VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE SPOUŠTĚCÍCH KONFIGURACÍ K-WAVE ÚLOH

OPTIMIZATION OF RUN CONFIGURATIONS OF K-WAVE JOBS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ SASÁK**
AUTHOR

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

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Sasák Tomáš**
Program: Informační technologie
Název: **Optimalizace spouštěcích konfigurací k-Wave úloh**
Optimization of Run Configurations of k-Wave Jobs
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s dostupnými plánovači úloh na současných superpočítačích. Zaměřte se především na systémy PBS a Slurm.
2. Seznamte se detailně s fungováním plánovače úloh ve středisku IT4Innovatioins.
3. Analyzujte chování různých implementací k-Wave v závislosti na charakteru vstupních dat a spouštěcí konfiguraci.
4. Navrhněte heuristiky, které na základě výkonnostních metrik získaných z předchozích běhů, naleznou vhodné spouštěcí konfigurace pro danou úlohu a aktuální vytížení superpočítače.
5. Navržené řešení implementujte v jazyce Python.
6. Na vhodné sadě úloh otestujte kvalitu navržených heuristik.
7. Zhodnoťte dosažené výsledky a diskutujte jejich přínos pro praxi.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 4 zadání.
- Vypracování textové části semestrální práce.

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

Vedoucí práce: **Jaroš Jiří, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Táto práca sa zaoberá plánovaním, resp. správnym odhadom spúšťacích konfigurácií úloh k-Wave na superpočítačoch infraštruktúry IT4Innovations. Presnejšie pre klastre Salomon a Anselm. Úloha predstavuje množinu simulácií, kde každá simulácia je spúšťaná pod toolboxom k-Wave. Pre spustenie jednotlivých simulácií je nutné správne vytvoriť konfiguráciu, ktorá sa skladá z množstva zdrojov (počet výpočtových uzlov, resp. jadier) a času rezervácie superpočítača, čo je pre neskúseného zložité odhadnúť. Zvolený problém odhadu je riešený na základe empirických dát, ktoré boli získané viacnásobným spúšťaním rôznych množín simulácií na klastroch. Tieto dáta sú uložené a spracované aproximátormi, ktoré konkrétne vykonávajú odhad týchto parametrov na základe metód interpolácie a regresie. V práci je popísaný a bol implementovaný systém predstavujúci plánovač, ktorý predstavuje rozhranie pre odhad. Experimentovaním bolo zistené že pre tento špecifický problém najpresnejšie odhady vykonáva trojica Akima spline, PCHIP interpolácia a kubický spline. Výsledky tejto práce umožňujú vykonávať istý odhad exekučného času a počtu vlákien pre ľubovolné simulácie automaticky a bez znalosti kódu k-Wave.

Abstract

This thesis focuses on scheduling, i.e. correct approximation of configurations used to run k-Wave simulations on supercomputers from the IT4Innovations infrastructure. Especially, for clusters Salomon and Anselm. A single work is composed of a set which contains many simulations. Every simulation is executed by some code from the k-Wave toolbox. To calculate the simulation, it is necesarry to select a suitable configuration, which means the amount of supercomputer resources (number of nodes, i.e. cores), and the duration of the rental. Creation of an ideal configuration is complicated and is even harder for an inexperienced user. The approximation is made based on the empiric data, obtained from multiple executions of different sets of simulations on given clusters. This data is stored and used by a set of approximators, which performs the actual approximation by methods of interpolation and regression. The text describes the implementation of the final scheduler. By experimenting, the most efficient methods for this problem has found out to be Akima spline, PCHIP interpolation and cubic spline. The main contribution of this work is creation of a tool which can find suitable configuration for k-Wave simulation without knowing the code or having lots of experience with its usage.

Kľúčové slová

superpočítač, plánovač, plánovanie, dáta, optimalizácia, k-wave, klaster, salomon, anselm, experiment, meranie, predikcia, it4i, simulácia, hpc, pbs, slurm, aproximácia, interpolácia, regresia, data-mining

Keywords

supercomputer, scheduler, scheduling, data, optimalisation, k-wave, cluster, salomon, anselm, experiment, measuring, prediction, it4i, simulation, hpc, pbs, slurm, approximation, interpolation, regression, data-mining

Citácia

SASÁK, Tomáš. *Optimalizace spouštěcích konfigurací k-Wave úloh.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Jiří Jaroš, Ph.D.

Optimalizace spouštěcích konfigurací k-Wave úloh

Prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Doc. Ing. Jiří Jaroš, Ph.D. A uviedol som všetky literárne zdroje, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Tomáš Sasák
25. mája 2020

Poďakovanie

Chcel by som venovať poďakovanie môjmu vedúcemu práce Doc. Ing. Jiří Jaroš, Ph.D, za veľkú aktivitu a rady pri vypracovávaní tejto práce. Taktiež, by som chcel poďakovať mojej rodine a kamarátom, ktorí ma behom štúdia podporovali. Táto práca bola podporená Ministerstvom školstva, mládeže a telovýchovy z Národného programu udržateľnosti II (NPU II) v rámci projektu "IT4Innovations excellence in science - LQ1602" a výsledky boli získané využitím výskumnej infraštruktúry podporenej z programu Veľkých infraštruktúr pre výskum, experimentálny vývoj a inovácia v rámci projektu "IT4Innovations národné superpočítačové centrum - LM2015070".

Obsah

Kapitola 1

Úvod

V dnešnej dobe, kedy počítače si už prešli dlhým vývojom je možné rôzne výpočty vyhodnotiť v okamihu niekoľko stotín sekundy. To ale neplatí pre veľmi náročné výpočty (napr. simulácie), kde je potrebná hrubá výpočtová sila superpočítačov, aby bol výpočet zdrojmi a časovo zvládnuteľný v prijateľnom rámci. Takisto je nutná automatizácia ovládania počítačov, pretože v istých situáciách nie je prípustné, aby používateľ neustále čakal na výsledok výpočtu a následne na to reagoval, a púšťal ďalšie výpočty.

Táto bakalárska práca sa zaoberá hľadaním vhodných konfigurácii pre spúšťanie simulácii na súprave nástrojov k-Wave na superpočítačoch od IT4Innovations. Cieľom je vytvoriť algoritmus, plánovač, ktorý tieto konfigurácie dokáže vhodne odhadnúť. Aktuálne neexistuje automatizované riešenie, ktoré by dokázalo odhadnúť efektívnu konfiguráciu na základe vstupov, dát a určitej heuristiky špecializovanej pre k-Wave. Konfigurácia predstavuje dvojicu hodnôt, kde prvá hodnota predstavuje množstvo výpočtových uzlov a exekučný čas, čiže doba alokácie týchto uzlov.

Ak používateľ potrebuje spustiť simuláciu, musí odhadnúť, aká konfigurácia spustenia by mohla byť uspokojujúca. Toto sa už odvíja od používateľa, na základe jeho skúseností s programom k-Wave a daným výpočetným strojom (v práci Salomon a Anselm). Vytvorenie odhadu môže byť veľmi zložité pre nových používateľov s nedostatočnými skúsenosťami. A zároveň časovo náročné aj pre skúsených používateľov.

Cieľom práce je zoznámiť sa so superpočítačmi od IT4Innovations, presnejšie s výpočetnými klastermi a plánovačom PBS a SLURM. Analyzovať chovanie rôznych implementácii k-Wave, pri rôznych variáciách vstupov a parametrov. Spracovať toto chovanie do logických celkov a rozumne pristupovať ku nazbieraným dátam. Zamyslieť sa a navrhnúť heuristiky, ktoré na základe týchto dát nájdu vhodné spúšťacie konfigurácie pre dané vstupy a parametre. Tieto heuristiky a celkové riešenie navrhnúť a implementovať v jazyku Python. Porovnať finálny výsledok s realitou a zamyslieť sa nad prínosom pre prax.

Ako úvod pre hlbšie pochopenie problematiky, práca popisuje superpočítače a ich architektúry. Vzhľadom na to, že na superpočítači pracuje viac používateľov naraz, je nutné aby exekúcia úloh bola nastavená v správnych mierach, preto práca sa ďalej práca zaoberá implementáciami plánovačov, ktoré sa starajú o správne riadenie a exekúciu úloh. Ako nasledujúce je nutné oboznámiť čitateľa s infraštruktúrou IT4Innovations na ktorej prebieha beh simulácie, pre ktorú prebieha daný odhad a plánovanie. Pretože odhad sa vykonáva špecializovane pre toolbox k-Wave, je nutné odôvodniť jeho podstatu a implementácie, na ktoré sa odhad bude zameriavať, toto je vysvetlené v časti k-Wave. Spôsob, akým je možné vytvárať odhad a aproximovať parametre je opísaný v kapitole pre Odhad na základe empirických dát. Týmito spôsobmi sa ďalej bude riadiť implementácia.

Na začiatku je nutné sa zamyslieť, čo už implementované existuje, čo treba implementovať a ako takýto plánovač by mohol pracovať, toto opisuje kapitola Cieľ, návrh a architektúra práce. Táto kapitola obsahuje popísaný cieľ práce a všeobecný algoritmus plánovača. Aby bolo možné realizovať vôbec odhad z empirických dát, je nutné dáta nazbierať, usporiadať do rozumného formátu a vytvoriť ku ním rozumné rozhranie pre prácu. Táto problematika a spôsob získavania dát špeciálne pre k-Wave je prebratý v ďalšej kapitole Implementácia plánovača.

Kapitola Implementácia plánovača sa zaoberá implementáciou výsledného plánovača, ktorý odhaduje parametre správneho spustenia úlohy na superpočítači na základe heuristík, ktoré boli predtým rozobraté.

Testovanie a porovnania rôznych heuristík a výsledky sú zhrnuté v kapitole Experimentálna časť. Táto časť obsahuje porovnanie aproximátorov a pohľad na odhad pre každú binárnu verziu k-Wave. Nakoniec každý pohľad obsahuje výsledky testovacej množiny špeciálne vytvorenej pre rôzne odhady.

Posledná kapitola s názvom Záver obsahuje zhrnutie dosiahnutých cieľov. Taktiež obsahuje informácie o osobnom prínose pre autora, návrhy nad možným vylepšením práce a výsledného produktu v budúcnosti.

Použité technológie

Práca využíva nasledovné technológie pre implementačnú časť, presnejšie vývoj a prácu. `Python3, Docker, Docker-compose, Kontajnerizácia, Peewee (ORM), SciPy, NumPy, MATLAB, Matplotlib, PostgreSQL, HDF, Jupyter Notebook, Bash, Linux.`

Kapitola 2

Superpočítače a ich architektúra

Táto kapitola obsahuje rozbor a detailný popis moderných superpočítačov. Zozačiatku popisuje podstatu superpočítačov a ich úlohu, ďalej rozoberá historické no aj aktuálne architektúry superpočítačov, ktoré môžme vidieť v superpočítačoch tejto doby. Na záver nasleduje porovnanie a pohľad na aktuálny stav superpočítačov. Táto časť obsahuje teóriu, ktorá tvorí základ tejto práce.

2.1 Superpočítač

Superpočítač (z definície) je počítač s veľmi vysokým výkonom v porovnaní s osobným počítačom. Superpočítače majú významnú rolu v informatike, a sú používané v širokej škále vysoko náročných výpočtov, ako napríklad kvantová mechanika, rôzne simulácie, predpoveď počasia a mnoho ďalších.

Merania výkonu superpočítača, sú dané v jednotkách FLOPS [8] (FLoating-point Operations Per Second), čo znamená počet operácii v pohyblivej desatinnej čiarke za sekundu. V dnešnej dobe (rok 2020) sa počty FLOPS pohybujú rádovo už v stovkách Peta. Zvyčajne, na superpočítači beží operačný systém Linux. Medzi krajiny ktoré vlastnia najviac superpočítačov patrí Spojené štáty americké, Čína, či Japonsko[1].

2.2 Terminológia

V nasledujúcich častiach práce, sa bude pracovať s určitými termínmi, ktorých význam je potrebné vysvetliť, aby nedošlo ku zavádzajúcim informáciám. Termíny sú nasledujúce.

Výpočetný uzol (Node)

Znamená jeden počítač, ktorý môže byť pripojený cez internú sieť a umožňuje komunikáciu s ostatnými uzlami. Výpočetný uzol je špecializovaný pre výpočet zadaných zložitých výpočtov (úloh). Jeden výpočetný uzol môže mať viacero procesorov, určitú veľkosť operačnej a vnútornej pamäte.

[1] Informácia prevzatá z Top500 (https://www.top500.org/lists/top500/).

Jadro (Core)

Znamená jedno jadro procesora. Je to jednotka procesora, ktoré vykonáva výpočet. Jadro sa môže skladať z viacero cache pamätí, FPU (floating-point unit, matematický koprocesor) a mnoho ďalších častí, pričom záleží na implementácii procesora.

Jadro-hodina (core-hours)

Jednotka udávajúca dĺžku použitia jedného jadra výpočetného uzlu. Udáva cenu výpočtu, kde poskytovateľ si účtuje čiastku za jednotku jadro-hodiny. Vzorec pre prepočet skutočného času na jadro-hodiny je nasledujúci.[2]

$$C_h = N_n \times N_{nc} \times T \tag{2.1}$$

Kde:
C_h - Počet jadro-hodín
N_n - Počet alokovaných uzlov
N_{nc} - Počet alokovaných jadier
T - Počet hodín alokácie

Vlákno

Sekvencia programového kódu, ktorý procesor vykonáva. Zvyčajne sa k jednému jadru viažu jedno a viac vlákien.

Úloha

Jednotka práce, skladá sa z krokov, ktoré môžu prebiehať sekvenčne alebo paralelne. Práca môže byť jeden rodičovský proces, ktorý neskôr vytvorí ďalšie potomkovské procesy. Úloha taktiež môže reprezentovať jeden celok kódu zadaný pre superpočítač.

Démon

Program, ktorý beží na pozadí a obvykle nepotrebuje priamu interakciu s používateľom. Vo väčšine prípadov ide o proces prebiehajúci v nekonečnom cykle, ktorý reaguje na určité volania alebo požiadavky.

2.3 Architektúra superpočítačov

V rámci práce je ďalej viacero pohľadov, architektúr ako je možné realizovať a implementovať superpočítač. Tieto architektúry sú uvedené chronologicky, od najstarších, jednoduchších architektúr, k aktuálnejším a pokročilejším architektúram. [5]

2.3.1 Superpočítač založený na vektorovom procesore

Jedna z najstarších architektúr superpočítačov. Superpočítač sa skladá z hlavného vektorového procesoru, ktorý je špecializovaný na prácu s vektormi a s ich operáciami. Vektor

[2] Výpočet prevzatý z dokumentácie od IT4Innovations. https://docs.it4i.cz/general/resources-allocation-policy/#normalized-core-hours-nch

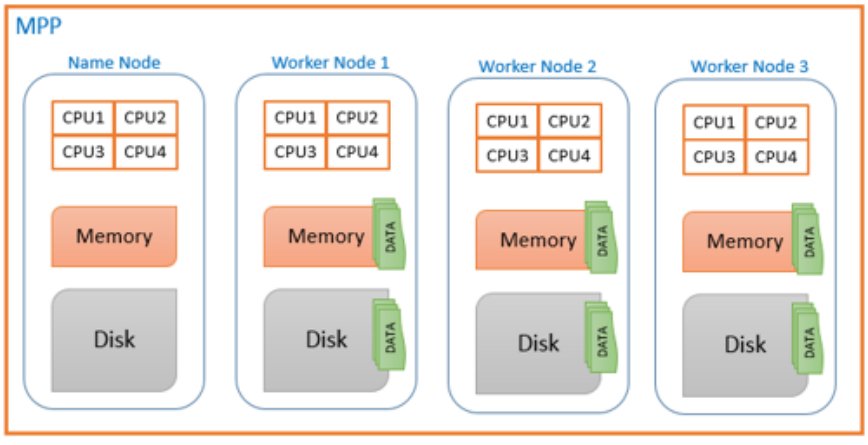
predstavuje veľký pamäťový blok do ktorého sa dokáže poskladať viacero hodnôt súčasne. Oproti skalárnym procesorom, ktoré spracovávajú jednu dátovú jednotku v čase, vektorový procesor pracuje nad celým vektorom súčasne. To znamená, že inštrukcie vektorového procesoru majú operandy celé vektory. Vďaka tejto vlastnosti, je možné vykonať viacero operácii súčasne v rámci jednej operácie. Taktiež, sa znižuje množstvo načítavania a prekladania názvu operácii.

Ako všetky architektúry, aj táto má určité nevýhody. Vektorové procesory sú príliš drahé oproti skalárnym procesorom, ktoré je možno využiť napríklad v klaster architektúre. Príčinou je, že vektorový procesor používa vysokorýchlostné pamäťové čipy. Najväčší problém ale vzniká z fyzického pohľadu, kde už nie je možné zvýšiť takt, ktorý procesory potrebujú. Preto sa na vektorové procesy zanevrelo a v dnešnej dobe už nenájdeme superpočítače, ktoré obsahujú takýto procesor, pričom sa môžu rovnať ostatným main-streamovým architektúram.

Vektorové operácie ale úplne nezmizli z procesorov terajšej doby. Podobné sady inštrukcii je možné nájsť v procesoroch od výrobcu Intel nesúce názov MMX, SSE a AVX. Pre tieto inštrukcie existujú špeciálne vektorové registre s veľkým bitovým obsahom, kde sa na základe inštrukcie z danej sady vykoná vektorová operácia. Treba ale pripomenúť, že procesor je stále skalárny ale obsahuje zopár špeciálnych vektorových operácii a registrov. Tento spôsob efektívneho výpočtu je taktiež možné nájsť v grafických kartách.

2.3.2 Masívne paralelný procesor

Po úpadku superpočítačov založených na vektorovom procesore sa objavila nová architektúra, ktorá nahradila vektorovú architektúru. Architektúra masívne paralelných procesorov (ďalej ako MPP) je založená na použití dvoch a viacerých procesorov, kde každý procesor má svoju vlastnú operačnú pamäť a vlastný operačný systém. Procesory bežia, a vykonávajú svoju samostatnú činnosť paralelne, pričom sú medzi sebou prepojené vysokorýchlostnou zbernicou na základovej doske. Medzi tieto procesory môže byť rozdelený náročný výpočet, v ktorom každý procesor vykonáva určitú časť a medzivýsledky si procesory predávajú medzi sebou pomocou spomínanej vysokorýchlostnej zbernice. Táto architektúra môže čitateľovi pripadať zhodná s klaster architektúrou, ale nie je rovnaká.



