pre nasadenie a službu. Rozdiel medzi týmto rozhraním a webovou aplikáciou spomenutou vyššie je jedine zmenený port pri definovaní šablóny.

2.7 Tvorba sprostredkovateľa pre SLURM

Keďže cieľom tohto používateľského rozhrania je ponúknuť používateľovi možnosť spúšťať a monitorovať úlohy na rôznych typoch dávkovacích systémov, je potrebné pripraviť viacerých sprostredkovateľov. Ako je možné vidieť na architektúre zobrazenej na Obr. 2.1, prepojenie medzi dávkovacím systémom a sprostredkovateľom Zmq2Ws prebieha pomocou protokolu ZeroMQ. Z toho dôvodu je potrebné vytvoriť dodatočných sprostredkovateľov, kde každý sprostredkovateľ obsahuje funkcionalitu na správnu prácu s daným dávkovacím systémom, keďže niektoré dávkovacie systémy nepodporujú protokol ZeroMQ. To je prípad dávkovacieho systému SLURM. Pri implementácii tohto sprostredkovateľa sa použije návrhový vzor Adapter, ktorý zabezpečí to, že z pohľadu Zmq2Ws bude možné používať rovnaké volania funkcií. Sprostredkovateľ bude vytvorený v jazyku Python, pričom bude použitá technológia ZeroMQ (ZMQ), kde ZMQ slúži

na spojenie sprostredkovateľa s dávkovacím systémom. Týmto spôsobom bude možné preposielať dáta z dávkovacích systémov do už existujúceho sprostredkovateľa Zmq2Ws, pričom bude taktiež možné vytvoriť medzivrstvu, ktorá môže slúžiť ako vyrovnávacia pamäť (*caching*), čo znamená, že nebude potrebné sa zakaždým pripájať na dávkovací systém. Pri tomto dávkovacom systéme je možné použiť nimi vytvorené REST API, ktoré je voľne dostupné a je volané pomocou HTTP požiadaviek. Toto aplikačné rozhranie je ponúkané cez SLURM démona nazývaného **slurmrestd**. Prístup ku zdrojom klastra SLURM cez SLURM REST API je potrebné získať token, pričom dokumentácia SLURM tiež hovorí, že toto rozhranie nie je zabezpečené proti rôznym útokom, keďže celá komunikácia prebieha cez nezabezpečený protol HTTP. Dôrazne odporúčajú zaobaliť komunikáciu s TLS verziou aspoň 1.3. Taktiež by mala byť monitorovaná sieťová premávka a zablokovať každého klienta, ktorý sa pokúša dostať cez prihlásenie. SLURM ponúka možnosť generovať tokeny pre používateľov pomocou príkazu **scontrol**, pričom tieto tokeny je potrebné generovať po určitej dobe, pretože ak by sa stalo, že tieto tokeny by platili natrvalo, tak by sa mohlo stať, že tento token unikne a môže byť použitý na ďalšie prihlasovanie. Druhou možnosťou, ktorú SLURM ponúka na autorizáciu, je využitie SSO (*Single Sign On*) poskytovateľa. V našom prípade sme sa rozhodli použiť JWT tokeny.

3 Vyhodnotenie

Ako cieľ tejto práce bolo vytvoriť možnosť preposielať údaje medzi dávkovacím systémom a klientom, pričom tento klient môže byť ľubovoľného typu. V tejto práci sme sa zaoberali klientom ako webovou aplikáciou a tiež prostredím virtuálnej reality. Okrem toho, mala táto práca za úlohu zabezpečiť, že ako dávkovací systém môže byť použitý ľubovoľný systém a nie len jeden. Tento projekt je aktuálne pripravený na prácu s dávkovacím systémom SALSA a systémom SLURM. V prípade ďalších dávkovacích systémov je možné jednoducho vytvoriť skript v ľubovoľnom jazyku, pričom my sme si zvolili jazyk Python, ktorý zabezpečí prenos údajov medzi dávkovacím systémom a sprostredkovateľom Zmq2Ws. Veľkou výhodou dávkovacieho systému je, že tento disponuje možnosťou použiť HTTP REST API, ktorý je možné jednoducho použiť v takomto skripte. Najlepším spôsobom ako vyhodnotiť túto prácu je pomocou overenia naplnenosti požiadaviek, ktoré sú kladené na tento systém.