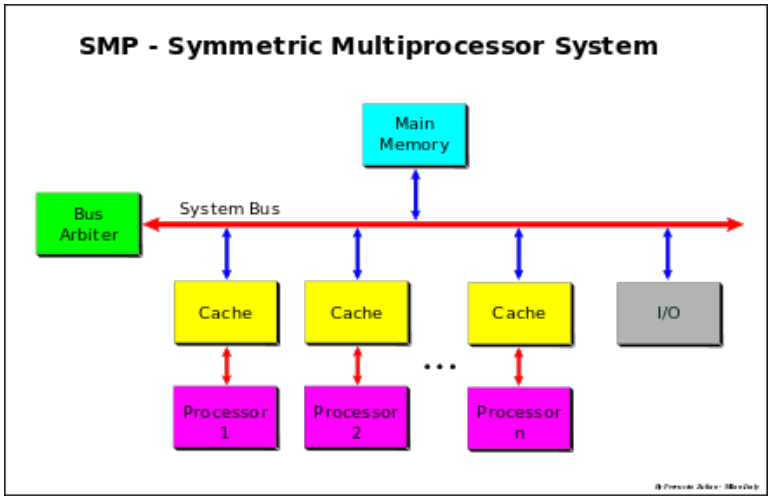
Obr. 2.1: Architektúra MPP [7]

MPP je do dnešnej doby používaná, a aj v aktuálnom zozname najlepších 500 superpočítačov dokážeme nájsť superpočítače využívajúce túto architektúru. MPP má taktiež značné využitie pri databázových systémoch, kde sa databáza rozdeľuje do viacero samostatných blokov (sharding), a následne každý procesor a operačný systém odpovedá pre danú časť databázy.

2.3.3 Symetrické multi-procesovanie

MPP architektúra pokračovala svojim úspechom a stále sa zlepšovala, medzitým však vznikla nová architektúra menom symetrické multi-procesovanie (ďalej ako SMP).

SMP sa skladá z dvoch a viacerých procesorov zapojených symetricky. Tieto procesory sú úzko spojené, čo znamená že majú prístup k zdieľanej operačnej pamäti pomocou vysokorýchlostnej zbernice a pracujú v rámci jedného operačného systému, ktorý väčšinou zaobchádza rovnako so všetkými procesormi. Vzhľadom na to, že procesory sú úzko spojené nie je nutné vzájomné vnútorné pripojenie cez sieť, takže oneskorenie medzi procesormi je veľmi nízke. Všetky procesory majú rovnaký prístup ku zdieľanej pamäti, Input-Output zariadeniam a službám operačného systému.



Obr. 2.2: Architektúra SMP[3]

Vzhľadom na to, že SMP používa spoločný operačný systém a operačnú pamäť vzniká limitácia počtu symetrických procesorov, ktorých maximálny počet sa pohybuje okolo 64.

2.3.4 Klaster (Cluster)

Klaster je architektúra superpočítača, ktorá funguje ako kolekcia viacero separátnych výpočtových uzlov, ktoré sú prepojené medzi sebou vysoko rýchlostnýmpripojením. Môže existovať väčší počet rôznych výpočetných uzlov, ktoré sa špecializujú na rôzne úlohy. Uzly môžu byť následovné.

Prihlasovací uzol (login Node)

Ovláda prístup na superpočítač, resp. klaster, a spracováva požiadavky na celý klaster.

[3]Obrázok prevzatý z https://commons.wikimedia.org/wiki/File:SMP_-_Symmetric_Multiprocessor_System.svg

Výpočtový uzol (Node)

Uzol, kde prebieha časť náročného výpočtu.

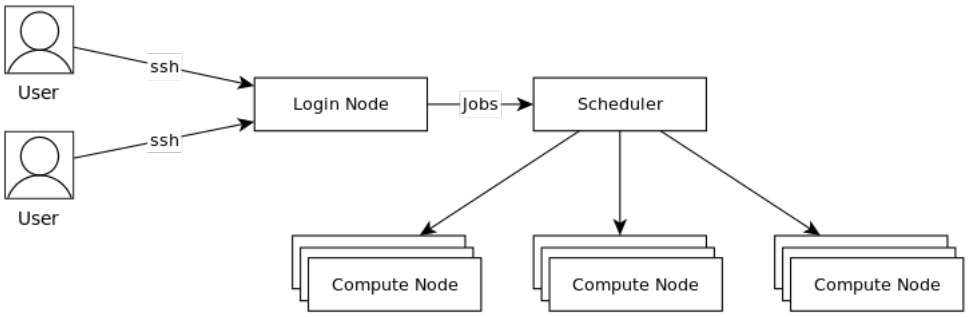
Grafický uzol (GPU Node)

Uzol obsahuje výkonné grafické karty pre podporu výpočtov ktoré potrebujú grafickú kartu.

Tlstý uzol (Fat Node)

Uzol obsahuje veľké množstvo pevnej pamäti.

Tieto uzly, tvoriace klaster, potrebujú byť spravované určitým programom, ktorý nesie meno plánovač. Klaster architektúra nemusí byť nutne realizovateľná iba pri superpočítačoch. Vzhľadom na to, že viacero výpočtových uzlov pracuje spoločne, je možné vytvoriť klaster aj z lacnejších výpočtových uzlov, čo môže byť vhodné pre firmy, univerzity, hosťovanie služieb atď.



Obr. 2.3: Architektúra klaster [6]

V neposlednom rade táto architektúra umožňuje stavbu superpočítačov za lepšiu cenu, pretože nie je nutné navrhnúť a implementovať špecializovaný procesor pre superpočítač, ale je možné nakúpiť viac lacnejších univerzálnych procesorov a vytvoriť z nich klaster.

2.4 Aktuálny stav

Aktuálne (rok 2019) je v zozname Top 500 superpočítačov dominantná majorita superpočítačov obsahujúcich architektúru Klaster, no taktiež je v ňom možné nájsť aj architektúru MPP.

Kapitola 3

Plánovače superpočítačov

Jedna z najzákladnejších a najpotrebnejších vecí pre superpočítač je plánovač. Plánovač je software, ktorý slúži pre kontrolu a exekúciu úloh na určitom zariadení. Plánovanie spúšťania jednotlivých úloh nie je možné vykonávať ručne, je nutné toto automatizovať. Pretože superpočítač v majorite prípadov využíva skupina ľudí, a nie len jeden užívateľ, je nutné zdroje superpočítača adekvátne rozdeliť.

Existuje mnoho rôznych parametrov, podľa ktorých plánovač spúšťa a uprednostňuje určité úlohy pred inými. Parametre teda môžu byť.

- Priorita úlohy
- Množstvo aktuálnych voľných zdrojov zariadenia
- Počet nevyužitých hodín alokácie používateľa
- Veľkosť požadovaného alokačného času
- Maximum a aktuálny počet úloh v rade

Používateľ zvyčajne pracuje s rozhraním plánovača, ktorému zadá žiadosť o vykonanie úlohy vo forme parametrov potrebných pre vykonanie úlohy a samotnú úlohu, ktorá sa má vykonať. Táto požiadavka je zaregistrovaná a uvedená do fronty čakajúcich požiadaviek, nad ktorými pracuje plánovač. V tejto práci budú podrobnejšie rozobrané dva plánovače úloh pre superpočítač.

- Portable Batch System (ďalej ako PBS) [9]
- Simple Linux Utility for Resource Management (ďalej ako SLURM) [10]

Je nutné podotknúť, že tieto plánovače sú primárne použité pre klaster architektúru superpočítačov.

3.1 Portable Batch System

Je jedna z implementácii plánovača úloh pre superpočítač. Existuje vo viacero implementáciách.

- PBS Job Professional
- OpenPBS

- TORQUE

PBS plánovač je majoritne spájaný so superpočítačmi, ktoré používajú operačný systém založený na UNIX, ale je možné ho použiť aj na operačnom systéme Windows.

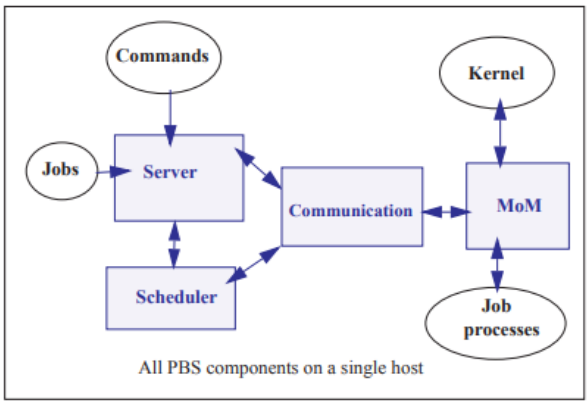
Skladá sa z dvoch hlavný komponentov, systémové procesy a príkazy na užívateľskej úrovni. Spočiatku bol PBS vyrobený v roku 1991 špeciálne pre jednotku NASA spoločnosťou MRJ Technology Solutions. Neskôr ale PBS prebrala spoločnosť Altair Engineering, ktorá nadobudla aj jeho výlučné vlastníctvo. Altair Engineering spravuje implementáciu PBS Job Professional, ktorú táto práca podrobnejšie rozoberá ďalej.

3.1.1 Architektúra plánovača PBS

PBS sa skladá z viacerých architektúr, ale v praxi sa na základe charakteristiky zariadenia, na ktorom PBS pracuje, používa iba jedna architektúra. [9]

Architektúra pracujúca na jednom zariadení

Ak PBS pracuje na zariadení, ktorého komponenty sú nainštalované na tom istom zariadení, architektúra vyzerá nasledovne.

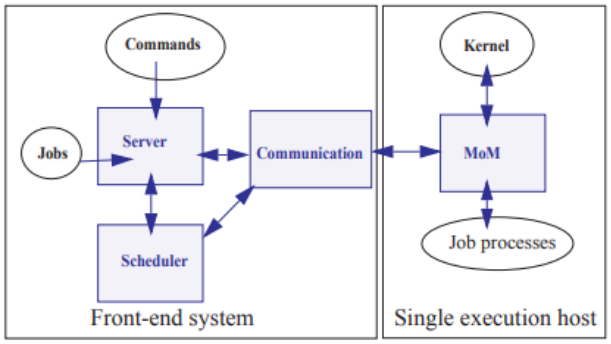


Obr. 3.1: Architektúra PBS na jednom zariadení [9]

Každá časť PBS pracuje na rovnakom zariadení a neexistuje žiadne oddelenie častí.

Architektúra pracujúca na jednom zariadení obsahujúce čelnú časť

Celkové zariadenie môže byť rozdelené na dva uzly. Uzol na ktorom prebieha výpočet a uzol, ktorý slúži ako rozhranie pre užívateľa na prácu s PBS. Potom architektúra vyzerá nasledovne.

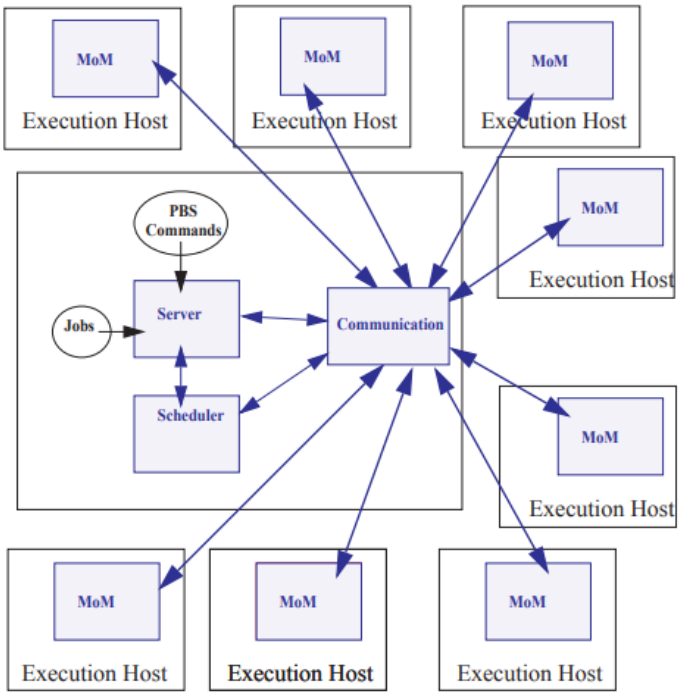


Obr. 3.2: Architektúra PBS na jednom zariadení s čelnou časťou [9]

Kde uzol, čelná časť naľavo, obsahuje spomínané rozhranie skladajúce sa z Server, Scheduler a Communication časti, a pravá strana obsahuje výpočetný uzol, na ktorom sa výpočet vykonáva a obsahuje časť MoM (exekútor úloh). Uzol na ktorom pracuje čelná časť môže byť výkonnostne omnoho slabší ako ten, na ktorom prebieha výpočet. Tento uzol by ale mal byť schopný obsluhovať viac používateľov naraz.

Architektúra pracujúca s viac výpočtovými uzlami a čelnou časťou

PBS môže pracovať aj na zariadení, ktoré obsahuje uzol s čelnou časťou a viac ako jeden výpočetný uzol. Podstata sa zachováva z predchádzajúcej architektúry, jediný rozdiel je v tom, že časť Scheduler si bude musieť pamätať, že riadi viac výpočtových uzlov naraz. To platí aj pre časť Communication, ktorá komunikuje naraz s viacerými výpočtovými uzlami.



Obr. 3.3: Architektúra PBS pre viac výpočtových uzlov a čelnú časť [9]

3.1.2 Časti architektúry

Pre úplné pochopenie architektúry plánovača, je nutné podrobnejšie vysvetliť bloky ktoré ju tvoria.

Server

Jedna z najhlavnejších častí PBS, zasielajú sa na neho užívateľské príkazy pre manipuláciu s úlohami. Všetka komunikácia so serverom je realizovaná pomocou internetovej siete, presnejšie pomocou sieťového protokolu Internet Protocol (IP).

Jeho hlavnou funkciou je poskytovať služby pre manažovanie úloh, ako napríklad prijatie požiadavky o exekúcii úlohy, rôzne modifikácie aktuálnej úlohy a následné spúšťanie úlohy. Jedna dôležitá funkcia, ktorú je nutné spomenúť, je ochrana úlohy proti systémovým pádom a útokom. Server si musí jasne pamätať, aké úlohy čakajú v rade rovnako, ako ich metadáta aj pri páde systému, či rôznych iných problémoch.

Scheduler (plánovač)

Rozum PBS, ovláda logiku plánovania úloh a vyberá, ktorá úloha má byť kedy spustená, na akej skupine výpočtových uzlov a na akom množstve uzlov zo skupiny. Toto riadenie plánovačom môže byť rôzne, záleží na tom aká politika plánovania je vybraná. Politika plánovania je zoznam pravidiel podľa ktorých sa snažíme dosiahnuť korektné chovanie plánovača a spôsobu plánovania úloh. O tejto politike rozhodujú administrátori zariadenia, na ktorom plánovač pracuje.

PBS obsahuje základnú preddefinovanú politiku, ale pokročilý administrátor môže implementovať svoju politiku, pričom v majorite prípadov je politika implementovaná podľa požiadaviek na zariadenie. Implementácia politiky je možná pomocou PBS nástrojov, ktoré ovládajú rôzne aspekty plánovania a tieto aspekty môže administrátor meniť tak, aby dosiahol požadovanú politiku. Tieto PBS nástroje, sa nemusia nevyhnutne vyskytovať na jednom bloku architektúry.

Communication Daemon (komunikačný démon)

Slúži pre komunikáciu s ostatnými démonmi. Nachádza sa na uzle s čelnou časťou, kde komunikuje s ostatnými démonmi, prevažne na výpočtových uzloch.

MoM (exekútor úloh)

MoM slúži ako exekútor úloh. Každý výpočetný uzol obsahuje MoM, vďaka ktorému sa úloha môže spustiť. Názov MoM, z anglického slova “mom” označujúci matku naznačuje, že každý proces úlohy je spustený ako potomok rodiča, čiže tohto rodičovského procesu.

MoM spúšťa proces úlohy potom, ako dostane správu od serveru s danou kópiou úlohy, ktorej výpočet sa má spustiť. Pri spúšťaní úlohy sa MoM snaží čo najviac napodobniť užívateľské prostredie, to znamená že, ak napríklad používateľ pri zadávaní úlohy použil typ zsh shell, MoM sa bude snažiť spustiť exekúciu danej úlohy pomocou zsh shell. Ďalšou hlavnou úlohou tejto časti, je vrátenie obsahu štandardných streamov operačného systému (napr. STDIN, STDOUT, STDERR) serveru, aby bolo možné pre používateľa skontrolovať výstup úlohy, prípadne chyby.

3.1.3 Fronta uzlov (Queue)

Pretože rôzne výpočetné uzly v architektúre klaster jedného superpočítača môžu mať rôzne komponenty a periférne zariadenia, existuje v plánovači PBS systém front. Superpočítač, môže byť rozdelený na výpočetné uzly a každý uzol môže zapadať do inej množiny fronty uzlov plánovača. Fronta je aplikačná, nie je implementovaná pomocou hardware. Každá fronta, má svoje meno a svoju politiku, podľa ktorej plánovač rozhoduje ako s frontou nakladať. Politika môže obsahovať: maximálnu doba alokácie uzlov, maximálny počet alokovaných uzlov, skupinu používateľov, ktorá môže môže uverejňovať prácu do fronty, atď.

Tento pojem môže byť trochu mätúci. Fronta v rámci PBS, označuje množinu uzlov pod daným menom a danou politikou, ale k tomu obsahuje aj frontu úloh, kde tieto úlohy žiadajú o zdroje (výpočetné uzly) z danej fronty. Fronta úloh v rámci fronty uzlov, nemusí byť nútene FIFO, LIFO, ale plánovač nad ňou pracuje pomocou spomínaných politík.

3.1.4 Žiadanie o zdroje a odosielanie úloh

Pri zverejňovaní výpočtovej úlohy na superpočítač, je nutné udať niektoré povinné parametre. Tieto parametre môžu byť následovné.

- Názov fronty uzlov
- Dĺžka alokácie uzlov
- Množstvo alokovaných uzlov
- Skript, ktorý sa spustí

Ďalej je nutné odôvodniť čo znamená tento skript.

Pracovný skript (Job script)

Pracovný skript, je typický skript daného skriptovacieho jazyka. Väčšinou implementovaný pre unixový shell menom Bash, ktorý interpretuje tento príkazový jazyk. Skript obsahuje postupnosť krokov, ktoré shell interpretuje, v momente keď je alokované miesto pre používateľa (došla na neho rada) a je mu pridelená vybraná množina uzlov z fronty uzlov, presnejšie došlo na jeho úlohu, zaradenú v fronte úloh.

Plánovač PBS umožňuje v komentároch pracovného skriptu (Bash skriptu) vypísať parametre pre požiadanú úlohu, aby používateľ nemusel pri každom uverejňovaní úlohy opakovane zapisovať požiadavky, a taktiež aby pracovné skripty mohli byť automaticky generované inými užívateľskými skriptami. Pretože pracovný skript je zátaž na plánovač a väčšina politík obmedzuje počet naraz uvedených pracovných skriptov, existuje ešte jeden spôsob uverejňovania veľkého množstva úloh.

Pole pracovných skriptov (Job array)

Pole slúži pre zverejnenie viacerých výpočtových úloh v rámci jednej úlohy, čo môže pomôcť predísť obmedzeniu počtu požiadaviek naraz. Jeden člen pracovného poľa sa nazýva podúloha (subjob). Jedna úloha sa teda skladá z viacerých podúloh, ktoré sú uložené v poli.

Každá pod-úloha je spustená za rovnakých podmienok, ako je zadané v požiadavke na celú úlohu. To znamená, že ak používateľ spustí jednu úlohu (pole), ktorá obsahuje sto podúloh a úloha všeobecne požaduje jeden výpočetný uzol, každá podúloha bude pustená

na jednom výpočtovom uzle. To, či už úlohy budú spustené naraz na rôznych uzloch alebo postupne na jednom uzle, záleží na nastaveniach plánovača, obsadenosti uzlov a na ďalších iných podmienkach.

3.1.5 Praktický príklad použitia PBS

Existuje superpočítač menom Lean, ktorý pre plánovanie a exekúciu úloh používa plánovač úloh PBS. Tento superpočítač používajú dvaja používatelia menom Abel a Johnatan. V rámci plánovača úloh, existujú dve fronty uzlov menom A a B. Superpočítač sa skladá zo štyroch uzlov, z čoho sú tri výpočetné a uzol s čelnou časťou. Prvé dva výpočetné uzly, patria pod frontu uzlov A a posledný výpočetný uzol pod frontu uzlov B. Obaja používatelia sa prihlásia na svoje účty na uzle s čelnou časťou. Johnatan zašle požiadavky o vykonaní troch úloh, kde prvé dve úlohy potrebujú jeden výpočetný uzol z fronty uzlov A a tretia úloha, ktorá žiada jeden výpočetný uzol z fronty uzlov B. V tom istom čase, Abel zašle požiadavku na vykonanie úlohy, ktorá potrebuje jeden výpočetný uzol z fronty uzlov B. Všetky tieto požiadavky sú odosielané z terminálu na server PBS, ktorý ich spracuje, uloží do fronty úloh pre danú frontu uzlov a začne nad nimi pracovať plánovač.

Plánovač na základe nakonfigurovanej politiky vyberie, kedy sa má ktorá úloha spustiť. Pre príklad, plánovač používa obyčajnú politiku, plánovač sa pozrie do fronty úloh pre frontu uzlov A, aké úlohy majú byť vykonané a pozrie sa na aktuálny stav uzlov, plánovač vidí, že na uzloch fronty A nepracuje nič, a umiestni tam prvú úlohu od Johnatana, ktorá potrebuje iba jeden výpočetný uzol. Zostal jeden volný výpočetný uzol pre frontu uzlov, a tak isto v fronte čaká aj druhá úloha od Johnatana pre tú istú frontu, zostal jeden výpočetný uzol voľný, na ktorom je možné spustiť exekúciu druhej úlohy, a to plánovač vykoná. Dve úlohy Johnatana na uzloch fronte A prebiehajú naraz, na dvoch oddelených uzloch, uzly sú teraz obsadené a každá ďalšia úloha bude musieť počkať do doby, až sa úlohy dokončia a uvoľní sa potrebný počet uzlov. Premiestnenie úlohy z fronty na výpočetný uzol, oznámi plánovač serveru, ktorý vytvorí kópiu úlohy, pridá metadáta a tento celok predá komunikačnej časti. Komunikačná časť spomínaný celok odošle časti MoM, ktorá celok prijíma, vykoná určitú predprípravu pred exekúciou úlohy, a následne spustí exekúciu úlohy v potomkovskom procese. MoM v priebehu vykonávania práce spätne odosiela stav štandardných streamov komunikačnej časti, ktoré sa predajú serveru. Tieto stavy štandardných streamov si používateľ môže pozrieť v priebehu vykonávania. Po dokončení úlohy MoM oznamuje serveru, opäť cez komunikačnú časť informáciu, ktorá signalizuje skončenie úlohy, prípadne k tomu dáta (napr. návratový kód), čím sa uvoľňuje uzol pre ďalšie použitie.

Medzi týmito operáciami ale plánovač vidí ďalšie dve úlohy, zaradené od oboch spomínaných požívateľov, vo fronte B. Tu vzniká najväčší problém plánovania, keďže obaja používatelia žiadajú o ten istý výpočtový uzol. Čitateľa už určite napadne, že toto rozhodovanie sa vykoná na základe nakonfigurovanej politiky plánovania. Politika plánovania, sa môže teda skladať z viacero aspektov, podľa ktorých sa vyberie exekúcia úlohy, ktorá sa má započať. Tieto aspekty sa v tejto práci nachádzajú v texte podkapitoly Plánovače. Plánovač teda vyberie úlohu, ktorú spustí rovnakým procesom komunikácie medzi blokmi, ako je opísané v predchádzajúcom odseku, kde potom druhá úloha musí čakať do konca vykonávania prvej, pretože nie je dostatok zdrojov (uzlov), na ktorých možno úlohu spustiť.

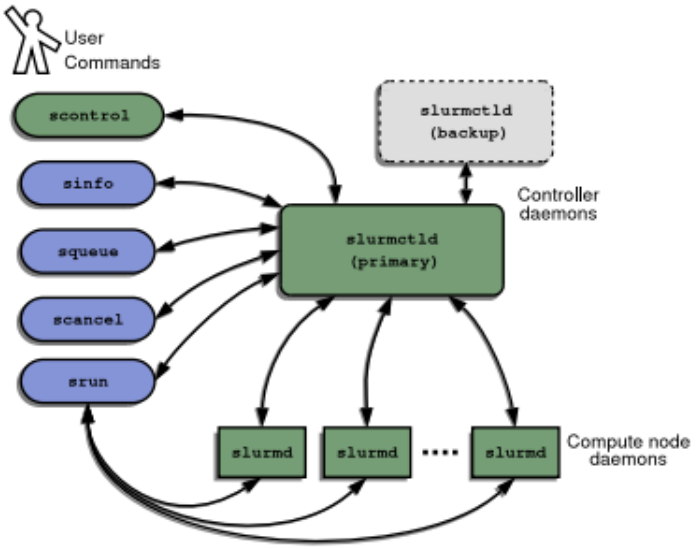
3.2 Simple Linux Utility for Resource Management

Je ďalšia implementácia plánovača pre superpočítače s architektúrou klaster (ďalej len ako SLURM). Nie je výhradne iba pre superpočítače, ale aj pre malé užívateľské klastre pre osobné použitie.

SLURM je open-source, chybovo tolerantný a škálovatelný plánovač. Pre jeho použitie nie je potrebné modifikovať kernel. Jeho úloha je identická ako pri plánovači PBS, teda manažovať exkluzívny alebo neexkluzívny prístup ku zdrojom superpočítača pre vykonanie náročných výpočtov. Rovnako ako PBS, poskytuje framework pre exekúciu, monitorovanie, spravovanie práce na výpočetných uzloch, a taktiež poskytuje riadenie uzlov, ktoré sú zaradené vo frontách uzlov. Princíp front uzlov z PBS má v SLURM meno partícia. Ako z nadpisu vyplýva, plánovač SLURM je možné použiť jedine pre operačný systém Linux.

3.2.1 Architektúra plánovača SLURM

Architektúru SLURM nám opisuje nasledujúci obrázok. Na obrázku je vidieť časti slurmd a slurmctld. Tieto časti slúžia ako démoni plánovača. Časti architektúry označené modrou farbou prezentujú používateľské príkazy. [10]



Obr. 3.4: Architektúra plánovača SLURM [10]

Slurmctld

Démon slurmctld slúži ako kontrolér a je hlavná časť SLURM plánovača. Principiálne z plánovača PBS môže pripomínať Server, Communication Demon a Scheduler v jednom. Ukladá si všetky stavové informácie o superpočítači, a pri reštartovaní ich znova načítava a pracuje s nimi. Taktiež periodicky ukladá na disk stavové informácie.

Používateľ superpočítača, v ktorom pracuje tento plánovač komunikuje práve s týmto kontrolérom pomocou programového rozhrania. To môže pripomínať čelnú časť pri plánovači PBS. Tento proces sa skladá z troch základných komponentov.

Manažér uzlov Monitoruje status každého uzla v klastri. Periodicky vyžaduje dáta od slurmd procesov na uzloch. Pred každým priradením uzlu pre prácu vykoná kontrolu, či uzol správne funguje a je voľný.

Manažér partícii Partície, rovnako ako fronty uzlov pri plánovači PBS, majú svoje limity nakonfigurované administrátormi superpočítača. Manažér partícii spravuje tieto limity a povoľuje ich modifikáciu administrátormi.

Manažér úloh Akceptuje úlohy zadané užívateľom a operuje s nimi. Funkcionalitou pripomína plánovač (scheduler) z plánovača PBS. Zaraďuje a riadi zadané úlohy z partícii na základe politík a aktuálnej obsadenosti uzlov. Taktiež povoľuje užívateľovi pracovať s aktuálnym stavom a dotazovať sa na to, v akom stave úloha je.

Je nutné podotknúť, že SLURM sa snaží predchádzať nehodám systému, preto existuje kópia slurmctld démona, ktorá je v stand-by režime a pri nehode pracujúceho slurmctld démona nastane výmena medzi stand-by slurmctld démonom, a slurmctld démonom, ktorý je v stave nehody.

Slurmd

Slurmd je ďalší démon, ktorý beží na každom výpočtovom uzle klastru a môže byť prirovnaný ku vzdialenému shell démonovi. Z plánovača PBS môže pripomínať časť MoM. Pracuje so SLURM konfiguráciou, spúšťa úlohy a vracia výstupové streamy pre kontrolér slurmctld. Výmena dát medzi slurmd a kontrolérom slurmctld prebieha asynchrónne. Slurmd sa skladá z piatich podstatných komponentov.

Status výpočetného uzla a úlohy Poskytuje kontroléru slurmctld status o aktuálnej úlohe a vyťaženie výpočetného uzla.

Vzdialená exekúcia Manažuje procesy (zvyčajne tie, ktoré patria vykonávanej úlohe) podľa toho, ako žiada slurmctld proces. Nemusí to byť vždy rušenie procesov, ale aj vytváranie procesov, napríklad nastavenie limít, nastavenie premenných prostredia.

Služba kopírovania streamov Manažuje prístup k streamom (STDIN, STDOUT...) úloh. Vstup môže byť presmerovaný zo súborov. Výstup môže byť presmerovaný do lokálnych súborov ,alebo môže byť presmerovaný k používateľovi, ktorý požiadal o status streamu.

Ovládanie práce Umožňuje kontroléru slurmctld ovládať signály pre dané procesy práce.

Scontrol

Scontrol slúži ako nástroj pre SLURM administrátorov pre administráciu plánovača. Umožňuje rôzne nastavenia. Zopár z nich je následne vymenovaných.

- **Shutdown** - Terminácia slurmctld a slurmd a uloženie stavu celého plánovača.
- **Ping** - Zobrazenie statusu pracujúceho slurmctld procesu a záložného (stand-by) slurmctld procesu.

- **Reconfigure** - Rekonfigurácia všetkých slurmctld a slurmd procesov, podľa zadaného konfiguračného súboru.
- **Show Job State** - Zobrazenie stavu ohľadom úloh na uzle alebo uzloch.

Kapitola 4

Infraštruktúra IT4Innovations

Táto kapitola sa zaoberá infraštruktúrou a super-počítačovými službami ponúkanými centrom IT4Innovations. Tieto služby je nutné pochopiť a zaradiť ich do tohto textu, pretože implementácia, všetky merané časy a odhady, sú vykonávané pre tieto superpočítače.

IT4I, národné superpočítačové centrum je súčasťou Technickej univerzity Ostrava. Misia IT4I je realizovať výskum v oblasti veľmi náročných výpočtov, veľmi náročnej dátovej analýzy a prevádzkovať národnú superpočítačovú infraštruktúru a sprostredkovať jej efektívne využitie za cieľom zvyšovania konkurencieschopnosti českej vedy a priemyslu.

IT4I aktuálne ponúka prístup a spravuje dva superpočítače, a to nasledovné Anselm a Salomon. Implementácia tejto bakalárskej práce sa zameriava na tieto dva superpočítače. [2]

4.1 Anselm

Klaster Anselm pozostáva zo 209 výpočtových uzlov. Každý uzol pozostáva z 16 jadier, presnejšie, je zložený z dvoch osemjadrových procesorov (Intel Sandy Bridge). Tieto uzly sa rozdeľujú do špeciálnych skupín podľa ich použitia. V týchto skupinách sa zároveň uzly nachádzajú v rámci fronty v plánovači, keď používateľ žiada o výpočet a vyberá frontu uzlov, ktorú chce použiť. Uzly sa teda delia.

- 180 výpočtových uzlov (2.4 GHz takt CPU, 64 GB RAM)
- 23 výpočtových uzlov s GPU akcelerátormi (2.3 GHz takt CPU, 96 GB RAM a NVIDIA Tesla Kepler K20)
- 4 výpočtové uzly s MIC akcelerátormi (2.3 GHz takt CPU, 96 GB of RAM, Intel Xeon Phi 5110P)
- 2 tlsté výpočtové uzly (2.4 GHz takt CPU, 512 GB RAM)

Všetky uzly sú prepojené vysokorýchlostným prepojením InfiniBand a Ethernet.

Teoretický výkon klastra Anselm je 94 TFlop/s. Všetky uzly zdieľajú 320 TiB disk, ktorý využívajú používatelia na ukladanie súborov. Súborové systémy a ich prepojenie je zabezpečené pomocou LUSTRE paralelného súborového systému. Anselm používa plánovač PBS Professional. Klaster používa operačný systém RedHat Linux.

4.2 Salomon

Klaster Salomon pozostáva z 1008 výpočetných uzlov. Každý uzol sa skladá z 24 jadier, presnejšie z dva dvanásťjadrových procesorov (Intel Haswell). Všetky výpočtové a prihlasovacie uzly sú prepojené vysokorýchlostnou sieťou 7D Infiniband. Uzly sa znova rozdeľujú.

- 576 výpočtových uzlov (2.5 GHz takt CPU, 128 GB RAM)
- 432 výpočtových uzlov s MIC akcelerátorom (2.5 GHz takt CPU, 128 GB RAM, Intel Xeon Phi 7120P 2-krát)

Teoretický výkon klastru naberá 2 PFlop/s. Salomon v dobe vydania (leto 2015) dosiahol 87. miesto v Top 500 superpočítačov na svete. Všetky uzly zdieľajú spoločne 500 TiB disk na ukladanie používateľských súborov. Uzly ďalej zdieľajú špeciálny 1.6 PiB disk, ktorý slúži na ukladanie medzivýpočtov, medzisúbory atď. Klaster taktiež používa plánovač PBS Professional.

K uzlom ešte existuje špeciálne zariadenie skladajúce sa z SMP architektúry podporujúci NUMA rozhranie. Zariadenie je spojené so Salomon klastrom, obsahuje 112 jadier a 3.25 TB RAM. Salomon používa RedHat distribúciu operačného systému Linux.

Kapitola 5

k-Wave

k-Wave je open-source MATLAB súbor nástrojov, ktorý bol vytvorený pre simuláciu propagácie akustických vĺn v 1D, 2D a 3D priestoroch, na základe časovej domény. Na počiatku simulácia prebiehala iba pre rekonštrukciu fotoakustických vĺn v bezstratovom médiu, postupným vývojom ale k-Wave simulácie nabrali viacero funkcionalít a možností simulácii.

Tento toolbox bol originálne vyvíjaný skupinou Photoacousting Imaging Group na University College London. Hlavní autori k-Wave sú Bradley Treeby, Ben Cox a Jiří Jaroš.

5.1 Optimalizované CPU a GPU implementácie

MATLAB je silný nástroj pre výskumné počítanie a obsahuje veľké množstvo knižníc a rozhraní pre počítanie s maticami, komplexnými číslami, vektormi atď. Tieto MATLAB skripty sú vysoko prenositeľné na rôzne operačné systémy a platformy, ktoré MATLAB podporuje.

Na druhej strane ale ide o veľkú nevýhodu a to tú, že ide o interpretovaný jazyk. Kde sa každý kód musí dynamicky analyzovať a vykonávať, čo prináša niekoľkonásobné spomalenie oproti kompilovaným jazykom, a omnoho menšiu kontrolu nad exekúciou príkazov, kde by táto kontrola mohla výpočet zefektívniť. Pre simulácie v malých doménach a dimenziách (1D a 2D), MATLAB implementácia neprináša problém, pretože výpočet trvá krátko a iné implementácie je možné vynechať. Problém ale nastáva pri veľkých rozmeroch a väčších dimenziách, kde 3D simulácia už simuluje priebeh v reálnom svete, a teda predstavuje najväčší počet prípadov použitia. Pri takýchto zložitejších úlohách už výpočetný čas v MATLAB implementácii môže dosahovať desiatky hodín. Pre daný problém vznikli ďalšie implementácie tohto toolboxu implementované v jazyku C++, pre zaručenie optimalizácie výpočtu.

Presnejšie sú to nasledujúce implementácie.

- `kspaceFirstOrder3D-OMP` - C++ implementácia, ktorá používa knižnicu OpenMP pre efektívnu implementáciu viacvláknových programov. Implementácia vytvára paralelizmus v rámci jedného procesora.
- `kspaceFirstOrder3D-CUDA` - C++ implementácia, ktorá využíva knižnicu CUDA pre zrýchlenie náročných výpočtov, kde sa namiesto procesora pre výpočty použije grafická karta.
- `kspaceFirstOrder3D-MPI` - C++ implementácia, ktorá používa knižnicu MPI (Message Passing Interface), pre efektívnu implementáciu paralelného kódu, kde už para-

lelizmus môže prebiehať medzi viacero procesormi pomocou predávania správ a môže sa získať najväčší level paralelizmu.

Danými implementáciami sa zaoberá táto práca a odhad konfigurácii prebieha práve pre tieto implementácie.

5.2 Simulácia (úloha)

Ako je vyššie spomínané, simulácia môže mať rôzne parametre, ktoré rozlišujú detaily výpočtu a ovplyvňujú tak jej dĺžku výpočtu. Najdôležitejšie parametre sú nasledujúce.

- **Homogenita** - médium môže byť homogénne alebo heterogénne. Tento parameter popisuje, aké vlastnosti má médium.
- **Linearita** - určuje vlasnosť propagácie vĺn. Vlny sa môžu šíriť lineárne alebo nelineárne, sú použité lineárne a nelineárne rovnice pre simuláciu.
- **Absorbcia** - určuje vlastnosť média, kde médium propagovanú vlnu postupne zoslabuje a absorbuje. Médium môže, ale nemusí byť absorpčné.

V implementácii k-Wave existuje ešte rada možností. Tieto sú ale najdôležitejšie, presnejšie táto práca a odhad sa nimi zaoberá.

- Rozmer úlohy
- Počet časových krokov

Jeden z najdôležitejších parametrov, je rozmer úlohy. Úlohy ktorými sa práca zaoberá sú 3-rozmerné, pretože trvajú najdlhšie a sú najviac používané pri k-Wave, pretože dokážu simulovať náš svet. Úlohy menších dimenzii bývajú vypočítané veľmi rýchlo, kedy meranie a odhad stráca význam. Úloha teda vyzerá nasledujúce.

$$N = X \times Y \times Z \tag{5.1}$$

Kde:
N - Reprezentuje celkovú veľkosť úlohy
X - Reprezentuje veľkosť 1. dimenzie úlohy
Y - Reprezentuje veľkosť 2. dimenzie úlohy
Z - Reprezentuje veľkosť 3. dimenzie úlohy

Tieto rozmery reprezentujú množstvo diskrétnych bodov, cez ktoré sa rovnice a transformácie počítajú. Z toho vyplýva, že čím väčší rozmer je, tým viac sa počíta.