3.1 Požiadavky

Ako bolo spomenuté vyššie, na overenie správnosti projektu či systému, je potrebné vykonať overenie naplnenosti požiadaviek. Nasledujúca časť rozdeľuje tiež požiadavky na splnené, teda tie, ktoré sa očakávali, že budú v tomto čase naplnené a tiež na nasledujúce alebo tiež budúce požiadavky, pričom tieto sú zväčša požiadavky, ktoré mohli vzniknúť počas samotného vývoja na projekte a neboli zahrnuté pred samotnou prácou.

3.1.1 Splnené požiadavky

Túto prácu je možné rozdeliť do dvoch častí, pričom prvá časť je o vytvorení webovej aplikácie, pomocou ktorej je možné spúšťať úlohy v dávkovacom systéme a je ju možné nasadiť napríklad do nástroja Cockpit a druhou časťou je vytvorenie prepojenia medzi touto aplikáciou (a tiež prostredím virtuálnej reality) s dávkovacím systémom.

V prvej časti, teda vytvorenia webovej aplikácie, sa naskytlo niekoľko požiadaviek. Prvou požiadavkou je samozrejme možnosť spúšťania úloh na akomkoľvek dávkovacom systéme priamo z prostredia webového prehliadača. Táto požiadavka bola splnená, keďže samotná aplikácia vytvára a odosiela HTTP požiadavky s údajmi o úlohe a použitom dávkovacom systéme do nami vytvoreného exekútora. Používateľ môže spustiť akúkoľvek úlohu zadaním do príslušného poľa pre zadávanie a následným kliknutím na spustenie úlohy. Okrem možnosti spúšťania má používateľ možnosť úlohy predčasne ukončiť (zabitím procesu) použitím tlačidla a tiež možnosť vymazať všetky dokončené úlohy.

Druhou požiadavkou bol opačný smer, a teda sprístupniť informácie o práve vykonávaných úlohách v tejto aplikácii. To bolo taktiež splnené, keďže dávkovací systém zdieľa (*publish*) údaje o stave cez knižnicu ZeroMQ, ktoré prechádzajú cez sprostredkovateľa Zmq2Ws a on tieto údaje zdieľa (*publish*) cez protokol WebSocket, ktorý je vhodný pre použitie práve s webovými aplikáciami. Používateľovi sa predvolene zobrazí tabuľka, ktorá obsahuje dôležité informácie pre používateľa, ako napríklad identifikačné číslo úlohy, používateľský identifikátor používateľa, ktorý úlohu spustil, priebeh vykonávania úlohy, počet čakajúcich, vykonávaných, dokončených, zrušených a neúspešných úloh. V prípade, že daný používateľ je zadávateľom tejto úlohy, môže túto úlohu aj predčasne ukončiť.

Druhá časť práce poukazuje na vytvorenie možnosti presúvať dáta, kde bolo potrebné vytvoriť exekútora, ktorý dokáže transformovať HTTP požiadavky na spúšťanie úloh v rámci viacerých dávkovacích systémov.

Prvou požiadavkou v tomto prípade bolo, aby používateľ mohol priamo spúšťať úlohy z aplikácie a prostredia virtuálnej reality. Táto požiadavka je splnená, keďže bol vytvorený API server v rámci FastAPI, ktorý počúva na používateľom definovanom porte. Po každom vytvorení HTTP požiadavky metódou POST, je zavolaná funkcia, ktorá určí, o ktorý dávkovací systém sa jedná a následne pripraví špecifický príkaz, ktorý je určený na spustenie v príkazovom riadku. Ten následne pridá danú úlohu do fronty pre dávkovací systém na vykonanie. Okamžite po vykonaní príkazu v príkazovom riadku sa očakáva návratová hodnota 0, čo znamená, že spustenie prebehlo v poriadku. Ak by po spustení prišla návratová hodnota iná ako 0, teda to indikuje akúkoľvek chybu, klientovi sa odošle status s chybou, pričom sa mu odošle taktiež daná návratová hodnota. To je užitočné napríklad v prípade, že používateľ urobí preklep v rámci názvu príkazu v úlohe, pričom v takomto prípade sa vyskytne návratová hodnota 127, symbolizujúca chybu, že daný príkaz resp. program neexistuje.