Ďalším dôležitým parametrom, je množstvo časových krokov. Množstvo časových krokov reprezentuje dĺžku, ako dlho sa v simulovanom svete bude simulácia vykonávať (nie skutočný čas). Tento počet silne ovplyvňuje výpočetný čas simulácie. Čím viac časových krokov je vyžiadaných, tým dlhšie sa simulácia počíta.

Hlavné a zvyšné parametre, prichádzajú ako vstup pre vymenované k-Wave implementácie. Parametre sú uložené v type súboru HDF5, s ktorým následne implementácie pracujú a vyberajú potrebné informácie. Výsledok priebehu simulácie je taktiež uložený v HDF5 súbore, ktorý obsahuje hlavný priebeh a ďalšie metadáta o jej priebehu, ako aj štatistiky.

5.2.1 Vhodný a nevhodný rozmer simulácie

K-wave implementácie spomínané v časti 5.1 používajú pre výpočty v majorite Fourierové transformácie a to ich rôzne implementácie, presnejšie používa Fast Fourier Transformation (ďalej ako FFT) pre rýchlosť a efektivitu výpočtu simulácie. Rýchlosť FFT zakladá na veľkosti rozmerov, presnejšie na prvočíselnom rozklade týchto rozmerov. Platí že, čím menší je rozklad prvočísiel, tým efektívnejšie môže byť simulácia vypočítaná. Preto, každá knižnica, ktorú používajú implementácie k-Wave definuje vhodné veľkosti prvočísiel od ktorých sa dá očakávať efektívne a správne chovanie. Tieto knižnice sú nasledujúce.

- **FFTW** (použité MPI implementáciou k-Wave) - prvočísla $2, 3, 5$ a 7
- **Intel MKL** (použité OMP implementáciou k-Wave) - prvočísla $2, 3, 5, 7$ a 11
- **cuFFT** (použité CUDA implementáciou k-Wave) - prvočísla $2, 3, 5$ a 7

Z tohto je vždy možné usúdiť, či očakávať správne chovanie simulácie na základe rozmeru vstupnej úlohy, a či je teda v silách plánovača približnú dobu exekúcie aproximovať.

Kapitola 6

Odhad na základe empirických dát

Nasledujúca kapitola sa zaoberá problematikou odhadovania a aproximácie nových dát z empirických dát (skúseností). Pretože kapitola je veľmi rozsiahla a je možné sa na danú problematiku pozerať z viacero uhlov, táto práca obsahuje popis spôsobov odhadovania, vďaka ktorým sa dosiahla jej implementácia. Práca používa nasledujúce skupiny metód.

- Extrapolácia - pre odhadovanie nových parametrov, ktoré sa nachádzajú mimo iných známych parametrov
- Interpolácia - pre odhadovanie nových parametrov, ktoré sa nachádzajú medzi inými známymi parametrami
- Pomer a priemer - pre odhadovanie hodnôt nových parametrov, na základe lineárnych koeficientoch v pomere známych parametrov

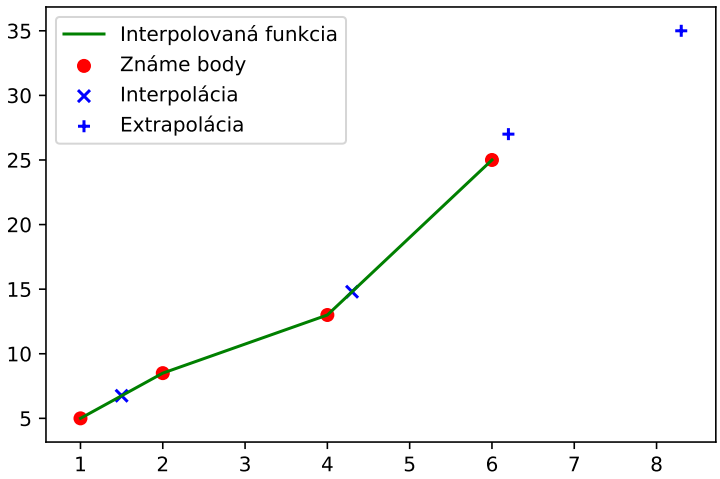
6.1 Interpolácia a Extrapolácia

Interpolácia a extrapolácia sú techniky použité pre nález predpisu analytickej funkcie, ktorá prechádza množinou zadaných bodov. Následne na základe tejto nájdenej funkcii môžeme predkladať vstup a získame približný výstup. [3]

- **Interpolácia** - Dosadený vstup leží v rozmedzí zadaných bodov.
- **Extrapolácia** - Dosadený vstup leží mimo rozmedzia zadaných bodov.

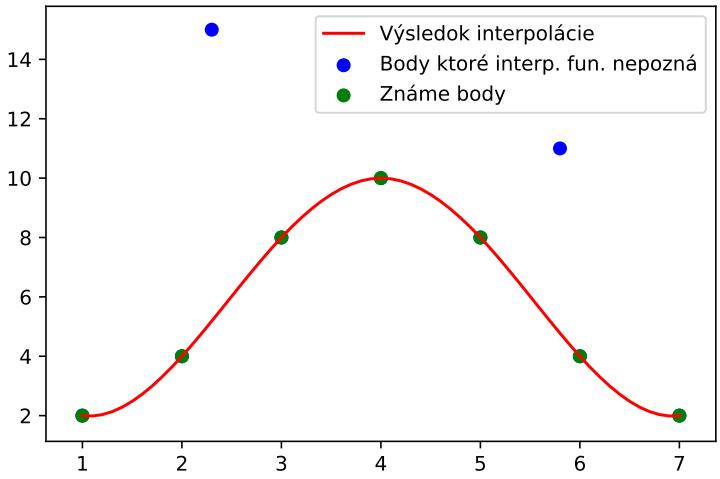
Interpolácia a extrapolácia prekladá vstupné body rôznymi útvarmi na základe vybranej metódy. Pretože vstup pre extrapoláciu leží mimo bodov, ktoré poznáme, výsledky extrapolácie bývajú omnoho menej presné.

Využitie týchto matematických metód je veľmi vhodné pre skúmaný problém pretože z nameraných diskrétnych hodnôt (behov), ktoré na sebe logicky závisia, je možné vytvoriť spojitú funkciu, ktorá túto logickú spojitosť medzi hodnotami sa snaží napodobniť a z toho je následne možné predikovať nové hodnoty (konfigurácie) pre neznáme simulácie.



Obr. 6.1: Interpolácia a Extrapolácia

Interpolácia ale taktiež môže byť dosť nepresná a vytvárať chyby. Ako príklad je možné uviesť 4 body, ktoré prejavujú vhodné správanie, kde ale v niektorých ďalších neznámych bodoch medzi týmito známymi bodmi vzniká veľký rozdiel, a pre dané body je výsledná hodnota vzdialenosti iná, a interpolácia nemá ako odhadnúť tieto hodnoty.



Obr. 6.2: Chyba, ktorú metóda neodhadne

Metód pre interpoláciu a extrapoláciu je mnoho, táto práca opisuje tie najdôležitejšie a tie, ktoré sa použili pre implementáciu.

6.1.1 Lineárna interpolácia

Najzákladnejšia a najjednoduchšia metóda interpolácie, jednoduchá pre implementáciu a výpočet.

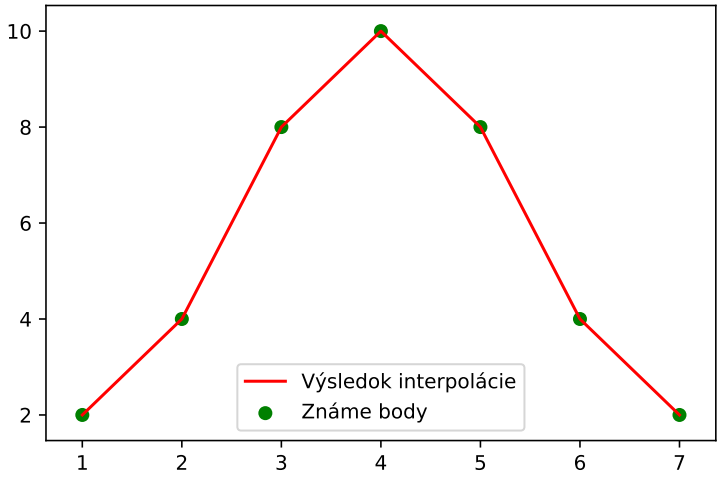
Známe diskrétne body sú prekladané polynómom prvého stupňa. A to presnejšie, pokiaľ poznáme dva body so súradnicami (x_0, y_0) a (x_1, y_1), lineárna interpolácia je priamka medzi týmito dvoma bodmi. Daná hľadaná hodnota x leží v rozmedzí (x_0, x_1), hodnota y je ale daná rovnicou. [3]

$$\frac{y_1 - y_0}{x_1 - x_0} = \frac{y - y_0}{x - x_0} \tag{6.1}$$

Následným odvodením rovnice pre hodnotu y sa dostáva.

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{6.2}$$

Lineárna interpolácia predpokladá, že hodnoty medzi dvoma známymi diskrétnymi bodmi rastú lineárne. Tento proces je nutné opakovať pre dvojice bodov medzi sebou, kde platí $x_0 < x_1$. Viacero bodov je nutné prekladať medzi sebou. Týmto vznikne kostrbatý priebeh medzi bodmi, pretože sú spojené lineárnymi krivkami.



Obr. 6.3: Príklad lineárnej interpolácie

6.1.2 Interpolácia polynómom

Jedna zo zložitejších metód interpolácie. Je zovšeobecnením lineárnej interpolácie, pretože výsledok lineárnej interpolácie je vždy polynóm prvého stupňa, kde ale výsledok interpolácie polynómom je vždy výsledok polynóm N-tého stupňa. Metóda zakladá na dvoch teorémoch. [3]

Teorém č.1 Každý polynóm N-tého stupňa, kde stupeň je > 0, má aspoň N koreňov (riešení). Nasledujúci teorém podporuje platnosť interpolácie polynómom.

Teorém č.2 Pre N nerovnakých bodov $x_1, x_2, ..., x_n$ a ich funkčných hodnôt $f(x_1), f(x_2), ..., f(x_n)$ existuje práve jeden polynóm p stupňa menší alebo rovný n pre ktorý platí

$$p(x_i) = f_i, i = 1, ..., n \tag{6.3}$$

Na základe týchto teorémov je možné konštruovať polynómy obecne viacero spôsobmi. A to napríklad.

- Lagrangerov tvar interpoľačného polynómu
- Newtonov tvar interpoľačného polynómu

Následne práca popisuje spôsob interpolovania pomocou Newtonovho tvaru, pretože tvar vyzerá lepšie čitateľný ako Lagrangerov, a jednoduchšie môže prijať nové uzly bez potreby prepočítavať celý polynóm.

Newtonov tvar interpoľačného polynómu

Newtonov tvar interpoľačného polynómu sa zakladá na nasledovnom tvare. [3]

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x1) + ... + a_n(x - x_0)(x - x_1)...(x - x_{n-1}) \quad (6.4)$$

Koeficienty a_i je možné určiť rôznymi spôsobmi, jeden zo spôsobov je pomocou *pomerových diferencii*.
Pre danú funkciu f a uzlové body $x_i, i = 0, ..., n$ vytvoríme podiely pomerovými diferenciami prvého rádu.

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, i = 0, 1, ..., n - 1 \quad (6.5)$$

Potom obecne pomerové diferencie K-tého radu pre $K \leq n$ definujeme takto.

$$f[x_i, x_{i+1}, ..., x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+k}] - f[x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i} \quad (6.6)$$

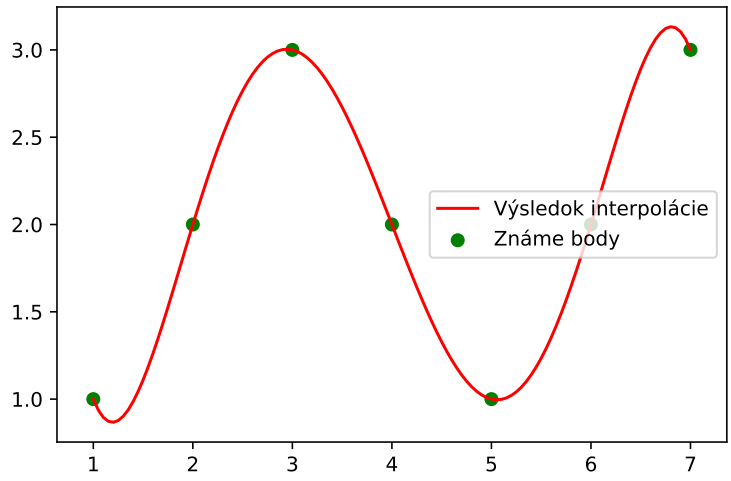
Dá sa dokázať, že pre koeficienty $a_i, i = 0, 1, ..., n$ platí.

$$\begin{aligned} a_0 &= f(x_0) \\ a_1 &= f[x_0, x_1] \\ a_n &= f[x_0, x_1, ..., x_n] \end{aligned} \quad (6.7)$$

Dosadením následne dostaneme Newtonov interpoľačný polynóm.

$$\begin{aligned} P_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + ... \\ + f[x_0, x_1, ..., x_n](x - x_0)(x - x_1)...(x - x_{n-1}) \end{aligned} \quad (6.8)$$

Následne dosadením x do výsledného $P_n(x)$ dostávame výsledok interpolovanej hodnoty, podľa vzniknutej interpolovanej funkcie. Výsledná interpolácia vyzerá nasledovne.



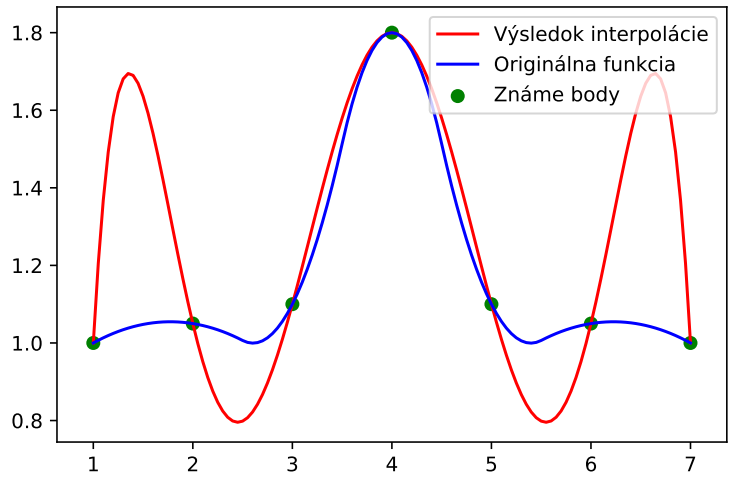
Obr. 6.4: Výsledok interpolácie polynómom

Problém interpoľačného polynómu

Interpoláciu môžeme využiť pri dvoch formuláciách problému, a to.

- Existujú body, ktoré popisujú chovanie javu, resp. popisujú funkciu, podľa ktorej sa jav chová a je pre riešiteľov funkcia neznáma. Nutno zistiť túto funkciu.
- Riešiteľ má zadanú funkciu, ktorá popisuje chovanie javu. Funkcia je zložitá pre analytické riešenie. Táto zložitá funkcia sa nahradí interpolovanou, a riešiteľ pracuje so zjednodušenou variantou.

Pretože pointa je nájsť riešenie funkcie (zodpovedajúce hodnoty) v neznámom mieste, je nutné dávať pozor na presnosť a chovanie interpolovanej funkcie, a vlastnosti interpolovaciej metódy.



Obr. 6.5: Rungeho jav a oscilácia výsledku

Tu nastupuje Rungeho jav. Na obrázku je vykreslená funkcia a jej interpolácia, kde vidieť že interpolácia je veľmi nepresná a naberá značnú osciláciu. Ako je vidieť, polynóm nie je vhodné použiť mimo rozpätia prvého a posledného známeho (zadaného) uzla, resp. extrapolácia. Taktiež je vidieť, že aj vo vnútri intervalu vzniká veľká oscilácia a interpolovaná funkcia naberá odchýlku so zvyšujúcou sa vzdialenosťou od známeho (zadaného) bodu. Podobná situácia môže nastať aj pri polynóme vysokého stupňa. Ako bolo predtým vysvetlené, stupeň polynómu závisí od známych (zadaných) bodov, a to $x \leq N$ (kde x je stupeň). Ak je známych vstupných bodov veľa, stupeň polynómu rastie a vzniká veľká oscilácia.

Tento pojem je relatívny a problémy vznikajú na základe konkrétnej situácie.

6.1.3 Spline

Spline je další spôsob interpolácie, ktorý problém rieši iným spôsobom ako metódy spomínané doteraz. Predchádzajúce metódy sa snažia nájsť jeden funkčný predpis pre celý interval hodnôt. Namiesto toho sa ale spline snaží nájsť funkčný predpis, ktorý je na intervaloch definovaný rôzne.

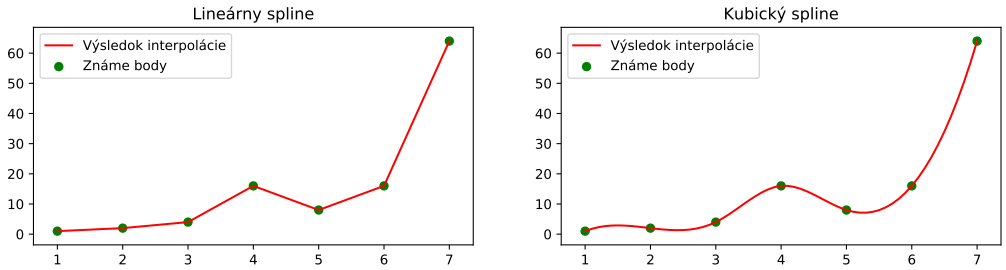
Spline rozdelí interval od najmenšieho známeho (zadaného) bodu po najväčší známy bod do podintervalov, kde každý pod-interval bude vždy v tvare $< x_i, x_{i+1} >$, tzn. bude vždy intervalom medzi dvoma známymi bodmi, a na každom tomto intervale bude existovať vhodný interpoľačný funkčný predpis. Je nutné aby spline spĺňal nasledujúce podmienky. [3]

- Funkcia musí prechádzať všetkými zadanými bodmi ako u predchádzajúcich interpoľačných metód
- Všetky časti funkcie (pod-intervaly) musia byť polynómy obecne rovnakého stupňa
- Susedné časti (pod-intervaly) musia vzájomne na seba nadväzovať, až do derivácie daného radu

Najzákladnejšie a najpoužívanejšie druhy spline existujú:

- **Lineárny spline** - Pod-intervaly sú tvorené úsečkou (Polynóm 1. stupňa)
- **Kubický spline** - Pod-intervaly sú tvorené polynómom 3. stupňa

Jedna z najväčších výhod použitia splinu, spočíva v použití nízkych úrovní polynómov, ktoré sú presné ako vysoké úrovne pri interpolovaní polynómami, čím sa zbavuje prítomnosti Rungeho javu.