Druhou požiadavkou je možnosť použitia iného dávkovacieho systému. To je

taktiež splnené, keďže v takom prípade je potrebné zistiť, aké možnosti daný dávkovací systém ponúka. Ak napríklad ponúka REST API, je možné vytvoriť jednoduchý skript, podobný ako v prípade dávkovacieho systému SLURM, ktorý bude vytvárať požiadavky na dávkovací systém a následne bude odosielať získané dáta ďalej klientovi. V prípade dávkovacieho systému, ktorý neponúka takéto API, je potrebné dodatočne vytvoriť skript, resp. program, ktorý dokáže získať dáta spôsobom, ktorým daný dávkovací systém disponuje a zaslať tieto pomocou knižnice ZeroMQ do sprostredkovateľa Zmq2Ws. V tomto prípade nezáleží, v akom programovacom jazyku bude daný program vytvorený, záleží len na schopnosti odosielať údaje pomocou ZeroMQ.

Poslednou požiadavkou je, aby tento projekt zvládal veľkú záťaž a mal vysokú mieru dostupnosti. To je taktiež splnené, keďže nami vytvorený exekútor je bežný Python server s použitým FastAPI rámcom, čo znamená, že je možné spustiť viacero inštancií tohto exekútora a je tak vhodný na nasadenie napríklad v rámci klastra Kubernetes alebo pomocou orchestračného nástroja docker-compose, v tomto prípade je tiež možné kontajnerizovať aplikáciu, čo tiež môže pomôcť bezpečnosti celého systému, keďže samotná aplikácia nebude mať prístup ku hosťovskému súborovému systému, a to znamená, že aj keby sa stalo, že sa útočníkom podarí nabúrať sa do tohto exekútora, získajú len prístup ku obmedzenému súborovému systému, ktorý sa nachádza v rámci daného kontajnera.

3.1.2 Možné nasledujúce požiadavky

V tejto časti je dobré spomenúť viaceré požiadavky, ktoré boli zmienené počas samotného vývoja a neboli splnené v rámci tejto práce, no je možné ich implementovať v budúcnosti.

Požiadavkou na webovú aplikáciu, ktorú je možné v budúcnosti implementovať, je možnosť využitia rôznych UI knižníc, teda aby samotná aplikácia nebola viazaná len na jednu knižnicu. V aktuálnej práci je použitá knižnica PatternFly, no v budúcnosti môže byť užitočná zmena na inú knižnicu, napríklad na MaterialUI, Semantic UI, Ant Design a i. V rámci webovej aplikácie by tiež bolo vhodné navrhnúť určitý spôsob načítania úloh zo súboru a tiež vytvorenie histórie úloh, ktoré boli spustené. To by bolo užitočné, ak by daný používateľ spúšťal rovnaké alebo veľmi podobné úlohy repetitívne. Okrem týchto požiadaviek by bolo vhodné mať aj jednoduchú sekciu pre odosielanie správ, čo by sa hodilo v prípade, ak by nastal akýkoľvek problém, ktorý nie je daný používateľ schopný vyriešiť sám a potreboval by pomoc.

V rámci nami vytvorenej knižnice to môže byť napríklad implementácia na-

hratia konfigurácie zo súboru, čo by znamenalo, že používateľ bude mať pripravený súbor napísaný v jazyku napr. JSON (*JavaScript Object Notation*), zlepšenie samotného modálneho okna s konfiguráciou.

3.2 Testovanie projektu

Samozrejmosťou vo fáze tvorby každého produktu, systému či aplikácie je fáza testovania. Táto patrí medzi najdôležitejšie, keďže pri nej sa zisťujú akékoľvek nedostatky či chyby. V našom prípade je možné projekt testovať viacerými spôsobmi.