Obr. 6.6: Porovnanie lineárneho a kubického splinu

Kubický spline

Kubický spline pre funkciu f s uzlovými bodmi $x_0, x_1, ..., x_n$ je funkcia $S(x)$, ktorá je kubický polynóm, označený $S_i(x)$ na každom podintervale $< x_i, x_{i+1} >, i = 0, 1, ..., n-1$, ktorý vyhovuje podmienkam. [3]

$$S_i(x_i) = f(x_i), i = 0, ..., n-1, S_{n-1}(x_n) = f(x_n) \tag{6.9}$$

Podmienka hovorí o zadaných (známych) bodoch, ktoré sa musia rovnať a nachádzať v daných x.

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1}), i = 0, ..., n-2 \tag{6.10}$$

Podmienka hovorí o spojitosti funkcie S (výsledného splinu) v zadaných (uzlových) bodoch.

$$\begin{aligned} S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}), i = 0, ..., n-2 \\ S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1}), i = 0, ..., n-2 \end{aligned} \tag{6.11}$$

Podmienky hovoria o spojitosti 1. a 2. derivácie funkcie S. Kde

$$S''(x_0) = S''(x_n) = 0 \tag{6.12}$$

je okrajová podmienka.

Výsledok interpolácie (spline) $S(x)$ na jednotlivých intervaloch $< x_i, x_{i+1} >, i = 0, 1, ..., n-1$, je definovaný a hľadaný ako.

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{6.13}$$

Tým, že kubický spline sa skladá z polynómov 3. stupňa vytvára interpoľačnú funkciu v ktorej sa (zvyčajne) nenachádza veľká oscilácia a polynómy v pod-intervaloch vyhladzujú výslednú inteprolovanú funkciu.

6.2 Regresia

Regresia je výsledok regresívnej analýzy, kde regresívna analýza slúži ku popisu závislosti dvoch alebo viacerých číselných premenných, a hľadáme ich matematický model. Regresia je matematický model, regresná funkcia, ktorá by mala vyjadrovať charakter závislosti a priebeh týchto numerických premenných. Regresná funkcia slúži pre odhad hodnôt a stredných hodnôt.

Dosadzovaním do výslednej regresnej funkcie je teda možné získať neznámu odhadovanú hodnotu. U regresie sa vyskytujú dva druhy premenných.

- Závislé premenné - Predstavujú výstup regresie, predikciu.
- Nezávislé premenné - Predstavujú vstup, na základe ktorého vznikajú závislé premenné.

Na základe počtu závislých a nezávislých premenných sa regresia delí.

- Jednoduchá regresia - pre popis dvoch číselných premenných. Hľadaný model je reprezentovaný ako $y = f(x)$. Jedna závislá a jedná nezávislá premenná.
- Viacnásobná regresia - pre popis závislosti jednej numerickej premennej (závislej) na základe skupiny numerických (nezávislých) premenných $y = f(x_1, x_2, ..., x_n)$.

Následne na základe charakteristiky dát je nutné zvoliť vhodný typ regresie, to môže byť napríklad.

- Lineárna regresia
- Regresia polynómom
- Logistická regresia

6.3 Lineárna regresia

Lineárna regresia je typ regresie, ktorý modeluje vzťah medzi jednou závislou premennou a jednou alebo viacerými nezávislými premennými.

Máme n nameraných dvojíc (x_i, y_i), pre ktoré chceme vytvoriť regresnú funkciu, táto funkcia bude v tvare $y = f(x)$. Regresná funkcia sa môže hľadať rôznymi spôsobmi, najčastejšie ale metódou najmenších štvorcov.

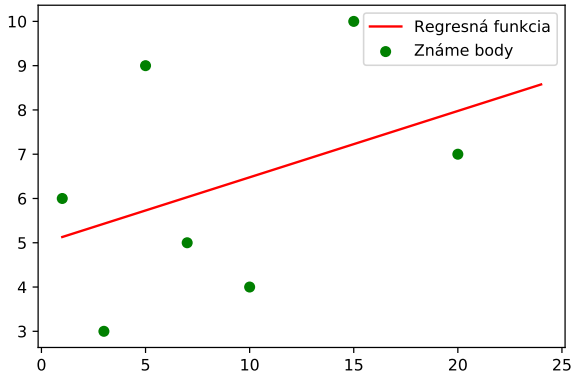
Hľadáme takú funkciu $f(x)$, ktorá má minimálny súčet druhých mocnín rozdielu y súradníc nameraných bodov a bodov ležiacich na regresnej krivke.

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{6.14}$$

Výsledkom je priamka v tvare.

$$f(x) = k \times x + q \tag{6.15}$$

Kde táto priamka je výsledná regresná funkcia (model).



Obr. 6.7: Výsledok lineárnej regresie

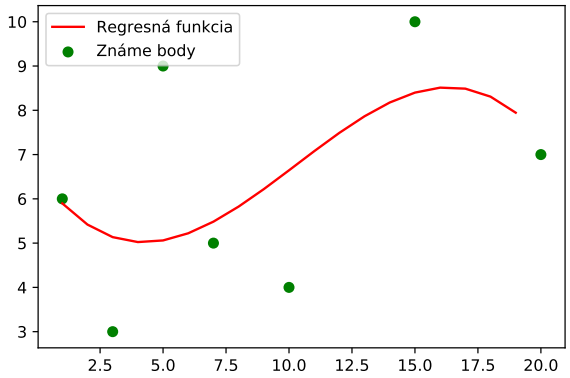
6.4 Regresia polynómom

Regresia polynómom je ďalší spôsob regresie, ktorý popisuje a aproximuje vzťah jednej nezávislej premennej a jednej alebo viacerých závislých premenných.

Narozdiel od lineárnej regresie, ktorej výsledná regresná funkcia je priamka, táto regresia ponúka ako výsledok polynóm daného n rádu. Zvolený rád polynómu by mal závisieť na počte nezávislých premenných, vlastnostiach známych dát, a ďalších faktorov. Výsledný polynóm potom vyzerá nasledovne.

$$y_i = a_0 + a_1 \times x_i + a_2 \times x_i^2 + ... + a_n \times x_i^n + e \tag{6.16}$$

Kde e je koeficient veľkosti chyby. Koeficienty polynómu, môžu byť rovnako ako pri lineárnej regresii odvodené rôznymi spôsobmi, napríklad metódou najmenších štvorcov. Výsledok takejto regresie potom vyzerá približne.



Obr. 6.8: Výsledok regresie polynómom 3. stupňa

Kapitola 7

Cieľ, návrh a architektúra práce

Táto kapitola popisuje návrh a architektúru plánovača. Na začiatku sa text snaží vysvetliť cieľ práce, zamyslieť sa nad pracovným tokom plánovača a podstatou funkcionality. Ďalej rozoberá návrh a časti architektúry, z ktorých sa plánovač bude skladať.

7.1 Cieľ práce

Cieľom práce je vytvoriť algoritmus, ktorý pre zadanú vstupnú simuláciu (úlohu) (kap. 5.2) a zadanú verziu toolboxu k-Wave (kap. 5) nájde optimálne hodnoty spustenia na superpočítači (kap. 2) - klastroch IT4I (kap. 4), kde optimálne hodnoty budú konfigurácia pre PBS plánovač (kap. 3.1). Algoritmus v práci sa ďalej bude nazývať ako "plánovač". Nájsť optimálne hodnoty znamená nájsť vhodný počet vlákien (počet uzlov - jadier) a exekučný čas, ktorý exekúcia simulácie zaberie na tomto počte vlákien. Plánovač je možné matematicky reprezentovať ako.

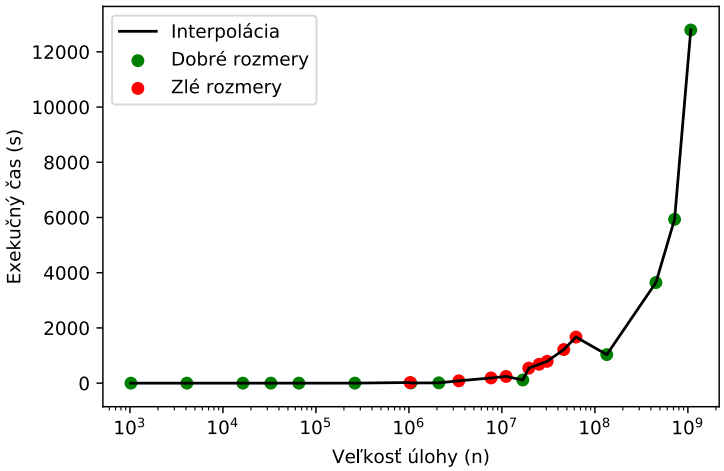
$$f(S, A) = (N, T) \tag{7.1}$$

Kde vstupmi sú:
S - Veľkosť úlohy (simulácie)
A - Atribúty akustického média v úlohe, spomínané v 5.2

A výstupmi:
N - Počet jadier, na ktorých bude ideálne úlohu spustiť
T - Veľkosť exekučného času, ktorý výpočet simulácie zaberie

Hlavnou úlohou plánovača, je teda poskytnúť predikciu času a počtu vlákien optimálnych pre výpočet na základe parametrov simulácie, ktorá bude na superpočítači spúšťaná. Plánovač bude využívať znalostnú databázu, z ktorej čerpá informácie o známych spustených simuláciách, presnejšie o ich parametroch, exekučných časoch, počtoch vlákien... Na základe týchto znalostí, vykonáva aproximáciu spomínaných výstupných hodnôt. Aproximácia spúšťacích hodnôt je vykonaná pomocou metód spomínaných v časti 6 a pokročilejších metód.

Nasledujúci obrázok zobrazuje priebeh zopár známych behov rôzne veľkých k-Wave simulácií. Tu vzniká problém a hlavná podstata, ktorú práca rieši. Ako predikovať neznáme behy vzhľadom na to, že nevhodné rozmery kazia a vytvárajú šum pri vhodných úlohách a naopak.



Obr. 7.1: Porovnanie všeobecného priebehu jedného typu simulácie (rovnaké vlastnosti média), pri rôznych veľkostiach média a vhodnosti rozmerov média (100 kroková simulácia)

7.2 Algoritmus plánovača

Algoritmus plánovača, ako aproximovať tieto výsledky sa dá najjednoduchšie vyjadriť niekoľkými krokmi.

1. Analyzovať zadanú vstupnú úlohu a jej vlastnosti.
2. Na základe týchto vlastností sa rozhodnúť, ako aproximáciu vykonať.
3. Načítať podobné výsledky zo znalostnej databázy, ako podobné sú výsledky záleží na analýze vstupnej úlohy.
4. Výsledky zoradiť a vykonať ich úpravy.
5. Výsledky zoradiť do listov, reprenzetujúce body na ose X a Y.
6. Listy so známymi výsledkami predať aproximátorom, ktoré vykonávajú interpoláciu alebo regresiu.
7. Prebrať výsledky z aproximátorov, urobiť nad nimi úpravy v podobe výpočtov a pridať časovú poistku.
8. Predať finálne výsledky aproximácie používateľovi.

Plánovač bude pracovať iba na základe vyžiadania. Tým, že databáza, ktorá obsahuje výsledky musí byť perzistentná a jej obsah sa mení na základe pridávania dát, rovnako sa bude meniť správanie a predikcia plánovača.

Plánovač a jeho algoritmus sa potom dá špecifikovať na základe niekoľko use-case prípadov. Tieto prípady môžu byť.

- Je vyžiadaný odhad exekučných parametrov úlohy pre binárnu verziu k-Wave OMP
  - a je zadaný počet vlákien, pre ktorý sa má odhadnúť dĺžka exekučného času.

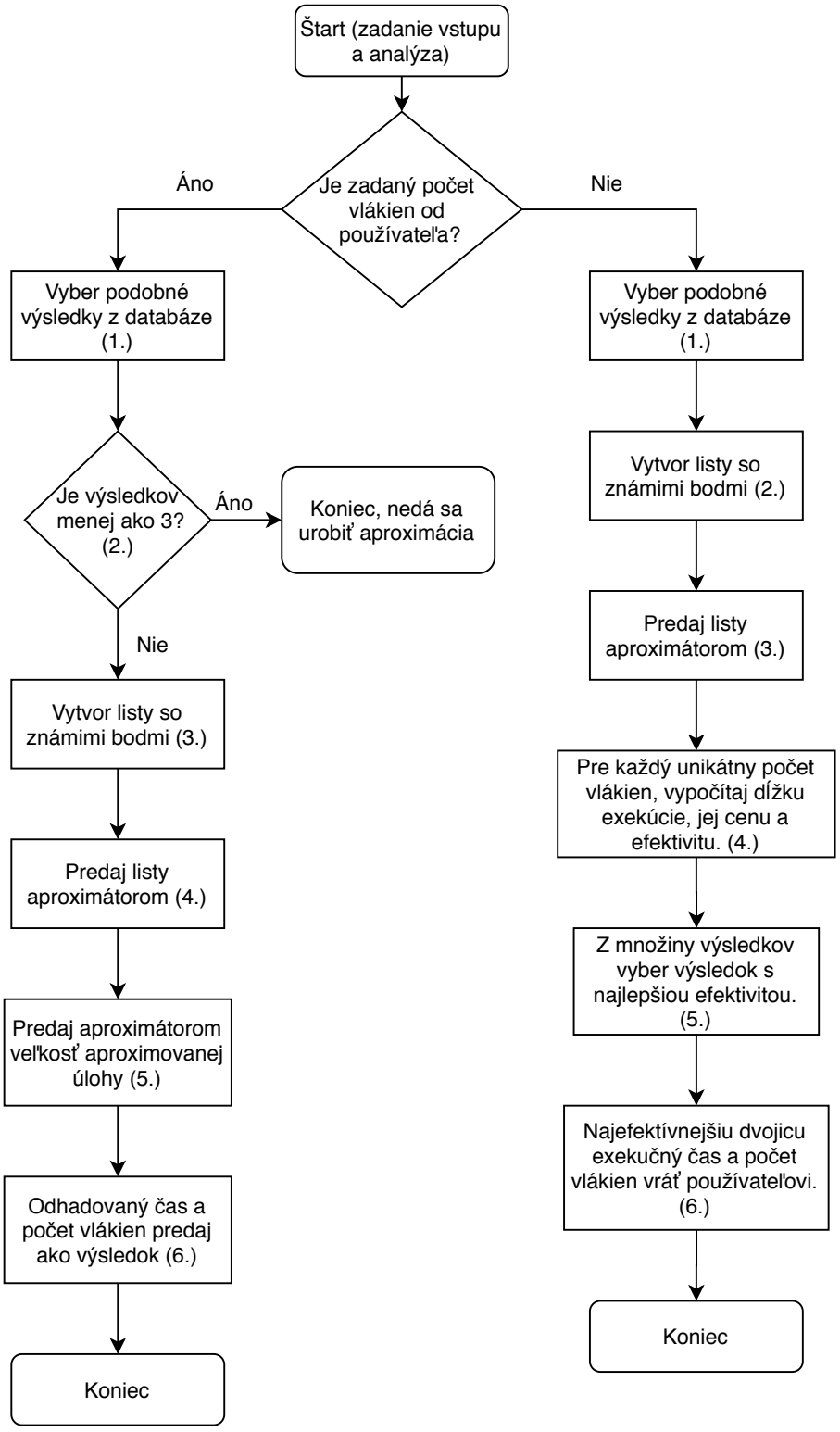
  - a nie je zadaný počet vlákien. Aproximovaný výsledok je dvojica počet vlákien a exekučný čas.

- Je vyžiadaná aproximácia pre binárnu verziu k-Wave CUDA
  - a je zadaný počet vlákien, pre ktorý sa má odhadnúť dĺžka exekučného času.
  - a nie je zadaný počet vlákien. Aproximovaný výsledok je dvojica počet vlákien a exekučný čas.

- Je vyžiadaná aproximácia pre binárnu verziu k-Wave MPI
  - a je zadaný počet vlákien, pre ktorý sa má odhadnúť dĺžka exekučného času.
  - a nie je zadaný počet vlákien. Aproximovaný výsledok je dvojica počet vlákien a exekučný čas.

7.2.1 k-Wave OMP a CUDA

Vývojový diagram 7.2 popisuje algoritmus odhadu pre k-Wave OMP a CUDA verzie, časti obsahujú v zátvorke písmená alebo čísla, ktoré referujú na podrobné vysvetlenie tohto procesu v krokoch vymenovaných v tejto sekcii ďalej. Pre zadaný počet vlákien sa algoritmus špecifikuje.



Obr. 7.2: Vývojový diagram pre predikciu k-Wave OMP a CUDA

1. Zo znalostnej databázy plánovač vyberie výsledky, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer) a počet vlákien. Tieto výsledky budú zoradené podľa veľkosti úlohy od najmenšieho po najväčšie.
2. Ak je výsledkov menej ako tri, aproximácia sa zastavuje a nepredáva sa výsledok, vracia sa výnimka.
3. Z týchto výsledkov sa vytvoria dva listy, ktoré reprezentujú body na ose X a Y.
   - List X - Obsahuje veľkosti rozmerov úloh na ose X.
   - List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.
4. Listy sa predajú aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.
5. Týmto aproximačným funkciám sa predá na vstup veľkosť požadovanej úlohy, funkcie vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v časti 8.6.6.
6. Výsledok sa predá používateľovi.