Knižnicu *react-ndmspc* je možné otestovať priamo v rámci vygenerovanej webovej aplikácie pomocou **Gitlab Pages**[1]. Táto stránka obsahuje vždy najnovšiu verziu knižnice, ktorá je na portáli Gitlab. Druhou možnosťou ako otestovať knižnicu je naklonovanie projektu a jeho lokálne spustenie. Projekt je možné naklonovať rovnako z portálu Gitlab[2]. Poslednou možnosťou je otestovanie pomocou klastra Kubernetes, pričom pre vytvorenie lokálneho klastra je možné použiť nástroj **kind**. Na stiahnutie konfigurácie pre klaster je možné naklonovať projekt z Gitlab-u[3] a následne v termináli spustiť nasledujúci príkaz:

```
./install.sh
```

Testovaniu sa podrobilo niekoľko osôb, prevažne z vedeckého prostredia, ktorí budú túto knižnicu aj reálne využívať. Cieľom bolo otestovať funkcionalitu pre dávkovací systém SALSA a dávkovací systém SLURM. Používatelia mali za úlohu spustiť niekoľko úloh na týchto systémoch s rôznymi konfiguráciami. Následne im bola zadaná úloha odstrániť dokončené úlohy a tiež odstrániť všetky úlohy. Ďalšou otázkou na používateľov bolo, či je zrozumiteľná tabuľka so stavom všetkých úloh v rámci klastra.

Z odpovedí používateľov vyplýva, že knižnica, resp. webová aplikácia je implementovaná správne, jej funkcionalita zabezpečuje všetky atribúty, ktoré títo používatelia pri svojej práci potrebujú. Pripomienkou však bolo, či je alebo bude možné v budúcnosti pridať podporu ďalších dávkovacích systémov. Odpoveďou na túto otázku či pripomienku je, že pridanie podpory pre ďalší dávkovací systém je jednoduché, je potrebné len mierne upraviť naše API pre spúšťanie úloh a tiež prepojiť sprostredkovateľa Zmq2Ws s daným dávkovacím systémom. V prípade, že daný dávkovací systém nepodporuje zasielanie údajov cez ZeroMQ, je

[1] http://ndmspc.gitlab.io/react-ndmspc
[2] https://gitlab.com/ndmspc/react-ndmspc
[3] https://gitlab.com/ndmspc/user/-/tree/master/k8s/ndmspc

potrebné vytvoriť jednoduchého prostredníka (*middleware*), ktorý poskytne prepojenie medzi týmito dvoma časťami.

3.3 Prednosti projektu

Keďže v tejto dobe neexistuje priamy konkurent tohto projektu, ktorý by disponoval rovnakými alebo aspoň podobnými vlastnosťami, je možné porovnať len niektoré aspekty tejto práce. V prípade webovej aplikácie existuje niekoľko podobných projektov, pričom žiaden z nich nedisponuje možnosťou ovládať dávkovací systém. V prípade dávkovacieho systému SLURM existuje projekt **slurm-web** [4], ktorý slúži ako frontend panel pre používateľov, administrátorov, kde sa nachádzajú všetky potrebné informácie ohľadom dávkovacieho systému. Ako už bolo spomenuté, tento projekt dokáže len sprostredkovať údaje pre používateľov a nedokáže spúšťať úlohy. Taktiež nie je možné ho využiť pri iných dávkovacích systémoch. Pre dávkovacie systémy DIRAC a SALSA neexistuje takéto rozhranie. Najbližším podobným rozhraním je TUI, ktorý je poskytovaný v rámci dávkovacieho systému SALSA. Práve kvôli týmto dôvodom je možné povedať, že prednosťou tejto webovej aplikácie je fakt, že je možné pomocou nej získavať údaje a tiež spúšťať úlohy nezávisle od použitého dávkovacieho systému.

Na druhej strane, v prípade exekútora je možné povedať, že pridanie nového dávkovacieho systému doň je veľmi jednoduché, pričom je potrebné len vytvoriť šablónu, ktorá bude slúžiť ako predloha pre príkazy pre daný dávkovací systém. V praxi to znamená, že ak používateľ požaduje, aby webová aplikácia alebo prostredie virtuálnej reality bolo schopné spúšťať úlohy na inom dávkovacom systéme, je potrebná len nepatrná zmena v exekútore. V prípade, že používateľ chce taktiež monitorovať spustené úlohy v rámci daného dávkovacieho systému, je potrebné doplniť aj sprostredkovateľa Zmq2Ws o ďalší záznam, pričom v prípade, že daný dávkovací systém nemá priame poskytovanie informácií cez knižnicu ZeroMQ, je potrebné dotvoriť program, ktorý tieto dáta získa a poskytne ich skrz danú knižnicu.

V skratke je možné povedať, že tento projekt (všetky jeho časti) má rad výhod, ktoré je možné zhrnúť v tomto zozname:

- podpora Kubernetes
- jednoduchá nasaditeľnosť a správa
- jednoduché pridávanie nových častí

[4] https://edf-hpc.github.io/slurm-web/

- práca s klastrom pomocou VR alebo webového prehliadača

4 Záver

V rámci tejto práce bolo cieľom pripraviť spôsob, ktorým by mohli rôzne prostredia (webová aplikácia, prostredie virtuálnej reality) komunikovať a manipulovať s klastrami a dávkovacími systémami.

Celkovo bola táto práca rozdelená do niekoľkých častí, pričom prvou bola analytická časť, v rámci ktorej boli opísané základné pojmy v oblasti fyziky vysokých energií, urýchľovačov, infraštruktúry GRID, dávkovacích systémov, GRID midlvérov a tiež opisu virtuálnej reality. Okrem toho boli v tejto kapitole ukázané súčasné prístupy k riadeniu získavaniu dát, pričom táto časť je základom pre definovanie úlohy tejto práce. V analytickej časti bolo taktiež uvedené modelovanie hrozieb, ktoré bolo vytvorené pomocou nástroja na modelovanie.

Na začiatku syntetickej časti bol opísaný princíp preposielania údajov, pričom pre lepšie porozumenie tu bol zobrazený aj sekvenčný diagram, ktorý tento proces zobrazuje. Ďalšou časťou tejto kapitoly bolo vypísanie prípadov použitia, ktoré môžu nastať pri práci s týmto projektom. Následne bolo potrebné nakonfigurovať a nasadiť klaster na architektúru Kubernetes. Potom začala implementácia samotnej knižnice react-ndmspc, v ktorej boli tiež spomenuté a vysvetlené technológie, s ktorými bola táto knižnica vytvorená. V tomto prípade je to knižnica React, protokol WebSocket a tiež dizajnový rámec PatternFly. Táto časť tiež obsahuje samotnú implementáciu komponentov knižnice, nasadenie knižnice do webovej aplikácie, do nástroja Cockpit a tiež nasadenie na Kubernetes. Nasledujúcim krokom bola tvorba aplikačného programového rozhrania pre spúšťanie úloh, pričom prvým krokom bol výber technológie pre vytvorenie rozhrania. Rozhodovanie prebiehalo medzi rámcami FastAPI, Flask a Express, pričom bolo vykonané aj záťažové testovanie týchto rámcov. Po výbere bola začatá implementácia a následne bolo opísané nasadenie do klastra Kubernetes. Posledným krokom tejto časti bolo vytvorenie sprostredkovateľa pre dávkovací systém SLURM, pričom tento bol napísaný v jazyku Python a jeho úlohou bolo prijímať údaje z dávkovacieho systému (napr. SLURM) cez ZeroMQ, pretransformovať ich do potrebnej podoby a poslať ich pomocou protokolu WebSocket do webovej aplikácie,

resp. prostredia virtuálnej reality.

Poslednou časťou práce je vyhodnotenie, v rámci ktorej je overené, či požiadavky, ktoré boli požadované, sú splnené, a tiež tu boli spomenuté aj možné budúce požiadavky na projekt.