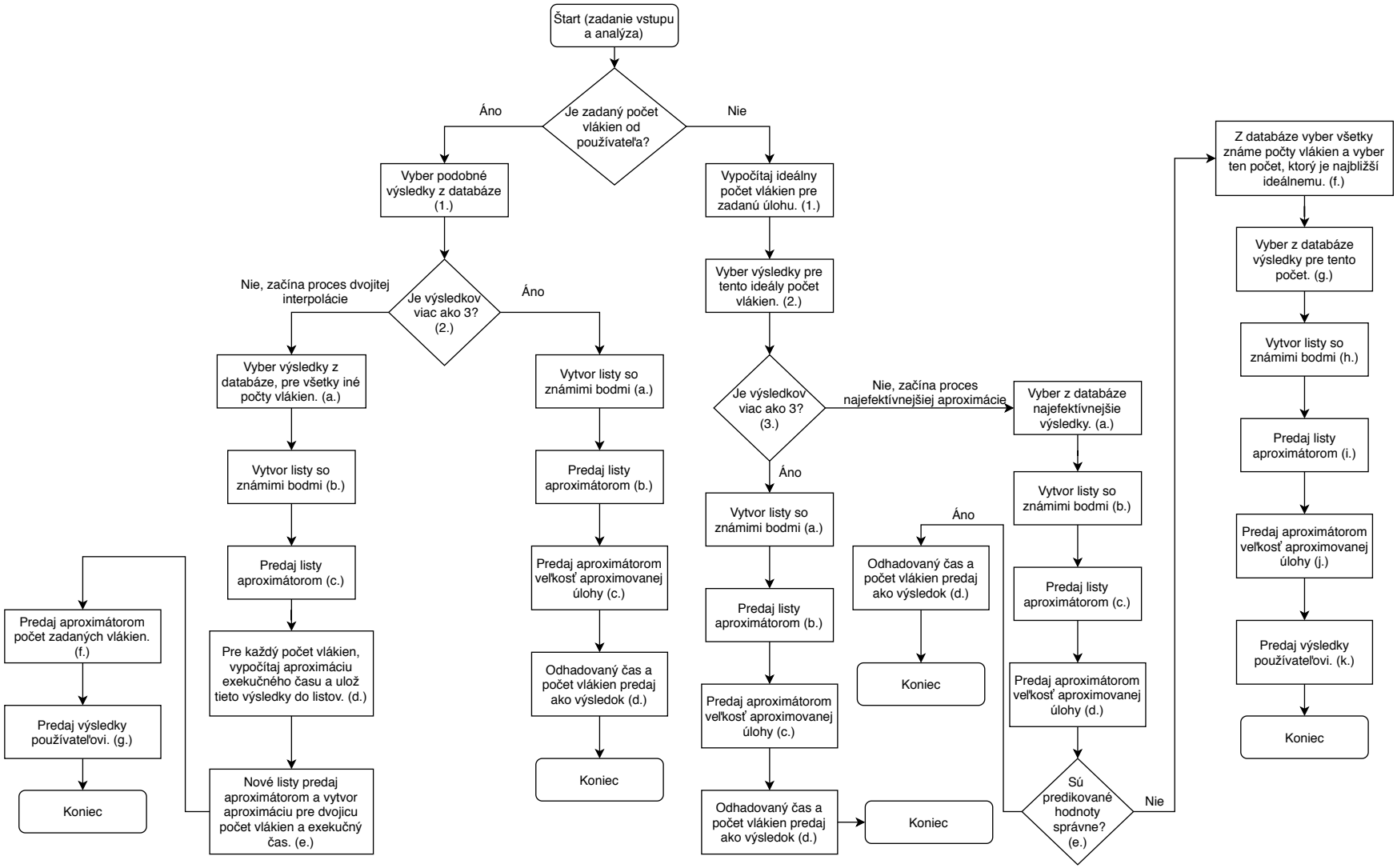
Pre nezadaný počet vlákien sa algoritmus špecifikuje.

1. Zo znalostnej databázy plánovač vyberie výsledky, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer). Tieto výsledky sa zoradia do množín, kde pre jednu množinu výsledkov platí, že boli spustené na rovnako počte vlákien.
2. Pre každú túto množinu, sú vytvorené dva listy reprezentujúce body na ose X a ose Y.
   - List X - Obsahuje rozmery úloh na ose X.
   - List Y - Obsahuje dĺžky exekučného času úloh z osy X na ose Y.
3. Tieto dvojice listov pre každú skupinu sú následne predané aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.
4. Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, a funkcie vrátia exekučný čas pre každý počet vlákien z množiny. Týmto vzniknú dvojice (počet vlákien, exekučný čas), z ktorých je nutné vybrať najefektívnejšiu dvojicu vzhľadom k jej cene.
5. Pre každú z dvojice sa vypočíta jej cenová náročnosť a z množiny dvojíc sa vyberie tá najmenej cenovo náročná. Výpočet cenovej náročnosti prebieha podľa vzorca (8.1). Nakoniec prebehne pre vybranú dvojicu finálny výpočet exekučného času podľa výpočtu v časti 8.6.6.
6. Najefektívnejšia dvojica sa navráti používateľovi.

Presná implementácia algoritmu v plánovači je rozobratá v sekcii 8.6.3 (OMP) a 8.6.5 (CUDA).

7.2.2 k-Wave MPI

Vývojový diagram 7.3 popisuje algoritmus pre odhad pre k-Wave MPI verziu, časti obsahujú v zátvorke písmená alebo čísla, ktoré referujú na podrobné vysvetlenie tohto procesu v krokoch vymenovaných v tejto sekcii ďalej. Pre zadaný počet vlákien sa algoritmus špecifikuje.



Obr. 7.3: Vývojový diagram algoritmu pre predikciu k-Wave verzie MPI

1. Zo znalostnej databázy plánovač vyberie výsledky, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer) a počet vlákien. Tieto výsledky budú zoradené podľa veľkosti úlohy od najmenšieho po najväčšie.

2. Pretože verzia MPI môže bežať na rôznom počte jadier a procesorov, je možné že takéto výsledky pre daný počet vlákien nebudú existovať. Ak výsledky neexistujú, ich počet je menší než 3, započína proces dvojitej interpolácie.

   (a) Zo znalostnej databázy sa vyberú výsledky, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer). Tieto výsledky sa zoradia do množín, kde pre jednu množinu výsledkov platí, že boli spustené na rovnako počte vlákien.

   (b) Pre každú túto množinu, sú vytvorené dva listy reprezentujúce body na ose X a ose Y.

   - List X - Obsahuje rozmery úloh na ose X.
   - List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.

   (c) Tieto dvojice listov pre každú skupinu sú následne predané aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

   (d) Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, a funkcie vrátia exekučný čas úlohy. Toto sa vykoná pre každú množinu známych výsledkov, kde tieto výsledky majú spoločný počet vlákien. Aproximované exekučné časy vo vzťahu k počtom vlákien sa znova uložia do dvoch listov reprezentujúcich body na ose X a Y.

   - List X - Obsahuje množinu počtu vlákien, pre ktoré plánovač vedel aproximovať exekučný čas zadanej úlohy.
   - List Y - Obsahuje veľkosti exekučného času zadanej úlohy pre počty vlákien z osy X na ose Y.

   (e) Týmto plánovač dostane len aproximované exekučné časy zadanej vstupnej úlohy pre iné počty vlákien ako používateľ zadal, preto je nutné naviac z týchto informácii aproximovať exekučný čas pre presne zadaný počet vlákien od používateľa. Následne sa tieto dva nové listy predajú znova aproximátorom, ktoré opäť vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

   (f) Týmto aproximačným funkciám sa predá počet zadaných vlákien, ktoré vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v sekcii 8.6.6.

   (g) Výsledky sa vrátia používateľovi.

3. Ak existujú výsledky, ich počet je väčší alebo rovný ako 3.

   (a) Z týchto výsledkov sa vytvoria dva listy, ktoré reprezentujú body na ose X a Y.

   - List X - Obsahuje rozmery úloh na ose X.
   - List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.

   (b) Listy sa predajú aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

(c) Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, funkcie vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v časti 8.6.6.

(d) Výsledok sa predá používateľovi.

Pre nezadaný počet vlákien sa algoritmus špecifikuje.

1. Plánovač vypočíta odporúčaný počet vlákien podľa vzorca odôvodneného v časti 8.2.
2. Zo znalostnej databázy sa vyberú výsledky, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer) a majú rovnaký počet vlákien, ako plánovač vypočítal.
3. Ak existujú výsledky, ich počet je väčší alebo rovný 3.
   (a) Z týchto výsledkov sa vytvoria dva listy, ktoré reprezentujú body na ose X a Y.
      - List X - Obsahuje rozmery úloh na ose X.
      - List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.

   (b) Listy sa predajú aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

   (c) Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, ktoré vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v sekcii 8.6.6.

   (d) Výsledok sa predá používateľovi.
4. Pretože verzia MPI môže bežať na rôznom počte procesorov (jadier), je možné že takéto výsledky pre daný počet vlákien nebudú existovať. Ak výsledky neexistujú, ich počet je menší než 3, započne sa proces najviac efektívnej aproximácie, podrobnejšia implementácia v časti 8.6.4.
   (a) Zo znalostnej databázy sa vyberú výsledky, pre ktoré platí, že ich rozmer veľkosti úlohy sadol najlepšie možne na daný počet vlákien výsledku, tento najlepší rozmer je odôvodnený vo výpočte (8.2). Ďalej musí platiť že počet vlákien a exekučný čas musí byť neklesajúci. Takýmto výberom, vznikne množina výsledkov najefektívnejších behov z pohľadu veľkosti úloh vo vzťahu ku počtu vlákien, na základe čoho je potom možné odhadnúť exekučný čas danej úlohy na najefektívnejšom počte vlákien, ktorý je vypočítaný plánovačom. Tento výber a podrobnejší algoritmus je popísaný v časti 8.6.4.

   (b) Z týchto výsledkov sa vytvoria dva listy, ktoré reprezentujú body na ose X a ose Y.
      - List X - Obsahuje rozmery úloh na ose X, tých, čo najviac sadli rozmeru vo vzťahu k počtu vlákien.
      - List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.

   (c) Listy sa predajú aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

   (d) Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, funkcie vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v časti 8.6.6.

(e) Výsledok sa predá používateľovi, ale no môže nastať situácia, kde táto aproximácia je náročná na dáta a nemusí dostatok dát existovať, alebo dáta môžu vyjsť zlé. V tomto momente sa výsledky zahodia a započne aproximácia najbližšieho ideálneho počtu vlákien.

(f) Zo znalostnej databázy sa vyberie počet vlákien, ktorý je numericky najbližší vypočítanému ideálnemu počtu vlákien.

(g) Následne sa z databázy vyberú výsledky pre tento počet vlákien, ktoré majú rovnako zadaný klaster, frontu, vlastnosti akustického média úlohy (nie rozmer).

(h) Z týchto výsledkov sa vytvoria dva listy, ktoré reprezentujú body na ose X a ose Y.

- List X - Obsahuje rozmery úloh na ose X, tých, čo najviac sadli rozmeru vo vzťahu k počtu vlákien.
- List Y - Obsahuje dĺžky exekučných časov úloh z osy X na ose Y.

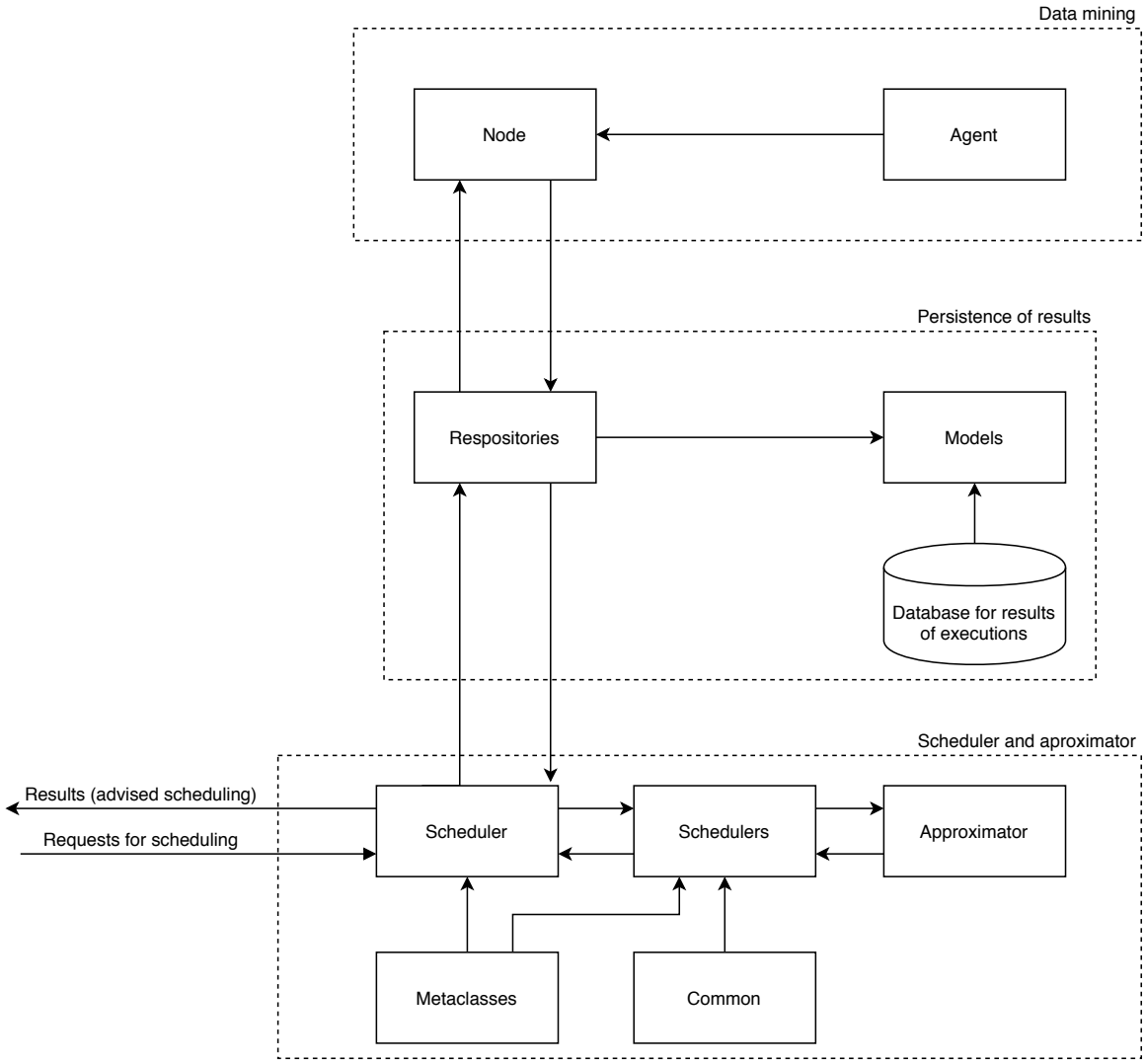
(i) Listy sa predajú aproximátorom, ktoré vykonajú aproximáciu (interpoláciu a regresiu), ktorou vzniknú aproximačné funkcie.

(j) Týmto aproximačným funkciám sa predá veľkosť požadovanej úlohy, funkcie vrátia exekučný čas pre daný počet vlákien a vykoná sa finálny výpočet exekučného času podľa výpočtu v časti 8.6.6.

(k) Výsledok sa vráti používateľovi.

7.3 Architektúra plánovača

Nasledujúci obrázok popisuje architektúru plánovača. Každá časť značí jeden modul (a jeho časti) v programovacom jazyku Python3. Vzťahy medzi časťami predstavujú vstupy a výstupy do modulov, napríklad návraty hodnôt z funkcii z daného modulu.



Obr. 7.4: Návrh architektúry výsledného plánovača

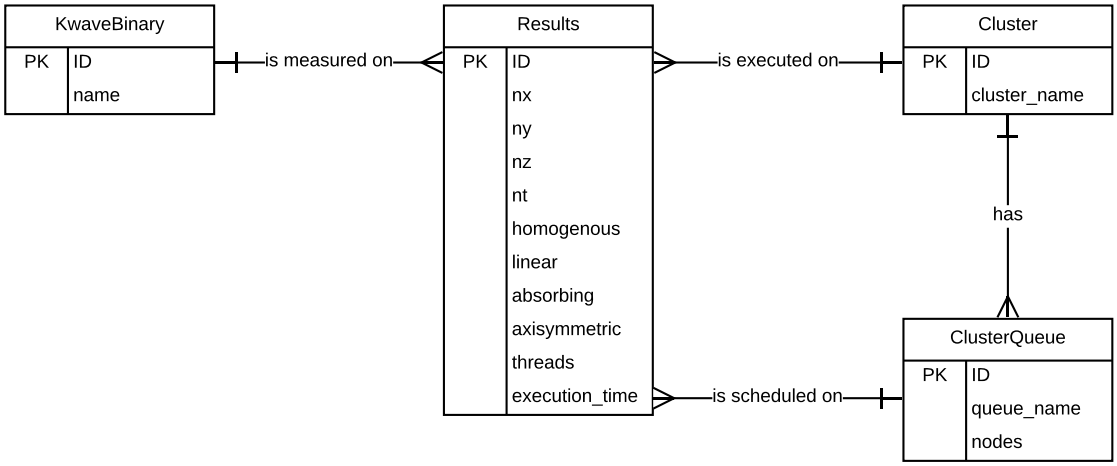
7.3.1 Perzistencia výsledkov

Tento logický celok uchováva a pracuje nad množinou získaných a uložených znalostných výsledkov, respektíve známych spustených simulácii úloh k-Wave. S touto časťou pracujú ostatné časti, pretože plánovač je založený na nazbieraných (empirických) dátach.

Databáza pre výsledky jednotlivých behov

Slúži pre perzistentné uchovanie výsledkov jednotlivých behov. Ako systém riadenia báze dát bola zvolená implementácia relačnej databázy PostgreSQL.

Ako prvé je nutné korektne navrhnúť správnu schému databázy. Toto bolo vykonané pomocou ER diagramov. Je nutné diagram navrhnúť čo najlepšie, aby boli zaznamenané všetky podstatné informácie pre správny priebeh aproximácie. Výsledný ER diagram vyzerá nasledovne.



Obr. 7.5: ER diagram pre perzistenciu výsledkov

Z tohto prístupu vyplýva, že databáza bude musieť niekde nepretržite pracovať alebo pracovať iba vtedy, keď pracuje plánovač. Presnejšia implementácia a použitie databázy je popísané v časti 8.1. Táto časť sa ďalej v práci bude nazývať ako znalostná databáza.

Modely (Models)

Modul pre modely slúži na mapovanie entít a vzťahov relačnej databázy na objekty objektovo-orientovaného paradigma. Táto technika sa presnejšie nazýva skratkou ORM (Object-Relational Mapping).

K premapovaným objektom databázy je potom možné jednoducho pristupovať, pomocou Python3 syntaxe, ako pri obyčajných objektoch. Z tohto vyplýva, že jedna trieda mapuje jednu tabuľku relačnej databázy. Nad inštanciou tejto triedy sú vykonávané dotazovacie operácie tak, ako ich poznáme v databázových systémoch z časti repozitárov.

Použitý framework pre ORM relačnej databázy na objekty nesie meno Peewee. Implementácia modelov je vysvetlená v časti 8.2.

Repozitár (Repository)

Repozitár modul sa snaží o implementáciu návrhového vzoru repozitár. Implementuje základné operácie nad databázou, respektíve implementuje rôzne druhy dotazov nad rôznymi tabuľkami databázy.

Opäť jeden repozitár (jedna trieda) je implementovaný nad jednou tabuľkou databázy. Metódy repozitára predstavujú dotazy. Repozitár zapúzdruje dátovú vrstvu celého plánovača a predstavuje rozhranie, vďaka ktorému môžu ostatné logické časti pracovať s dátami v databáze a už nemusia riešiť implementačné problémy tejto časti. Metódy a dotazy boli implementované na základe potrieb plánovača. Implementácia repozitárovej vrstvy je opísaná v sekcii 8.3.