Literatúra

1. PERKINS, Donald H; PERKINS, Donald H. *Introduction to high energy physics*. CAMBRIDGE university press, 2000.
2. *About | CERN*. 2022-01-27. Dostupné tiež z: `https://home.cern/about`.
3. *JINR | Joint Institute for Nuclear Research*. 2022-01-28. Dostupné tiež z: `http://www.jinr.ru/about-en/`.
4. FOSTER, Ian; ZHAO, Yong; RAICU, Ioan; LU, Shiyong. Cloud computing and grid computing 360-degree compared. In: *2008 grid computing environments workshop*. 2008, s. 1–10.
5. PETERS, AJ; SINDRILARU, EA; ADDE, G. EOS as the present and future solution for data storage at CERN. In: *Journal of Physics: Conference Series*. 2015, zv. 664, s. 042042. Č. 4.
6. APOLLINARI, Giorgio; BRÜNING, O; NAKAMOTO, Tatsushi; ROSSI, Lucio. High luminosity large hadron collider HL-LHC. *arXiv preprint arXiv:1705.08830*. 2017.
7. EVANS, Lyndon. The large hadron collider. *New Journal of Physics*. 2007, roč. 9, č. 9, s. 335.
8. KHODZHIBAGIYAN, HG; AGAPOV, NN; AKISHIN, PG; BLINOV, NA; BORISOV, VV; BYCHKOV, AV; GALIMOV, AR; DONYAGIN, AM; KARPINSKIY, VN; KOROLEV, VS et al. Superconducting magnets for the NICA accelerator collider complex. *IEEE transactions on applied superconductivity*. 2013, roč. 24, č. 3, s. 1–4.
9. KEKELIDZE, VD. NICA project at JINR: status and prospects. *Journal of Instrumentation*. 2017, roč. 12, č. 06, s. C06012.
10. *Multi Purpose Detector*. 2022-01-27. Dostupné tiež z: `http://mpd.jinr.ru`.

11. KVAPIL, Jakub; BHASIN, Anju; BOMBARA, Marek; EVANS, David; JUSKO, Anton; KLUGE, Alexander; KRIVDA, Marian; KRALIK, Ivan; LIETAVA, Roman; NAYAK, Sanket Kumar; RAGONI, Simone; BAILLIE, Orlando Villalobos. *ALICE Central Trigger System for LHC Run 3*. 2021. Dostupné z arXiv: 2106.08353 [physics.ins-det].

12. BIRD, Ian. Computing for the large hadron collider. *Annual Review of Nuclear and Particle Science*. 2011, roč. 61, s. 99–118.

13. FAJARDO, EM; DOST, JM; HOLZMAN, B; TANNENBAUM, T; LETTS, J; TIRADANI, A; BOCKELMAN, B; FREY, J; MASON, D. How much higher can HTCondor fly? In: *Journal of Physics: Conference Series*. 2015, zv. 664, s. 062014. Č. 6.

14. JASON BOOTH. *Field Notes From the Frontlines of Support*. 2016-09-21. Dostupné tiež z: https://slurm.schedmd.com/SLUG21/Field_Notes_5.pdf.

15. M. JETTE AND M. GRONDONA. *Slurm: Simple Linux Utility for Resource Management*. 2003-06-23. Dostupné tiež z: https://slurm.schedmd.com/slurm_design.pdf.

16. M. JETTE. *Resource Management using SLURM*. 2006-05-01. Dostupné tiež z: https://slurm.schedmd.com/pdfs/lci.7.tutorial.pdf.

17. ADAPTIVE COMPUTING ENTERPRISES, Inc. Torque Resource Manager Administrator Guide. 2021. Dostupné tiež z: https://support.adaptivecomputing.com/wp-content/uploads/2021/02/torqueAdminGuide-6.1.3.pdf.

18. TSAREGORODTSEV, A; BARGIOTTI, M; BROOK, N; RAMO, A C; CASTELLANI, G; CHARPENTIER, P; CIOFFI, C; CLOSIER, J; DIAZ, R G; KUZNETSOV, G; LI, Y Y; NANDAKUMAR, R; PATERSON, S; SANTINELLI, R; SMITH, A C; MIGUELEZ, M S; JIMENEZ, S G. DIRAC: a community grid solution. 2008, roč. 119, č. 6, s. 062048. Dostupné z DOI: 10.1088/1742-6596/119/6/062048.

19. TSAREGORODTSEV, A; GARONNE, V; CLOSIER, J; FRANK, M; GASPAR, C; HERWIJNEN, E van; LOVERRE, F; PONCE, S; DIAZ, R Graciani; GALLI, D et al. DIRAC–distributed infrastructure with remote agent control. In: *Proc. of CHEP2003*. 2003.

20. STAGNI, F; TSAREGORODTSEV, A; ARRABITO, L; SAILER, A; HARA, T; AND, X Zhang. DIRAC in Large Particle Physics Experiments. 2017, roč. 898, s. 092020. Dostupné z DOI: 10.1088/1742-6596/898/9/092020.