7.3.2 Zber dát

Zber dát je ďalšia logická časť plánovača, ktorá sa zaoberá ako z názvu vyplýva, zberom dát. Pretože plánovač nemôže pracovať bez dát, je nutné dáta zozbierať a implementovať sadu, ktorá tento zber zabezpečí. Zber dát prebieha na umelých vstupných simuláciách, to znamená že simulácie (vstupné súbory) pre k-Wave sú používané iba na získavanie dôležitých znalostí do databázy a spúšťajú sa na zistenie priebehu istých N krokov. Ako sa zber dát skutočne uskutočnil je vysvetlené v časti 8.6.8. Popis implementácie častí, ktoré zber vykonávajú sa nachádza v časti 8.6.8.

Agent

Modul Agent slúži ako tvorca poľa pracovných skriptov, vďaka ktorým je vykonaný náber dát do znalostnej databázy. Agent je vytvorený pre superpočítače od IT4I, presnejšie Salomon a Anselm.

Agent má rôzne parametre pre nastavenie a generuje pole pracovného skriptu na základe toho, aký IT4I superpočítač bude vybraný. Pole pracovného skriptu, ktorý vytvorí Agent, žiada superpočítač o zadané zdroje, ktoré používateľ zadal pri generovaní skriptu. Ak plánovač (plánovač na klastri, PBS) predá zdroje a kontrolu, vygenerovaný skript nastaví superpočítač na zber dát, spúšťanie k-Wave binárnych súborov a následne začne pracovať nasledujúci modul Node. Agent môže byť spúšťaný na ľubovoľnom stroji, požiadanie práce a predanie poľa skriptov musí byť už na samostatnom superpočítači, presnejšie na prihlasovacom uzle čelnej časti klastru.

Uzol (Node)

K práci uzlového modulu prichádza už a iba na superpočítači. Spúšťa k-Wave binárne súbory s rôznymi parametrami na základe nastavenia v poly skriptov, ktoré vygeneroval Agent. Simulácie spúšťa s určitým počtom krokov, aby nemusela umelo vytvorená simulácia bežať celá, čo je vhodné pri veľkých simuláciách, ktoré by si žiadali veľa jadro-hodín. Po dokončení umelého behu simulácie na N krokoch, spracuje vstupný a výstupný súbor simulácie pre zber výsledkov a dôležitých hodnôt, následne uloží tieto hodnoty do znalostnej databázy.

7.3.3 Plánovanie a odhad parametrov simulácie

Srdce plánovača, modul odhaduje a plánuje parametre simulácie na základe požiadaviek používateľa, prípadne iného modulu, ktorý komunikuje s touto časťou. Modul zastrešuje celú prácu a plánovač. Bežný používateľ pracuje práve s týmto rozhraním.

Pomocný modul (Common)

Tento modul obsahuje základné pomocné triedy a metódy pre správny beh plánovača. Obsahuje.

- Matematická trieda (Maths) - Trieda vykonáva matematické funkcie a počty, ktoré sú použité pre odhad parametrov simulácie na základe dát.
- Formátovacia trieda (Printer) - Trieda formátuje čitateľný výstup plánovača na štandardný výstup.

- Nástroje (Utils) - Skript obsahuje implementované funkcie, ktoré dopomáhajú plánovaču ale nenašli presné zaradenie v triedach, napr. súčasné prechádzanie polí pre vymazávanie duplicít.

Aproximátor (Approximator)

Modul aproximátor obsahuje aproximátory, ktoré odhadujú neznáme hodnoty zo známych hodnôt na základe metód spomínaných v sekcii 6. Aproximátory sa delia na triedy na základe toho, akú metódu aproximácie vykonávajú. Všetky aproxmátory sú spolu zastrešované spoločným rozhraním. Každá trieda už obsahuje samostatne implementovanú metódu aproximácie. Plánovač pomocou rozhrania vytvára kostru aproximátora, z tejto kostry aproximátory dedia a implementujú definované funkcie rozhrania. Implementované aproximátory sú nasledujúce.

- Aproximátor pomocou kubického splinu
- Aproximátor pomocou Akima splinu [1]
- Aproximátor pomocou lineárneho spline
- Aproximátor pomocou čiastočného kubického Hermitovho interpolačného polynómu (ďalej ako PCHIP) [4]
- Aproximátor pomocou regresie polynómom
- Aproximátor pomocou spline regresie

Plánovač (Scheduler)

Modul slúži ako hlavné rozhranie pre používateľa alebo skript. Trieda obsahuje sadu metód, ktoré bude používať používateľ pre dotazovanie na odhad výsledných konfigurácii na základe uvedených parametrov k-Wave simulácie. Plánovač môže, ale nemusí vykonať zmeny úlohy na požiadavku pred jeho propagáciou. Plánovač rovnako vracia výsledky používateľovi. Plánovač ďalej na základe vybratej k-Wave verzie v parametroch, predá požiadavku správnemu plánovaču z nasledujúcej sekcie.

Plánovače (Schedulers)

Modul obsahuje triedy, ktoré implementujú konkrétne správanie plánovača na základe toho, akú binárnu k-Wave verziu používateľ zvolil. Metódy týchto tried, sú volané z hlavného modulu plánovača.

7.3.4 Metatriedy (Metaclasses)

Modul zastrešuje vstupno-výstupné triedy plánovača, trieda vstupu, výstupu, výnimky (exceptions) plánovača.

Kapitola 8

Implementácia plánovača

Kapitola detailnejšie rozoberá implementáciu plánovača a opisuje, ako plánovač rozhoduje o predikcii.

Celá práca je implementovaná v programovacom jazyku Python3. Python3 ponúka množstvo implementovaných a optimalizovaných knižníc pre zjednodušenie a zefektívnenie implementácie.

8.1 Databáza

Inicializačný databázový skript bol implementovaný v jazyku PostgreSQL. Databáza je umiestnená v kontajneri pomocou software Docker. Toto nesie výhodu pre jednoduché operovanie s databázou a ďalšie iné možnosti, ktoré ponúka Docker.

Konfiguračný súbor **Dockerfile** pre kontajner obsahuje nastavenia databázy ako meno, heslo a ďalšie nastavenia. Nad týmto súborom pracuje utilita **docker-compose**, ktorá obaľuje kontajner pre jednoduchú prácu a spúšťanie databázy. Súbor **docker-compose.yml** obsahuje dodatočné nastavenia ako mapovanie súborov, otvorenie portov do siete pre pripojenie na databázu, názov kontajnera... Používateľ alebo admin potom môže databázu jednoducho spustiť pomocou docker-compose utility. Všetky konfiguračné súbory sa nachádzajú v priečinku **database**.

8.2 Modelová vrstva

Pre každú tabuľku relačnej databázy zo sekcie 7.3.1 bola implementovaná trieda, ktorá mapuje túto tabuľku na triedu pre prácu s ňou v kóde plánovača. Triedy sú implementované pod modulom **models**.

Každá trieda dedí zo základnej mapovacej triedy **BaseModel**, ktorá implementuje pripojenie k databáze, presnejšie hosť, meno, heslo, používateľ databázy a databázový driver. Pri inicializácii tohto objektu triedy sú parametre databázového pripojenia otestované, či sú exportované ako premenné operačného systému na bežiacom systéme. Ak premenné neexistujú, použije sa predvolené nastavenie (localhost). Trieda kontroluje nasledujúce premenné systému.

- **IBP_DB_HOST** - Adresa lokácie databázy
- **IBP_PWD** - Heslo databázy
- **IBP_USER** - Meno databázového účtu

- **IBP_DB_NAME** - Meno databázy
- **IBP_DB_PORT** - Port databázy (typicky 5432)

Konkrétne implementované modely sú nasledujúce.

- **ClusterModel** - Mapuje tabuľku Cluster
- **ClusterQueueModel** - Mapuje tabuľku ClusterQueue
- **KBinaryModel** - Mapuje tabuľku KwaveBinary
- **ResultsModel** - Mapuje tabuľku Results

8.3 Repozitárová vrstva

Táto vrstva je implementovaná nad modelovou vrstvou. Nachádza sa pod modulom menom **repositories**. Nad každým modelom z časti 8.2, bola implementovaná samostatná trieda. Táto trieda reprezentuje jeden repozitár, ktorý obsahuje sadu používaných dotazov nad znalostnou databázou. Tieto dotazy používa ďalej implementácia plánovača. Vďaka tejto vrstve sa predchádza redundancii kódu. Boli implementované nasledujúce repozitáre.

- **ClusterRepo** - Repozitár nad modelom ClusterModel
- **ClusterQueueRepo** - Repozitár nad modelom ClusterQueueModel
- **KBinaryRepo** - Repozitár nad modelom KBinaryModel
- **ResultsRepo** - Repozitár nad modelom ResultsModel

8.4 Aproximátory

Aproximátory popísané v časti 7.3.3 sú implementované v module **approximator**, kde každý aproximátor je definovaný ako trieda. Všetky definované triedy dedia zo základnej triedy **BaseApproximator**. Táto trieda v konštruktore prijíma 2 listy ako argumenty, pričom listy reprezentujú body na ose X a Y. S listami aproximátor následne pracuje a vytvára z nich aproximačné funkcie. Trieda obsahuje implementované základné metódy.

- **std_x()** - Smerodajná odchýlka osy X.
- **std_y()** - Smerodajná odchýlka osy Y.

Aproximátory boli implementované nasledujúce.

- **AkimaSplineApproximator** - Vykonávajúci Akima spline
- **CubicSplineApproximator** - Vykonávajúci kubický spline
- **LinearSplineApproximator** - Vykonávajúci lineárny spline
- **PchipApproximator** - Vykonávajúci PCHIP interpoláciu
- **PolyfitApproximator** - Vykonávajúci regresiu polynómom

- **SplineRegressionApproximator** - Vykonávajúci spline regresiu

Tieto triedy obsahujú implementáciu aproximačných metód z kapitoly 6. Vypočítanie aproximácie sa zavolá "magic" metódou **___call___**, resp. zavolaním objektu triedy aproximátora ako funkciu. Kde funkcia prijíma jeden argument, kde tento argument je hodnota na ose X. Funkcia navracia aproximovanú hodnotu na ose Y.

8.5 Analýza vstupnej úlohy

Vstupná úloha sa musí analyzovať na základe jej vlastností, aby plánovač vedel ako s danou úlohou zaobchádzať a aproximovať dvojicu konfigurácií.

Pre analýzu vhodného alebo nevhodné rozmeru simulácie (5.2.1) existuje implementovaná metóda triedy **Math** v module **common** menom **has_good_shape()**, ktorá vypočíta prvočíselný rozklad rozmeru úlohy, a na základe vybranej k-Wave verzie určí Boolean hodnotu, či úloha má vhodný rozmer a túto hodnotu vráti.

8.6 Plánovač

Plánovač môže byť spúšťaný v dvoch módoch, presnejšie buď importovaním hlavného modulu, alebo spustením hlavného modulu cez príkazový riadok.

Pri použití plánovača cez príkazový riadok sa plánovač spúšťa pomocou hlavného skriptu **scheduler.py**, príkazom **python3 scheduler.py <argumenty>**. Kde tieto argumenty sú.

- **-x*** - Veľkosť dimenzie X úlohy
- **-y*** - Veľkosť dimenzie Y úlohy
- **-z*** - Veľkosť dimenzie Z úlohy
- **-n*** - Počet krokov úlohy
- **-t** - Počet vlákien na ktorých úloha bude spustená
- **-k*** - Názov binárnej verzie k-Wave ("mpi", "omp", "cuda")
- **-c*** - Názov klastru na IT4I ("salomon", "anselm")
- **-q*** - Názov fronty na klastri ("qprod", "qnvidia")
- **-h** - Homogenita úlohy
- **-l** - Linearita úlohy
- **-a** - Absorbcia úlohy
- **-j** - Výpis výsledkov v formáte JSON
- **-?** - Výpis nápovedy

(Parametre označené hviezdou sú povinné.)

Tieto parametre sú spracované plánovačom a poukladané do objektu reprezentujúceho vstup plánovača triedou **SchedulerInput** a následne je plánovač spustený. Tento proces spracovania argumentov a spustenia vykonáva modul **bootstrap**.

Hlavný vstupný modul a základ plánovača je **scheduler**. Tento modul obsahuje hlavnú triedu menom **Scheduler**. Inicializovaním tejto triedy používateľ získava rozhranie pre prácu s plánovačom. Na požiadanie o predikciu parametrov slúži metóda **request_schedule()**, ktorej na vstup musí byť predaný objekt triedy **SchedulerInput** z modulu **metaclasses**.

Po zavolaní tejto metódy sa započína aproximácia na základe predaného vstupného objektu. Spočiatku sa rozhodne, pre akú k-Wave binárnu verziu sa započína odhad, pretože rôzne implementácie k-Wave majú rôznu heuristiku pre odhad a tým pádom aj iný workflow.

Tieto heuristiky a rôzne implementácie patria pod modul s názvom **schedulers**, ktorý obsahuje 3 triedy, ktoré vykonávajú výslednú aproximáciu na základe zvolenej binárnej verzie. Základná trieda má názov **BaseScheduler**, z ktorej ostatné plánovače dedia, definuje kostru a zopár základných spoločných metód pre plánovače. Tieto špecifické plánovače boli implementované vo vzťahu ku binárnym verziám k-Wave.

- **MpiScheduler** - Trieda vykonávajúca aproximáciu pre MPI verziu k-Wave
- **OmpScheduler** - Trieda vykonávajúca aproximáciu pre OMP verziu k-Wave
- **CudaScheduler** - Trieda vykonávajúca aproximáciu pre CUDA verziu k-Wave

Trieda **BaseScheduler** obsahuje deklaráciu metódy **schedule()**, ktorú vymenované triedy implementujú.

Z toho vyplýva, že základná spomínaná trieda **Scheduler** vytvára a implementuje rozhranie pre používateľa, ktorému predá vstupné dáta a na základe vybranej k-Wave binárnej verzie vyberie, ktorú triedu z vyššie vymenovaných inicializovať, a následne zavolať metódu **schedule()**. Implementácia a popis tried z modulu **Schedulers** je popísaná nasledovne.

Zo zastrešovacej triedy a hlavnej metódy **request_schedule()** sa vracajú objekty **SchedulerOutput** v liste. Trieda **SchedulerOutput** implementuje triedu, ktorej objekt reprezentuje výstup plánovača.

Následne, ak predikcia bola správna a používateľ chce vložiť nový známy beh do znalostnej databázy, existuje implementovaná metóda triedy **Scheduler** menom **confirm_result()**, ktorá ako vstup prijíma objekt výstupu plánovača **SchedulerOutput**. Následne vyjme predané informácie a predá ich repozitárovej vrstve, kde táto repozitárová vrstva vytvorí nový objekt modelu **ResultsModel** a výsledok uloží do znalostnej databázy tak, aby plánovač mohol brať do úvahy výsledok v ďalších predikciách.

8.6.1 Vstup plánovača (SchedulerInput)

Táto trieda **SchedulerInput** popisuje parametre simulácie, pre ktorú má byť predikcia vykonaná. Skladá sa z atribútov, ktoré reprezentujú parametre vstupnej simulácie spomínané v kapitole 5.2 a z parametrov, ktoré sa predávajú pri spúšťaní cez príkazový riadok 8.6. Dokopy má vstup nasledujúce atribúty.

- **nx** - Rozmer 1. dimenzie aproximovanej úlohy
- **ny** - Rozmer 2. dimenzie aproximovanej úlohy

- **nz** - Rozmer 3. dimenzie aproximovanej úlohy
- **nt** - Počet časových krokov aproximovanej úlohy
- **homogenous** - Homogenita alebo heterogenita aproximovanej úlohy
- **linear** - Linearita alebo nelinearita aproximovanej úlohy
- **absorbing** - Absorbcia alebo neabsorbcia aproximovanej úlohy
- **kBinaryId** - Meno binárnej verzie k-Wave
- **requestedClusterName** - Meno klastru na IT4I
- **requestedQueueName** - Meno fronty na klastri
- **threads** - Počet vlákien na ktorých má byť aproximovaná úloha spustená
- **gridpoints** - Veľkosť úlohy

Počet vlákien musí byť vyplnené číslo pre prípad, ak používateľ chce odhad pre presne daný počet vlákien, pri nedefinovanom počte plánovač rozhoduje o počte vlákien. Trieda kontroluje validáciu parametrov pomocou getterov a setterov, a to presnejšie dotazom nad databázou, či existuje záznam uvedenej k-Wave binárnej verzie v databáze, uvedený názov klastru a uvedený názov fronty.

8.6.2 Výstup plánovača (SchedulerOutput)

Trieda **SchedulerOutput** dedí z triedy **SchedulerInput**, vďaka čomu môže poskytnúť rovnakú funkcionalitu ako vstup. Ku tomu ale ešte pridáva zopár atribútov, ktoré reprezentujú výstup aproximácie. Tieto atribúty sú.

- **execTimePredicted** - Odhadovaný exekučný čas simulácie (bez časovej poistky, riziko použitia)
- **execTimePredictedSafety** - Finálny odhadovaný exekučný čas simulácie
- **methodUsed** - Názov použitej aproximačnej metódy
- **calculationPrice** - Cena v jadro-hodinách odhadovaného exekučného času (bez časovej poistky)
- **calculationPriceSave** - Cena v jadro-hodinách finálneho odhadovaného exekučného času

Ak plánovač rozhodoval o počte vlákien, atribút **threads** bude mať pri výstupe vyplnenú celočíselnú hodnotu počtu vlákien.

8.6.3 OpenMP k-Wave (OmpScheduler)

Táto časť sa snaží slovne popísať pracovný tok časti plánovača pre binárnu verziu k-Wave OMP. Slovný popis obsahuje taktiež názvy funkcii, ktoré sa priebežne volajú pre vykonanie aproximácie.

Plánovač sa rozhoduje na začiatku ako sa zachová. Toto rozhodnutie závisí na tom, či používateľ zadal počet vlákien alebo nezadal.

Ak výsledná predikcia je nevhodná (napr. záporný exekučný čas), plánovač tieto výsledky zahadzuje a nevracia ich používateľovi. Toto platí pre obe možnosti predikcie (s zadaným počtom vlákien a bez).

Zadaný počet vlákien

Pri zadanom počte vlákien sa dotaz na repozitár upravuje a špecifikuje sa daný počet zadaných vlákien. To znamená, že zo znalostnej databázy výsledkov sa použijú iba známe výsledky pre daný klaster, binárnu verziu k-Wave, frontu, presný počet vlákien a rovnaké vlastnosti simulácie, presnejšie vlastnosti jej média. Tieto výsledky sa vyberú pomocou repozitárovej metódy **get_results_threads()**, repozitára **ResultsRepo**. Znalostná databáza ale pre daný počet vlákien nemusí poznať empirické výsledky, alebo počet empirických výsledkov nemusí byť tak veľký, aby interpolácia alebo regresia mohla byť vykonaná, v tom prípade plánovač ukončuje svoju činnosť. Minimálny počet spomínaných výsledkov (výsledkov spomínaného dotazu) pre túto aproximáciu je 3.