21. CASAJUS, Adrian; GRACIANI, Ricardo; PATERSON, Stuart; TSAREGORODTSEV, Andrei; LHCB DIRAC TEAM, the. DIRAC pilot framework and the DIRAC Workload Management System. 2010, roč. 219, č. 6, s. 062049. Dostupné z DOI: 10.1088/1742-6596/219/6/062049.

22. BAGNASCO, Stefano; BETEV, L; BUNCIC, P; CARMINATI, F; CIRSTOIU, C; GRIGORAS, C; HAYRAPETYAN, A; HARUTYUNYAN, A; PETERS, AJ; SAIZ, Pablo. AliEn: ALICE environment on the GRID. In: *Journal of Physics: Conference Series*. 2008, zv. 119, s. 062012. Č. 6.

23. SHORT, H; MANZI, A; DE NOTARIS, V; KEEBLE, O; KIRYANOV, A; MIKKONEN, H; TEDESCO, P; WARTEL, R. x509-free access to WLCG resources. In: *Journal of Physics: Conference Series*. 2017, zv. 898, s. 102001. Č. 10.

24. HOLZ, Ralph; BRAUN, Lothar; KAMMENHUBER, Nils; CARLE, Georg. The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements. In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. 2011, s. 427–444.

25. WEISE, Joel. Public key infrastructure overview. *Sun BluePrints OnLine, August*. 2001, s. 1–27.

26. GREENGARD, Samuel. *Virtual reality*. Mit Press, 2019.

27. *53 VIRTUAL REALITY TECHNOLOGIES IN ARCHITECTURE AND ENGINEERING*. 2022-03-25. Dostupné tiež z: https://www.viatechnik.com/resources/50-virtual-reality-technologies-in-architecture-and-engineering/.

28. MATIS, Dominik. Webové rozhranie pre systém SALSA. 2020.

29. AGGARWAL, Sanchit. Modern web-development using reactjs. *International Journal of Recent Research Aspects*. 2018, roč. 5, č. 1, s. 133–137.

30. VIPUL, AM; SONPATKI, Prathamesh. *ReactJS by Example-Building Modern Web Applications with React*. Packt Publishing Ltd, 2016.

31. FETTE, Ian; MELNIKOV, Alexey. *The websocket protocol*. RFC 6455, December, 2011.

32. *PatternFly*. 2021-10-29. Dostupné tiež z: https://www.patternfly.org/v4/.

33. *About PatternFly*. 2022-01-27. Dostupné tiež z: https://www.patternfly.org/v4/get-started/about.

34. *Cockpit*. 2021-10-29. Dostupné tiež z: https://cockpit-project.org/.

35. MARTIN PITT. *Starter Kit - the turn-key template for your own pages*. 2018-03-09. Dostupné tiež z: `https://cockpit-project.org/blog/cockpit-starter-kit.html`.

36. ZAMOT, Michael. *An introduction to Cockpit, a browser-based administration tool for Linux*. Red Hat, 2020. Dostupné tiež z: `https://www.redhat.com/sysadmin/intro-cockpit`.

Zoznam skratiek

AI Artificial Intelligence.

API Application Programming Interface.

AR Augmented Reality.

CA Certification Authority.

CD Continuous Delivery.

CI Continuous Integration.

HEP High Energy Physics.

HPC High Performance Computing.

HTTP HyperText Transfer Protocol.

HTTPS HyperText Transfer Protocol Secure.

JSON JavaScript Object Notation.

JWT JSON Web Token.

LDAP Lightweight Directory Access Protocol.

NPM Node Package Manager.

PKI Public Key Infrastructure.

SDK Software Development Kit.

SSO Single Sign On.

TLS Transport Layer Security.

TUI Terminal User Interface.

UI User Interface.

UML Unified Modeling Language.

URL Uniform Resource Locator.

VR Virtual Reality.

Zoznam príloh

Príloha A CD médium – záverečná práca v elektronickej podobe,

Príloha B Článok,

Príloha C User guide (web verzia na CD),

Príloha D Admin guide (web verzia na CD),

Príloha E Devel guide (web verzia na CD)