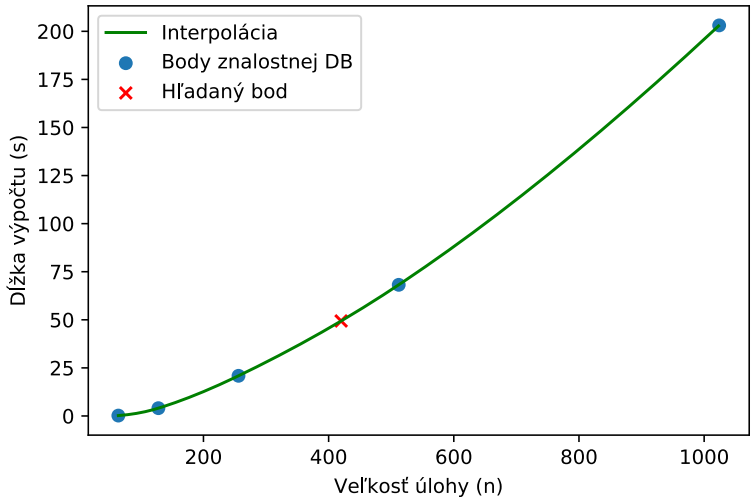
Ak výsledky existujú, začne sa vyšetrovať rozmer zadanej úlohy popísaný v časti 8.5 pomocou **has_good_shape()**. Ako prvé sa začnú pripravovať dáta pomocou metódy triedy **BaseScheduler** menom **prepare_data()**. Táto metóda pretransformuje ORM databázové objekty (známe výsledky), na listy dát. Listy obsahujúce veľkosti úloh, ich exekučné časy a počty vlákien (tento list sa ale zahadzuje). Tieto listy obsahujú teda konkrétne hodnoty známych výsledkov, ktoré plánovač už môže považovať za konkrétne spoločné body na osách, podla toho ako listy predá aproximámtorom. Listy sú hodnotami logicky spoločne zarovnané na rovnakú pozíciu, čiže index číslo 0 vo všetkých listoch reprezentuje výsledok, ktorý má určitú veľkosť úlohy, exekučný čas a počet vlákien na ktorom bol spustený.

Na začiatku sú v listoch aj výsledky s duplicitnými veľkosťami úlohy, kde tieto listy s duplicitami sa použijú pre inicializáciu aproximátorov, ktoré využívajú regresiu, pretože regresia dokáže pracovať aj s duplicitnými známymi bodmi na ose X. Následne sú tieto listy predané metóde triedy **BaseScheduler** menom **init_regressions()**, kde metóda inicializuje objekty triedy z modulu **approximators**, ktoré pre odhad používajú regresiu. Tieto aproximačné objekty si **OmpScheduler** ukladá do listu.

Následne sú podobne inicializované aproximátory, ktoré používajú interpoláciu. Na začiatku je ale nutné z listov odstrániť duplicity, pretože interpolácia nedokáže pracovať s dvoma rovnakými hodnotami, toto prefiltrovanie zariadi funkcia modulu **utils**, **make_average_data()**. Táto funkcia spočíta priemer hodnôt Y, pre ktoré sa opakujú hodnoty na ose X. To znamená že ak v znalostnej databáze sú známe behy, ktorých veľkosť úlohy sa opakuje, jej všetky zaznamenané exekučné časy sa spriemerujú a berie sa tento priemer ako jediný bod pre danú úlohu na ose Y. Pre inicializáciu je použitá metóda **init_interpoľators()** a objekty aproximátorov sú opäť uložené do listu aproximátorov, resp. pridané za predchádzajúce aproximátory.

Po inicializácii a úprave dát sa prechádza listom aproximátorových objektov a volá sa aproximácia pomocou **___call___** metódy, ktorej vstup je rozmer aproximovanej úlohy. Respektíve vstupom je bod na ose X. Táto metóda navráti iba hodnotu odhadovaného

exekučného času z aproximačnej funkcie, následne je nutné vykonať finálnu kalkuláciu exekučného času. Táto kalkulácia je odôvodnená v časti 8.6.6 a vykoná sa zavolaním metódy **calculate_safety()** z triedy **BaseScheduler** modulu **schedulers**. Po každej aproximácii je vytvorený objekt výstupu plánovača, resp. **SchedulerOutput** pomocou triednej metódy **create_output()** a navracia sa list týchto výstupných objektov, ak nepríde na filtráciu zlého výstupu (8.6.7).



Obr. 8.1: Interpolácia známych výsledkov pre zadaný počet vlákien (24 vlákien) a vyžiadanú úlohu $420 \times 420 \times 420$

Nezadaný počet vlákien

Pre nezadaný počet vlákien je odhad zložitejší, pretože je nutné prepočítať cenu výpočtu vzhľadom ku každému známemu počtu vlákien (záznamov) v znalostnej databázy.

Ako prvé sa zo znalostnej databázy vyberie pomocou repozitárovej metódy **get_threads_distrinct()** list obsahujúci unikátne počty vlákien v znalostnej databáze, pre ktoré existujú v databáze viac ako 3 výsledky a majú rovnakú k-Wave binárnu verziu, rovnaký IT4I klaster, rovnakú frontu na klastri a rovnaké vlastnosti simulácie. Vzhľadom na to, že OMP verzia k-Wave dokáže pracovať iba na jednom procesore, resp. maximálny počet vlákien je daný počtom jadier procesora, je tento list vyfiltrovaný na základe uvedeného maximálneho počtu vlákien. (Salomon 24 a Anselm 16 vlákien)

Následne pre každý počet vlákien z listu, sú vybrané známe výsledky pomocou repozitárovej metódy **get_results_threads()**. Následne sú známe výsledky podobne spracované pomocou metódy **prepare_data()**, ktorá premieňa objekty databázy (známe výsledky) na konkrétne hodnoty, ktoré sa dajú považovať už ako konkrétne hodnoty výsledkov, spomínaných v predchádzajúcej časti (zadaný počet vlákien). Čiže pre každý počet vlákien zo znalostnej databázy sa dáta vyžiadajú a spracujú do listov.

Každý aproximátor je následne inicializovaný prázdnymi listami. Následne pre každý list známych hodnôt, kde hodnoty majú rovnaký počet vlákien, sa predávajú aproximátorom pomocou metódy **update_vectors()** a začne výpočet aproximácie pre daný vstup na ose X - rozmer úlohy. Pretože takto sa vypočíta aproximácia exekučného času zadanej úlohy

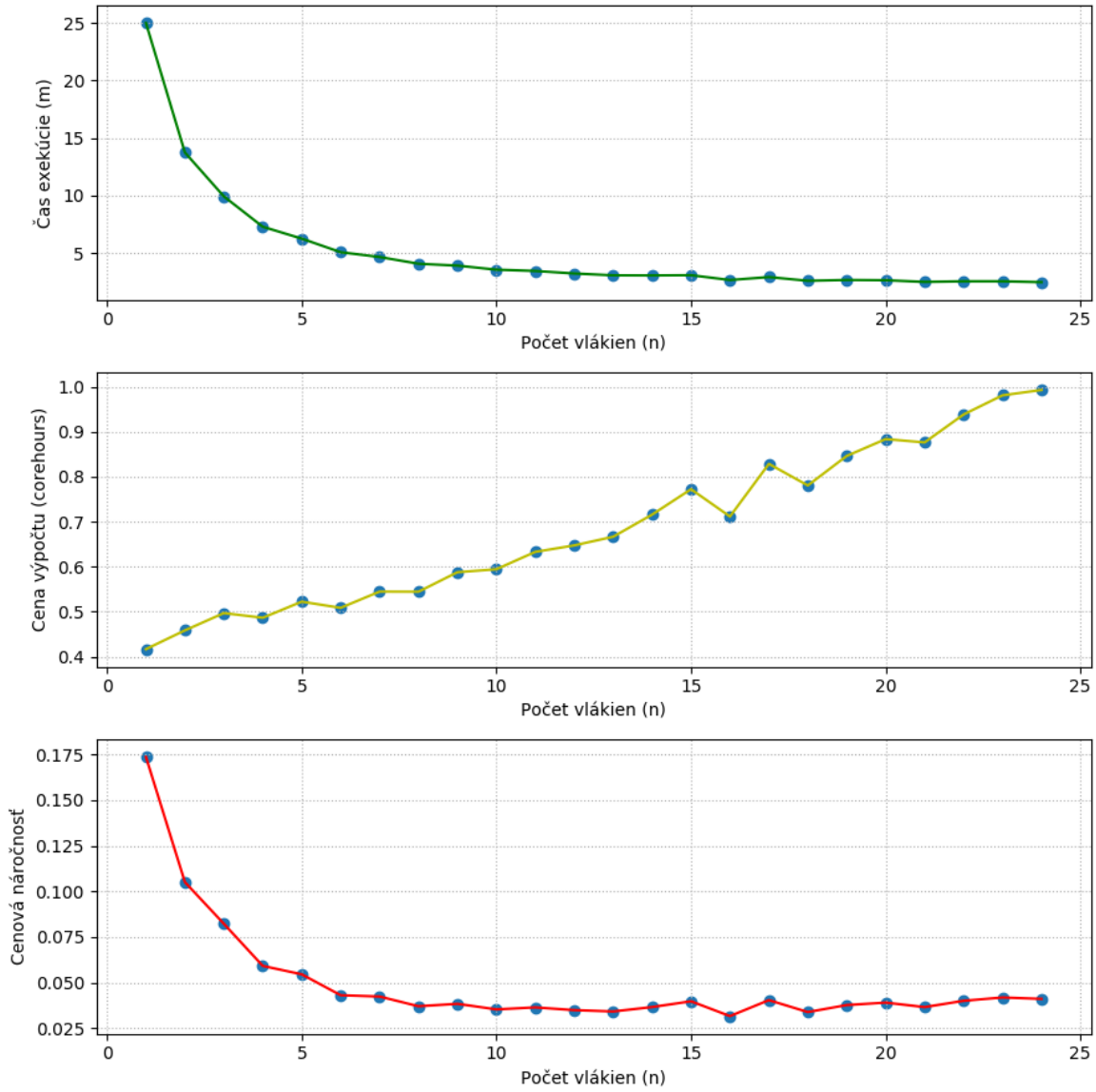
pre každý známy počet vlákien, je nutné rozhodnúť, ktorý exekučný čas a počet vlákien bude zvolený za najlepší výsledok.

Určenie každého výsledku prebieha výpočtom cenovej náročnosti. Náročnosť je vypočítaná ako.

$$E = P \times H \tag{8.1}$$

Kde:
E - Cenová náročnosť
P - Cena výpočtu (Jadro-hodiny)
H - Dĺžka behu

V tomto vzťahu platí, že čím menšia hodnota výsledku, tým efektívnejší výpočet je. Čiže pre každý výsledok aproximácie je vypočítaná táto cenová náročnosť a za výsledok, resp. výsledný počet jadier a čas exekúcie, sa považuje ten, ktorý má najlepšiu (hodnotou najmenšiu) cenovú náročnosť.



Obr. 8.2: Porovnanie doby exekúcie úlohy $256 \times 256 \times 256$, výpočtu ceny a cenovej náročnosti

Aproximátorový objekt navrátil iba hodnotu z aproximačnej funkcie, následne je ale nutné vykonať finálnu kalkuláciu exekučného času. Táto kalkulácia je odôvodnená v časti 8.6.6 a vykoná sa zavolaním funkcie **calculate_safety()**. Tento finálny výsledok je opäť spracovaný a pretransformovaný do objektu výstupu plánovača **SchedulerOutput** a navrátený. Výber výsledných najefektívnejších výsledkov prebieha samostatne pre každý aproximátor samostatne, napr. ak je 8 aproximátorov, výsledok bude 8 najlepších aproximácii vzhľadom na efektivitu.

8.6.4 MPI k-Wave (MpiScheduler)

Tento plánovač je najzložitejší a implementuje najnáročnejšie operácie. Pretože MPI verzia k-Wave môže pracovať na všetkých procesoroch celého klastru predikcia je komplikovanejšia.

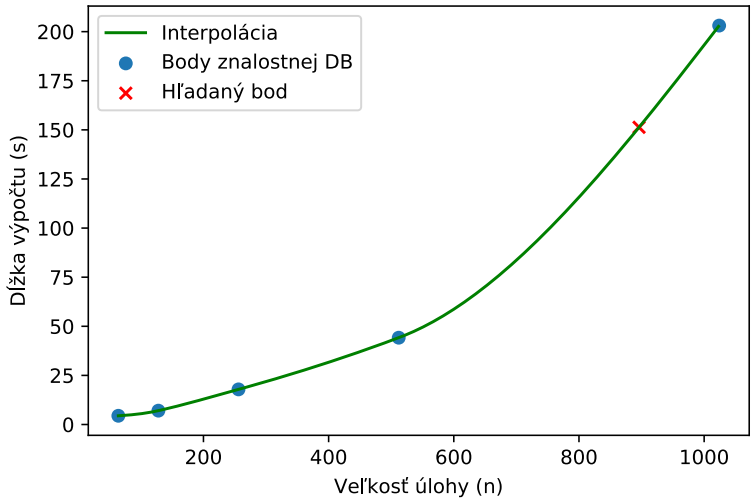
Postup odhadu sa rozdeľuje na základe zadaného (vyžiadaného) alebo nezadaného počtu vlákien.

Zadaný počet vlákien

Ak používateľ zadal počet vlákien, plánovač sa snaží odhadnúť parametre presne pre tento počet. Ako prvé plánovač skontroluje z databázy, či existujú známe výsledky v znalostnej databázy presne pre daný počet vlákien a daný klaster. Toto je vykonané pomocou metódy **get_results_threads()**, repozitára **ResultsRepo**.

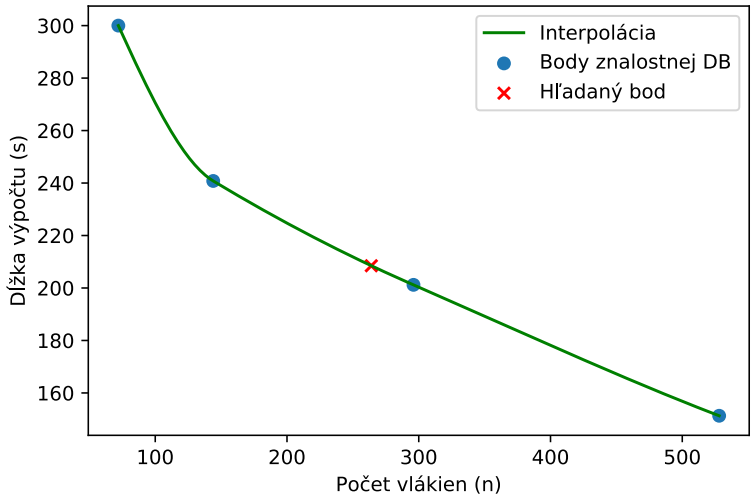
Ak existujú známe výsledky a ich počet je väčší ako 3, započne sa aproximácia na základe týchto výsledkov. Podobne, ako predchádzajúce plánovače, prebehne preparácia dát pomocou metódy **prepare_data()**. Navrátené listy so známymi hodnotami, resp. bodmi, sú predané aproximátorom, ktoré používajú regresiu metódou **init_regressions()**. Potom sú dáta prefiltrované podľa unikátnosti rozmeru úlohy funkciou **make_average_data()** z modulu **utils** a predané aproximátorom, ktoré používajú interpoláciu metódou **init_interpolators()**. Aproximátorové objekty sú podobne uložené do listu. Nasledovne sa pre každý aproximátor vyžiada aproximácia na základe vstupu, presnejšie zadaného rozmeru úlohy, pomocou "magic" **___call___** metódy aproximátoru. Tieto hodnoty z aproximátorov, aproximované exekučné časy sú opäť prepočítané s časovou poistkou zo sekcie 8.6.6, pomocou metódy **calculate_safety()**. Následne sú finálne výsledky podobne uložené do výstupných objektov plánovača **SchedulerOutput** a navrátené používateľovi.

Ak ale neexistujú známe výsledky pre zadaný počet vlákien, je nutné aproximovať iným spôsobom. Spôsob ako aproximovať dĺžku behu pre tento počet vlákien, je použitím dvojitej interpolácie. Dvojitá interpolácia, ako prvý krok nájde každý známy počet vlákien, kde počet výsledkov je väčší ako 3 pomocou repozitárovej metódy **get_threads_distinct()**. Následne vytvorí sady výsledkov, kde každá sada predstavuje výsledky ktoré majú spoločné počty vlákien, pomocou repozitárovej metódy **get_results_threads()** a metódy **prepare_data()**. Pre každú sadu výsledkov daného počtu vlákien sa vytvorí interpolácia pomocou aproximátorov, ktoré využívajú interpoláciu metódou **init_interpoľators()**. Následne sa vypočíta interpolácia exekučného času rozmeru zadanej úlohy pomocou "magic" metódy **___call___**. Týmto sa získa aproximácia dĺžky času exekúcie úlohy, na známych počtoch vlákien v znalostnej databáze, ale nie pre presný (zadaný) počet vlákien od používateľa. Tieto časy vo vzťahu ku počtom vlákien sa uložia do listov. Týmto sa vykonala prvá interpolácia z dvoch.



Obr. 8.3: Interpolovanie jednej sady z N sád výsledkov vo vzťahu ku počtu vlákien pre vytvorenie vstupov pre druhú interpoláciu, 528 vlákien, žiadaná úloha $896 \times 896 \times 896$ a 264 žiadaných vlákien

Ďalší krok je vytvoriť z týchto dvoch listov, finálnu interpoľačnú funkciu vďaka ktorej je možné získať dĺžku exekúcie na neznámom počte vlákien. Pomocou už alokovaných a inicializovaných aproximátorov sa vykoná druhá interpolácia. Alokovaným aproximátorom sú predané spomínané nové dva listy pomocou metódy aproximátorov **update_vectors()**. Interpolovaním týchto hodnôt vzniká krivka vytvárajúca závislosť medzi počtom vlákien (osa X) a dĺžkou výpočtu pevnej (zadanej) úlohy (osa Y). Následne sa teda interpoluje už presná, resp. žiadaná veľkosť vlákien. Opäť sa pre každý aproximátor zavolá "magic" metóda **___call___** a výsledok sa prepočítava s časovou poistkou z 8.6.6. Finálny výsledok sa usporiada do výstupného objektu plánovača **SchedulerOutput**. Týmto aproximácia pre danú situáciu končí, a plánovač navracia list výstupov plánovača.



Obr. 8.4: Druhá interpolácia, interpolovanie výsledkov pre žiadaný rozmer úlohy, kde už odhad je pre presne žiadaný počet vlákien (264 vlákien)

Nezadaný počet vlákien

Pretože MPI verzia podporuje rôzny počet procesorov, je viac pravdepodobné že výsledky pre zadaný počet vlákien nebudú v znalostnej databáze.

Preto je nutné vypočítať najoptimálnejší počet vlákien pre danú úlohu. Pretože simulácia je v trojrozmernom priestore, MPI implementácia k-Wave podáva najlepší výkon v situácii, kde doska o veľkosti $N_X \times N_Y \times 1$ pripadá na jedno fyzické jadro a tým pádom na jedno vlákno, kde počet vlákien je rovnaký s počtom fyzických jadier procesora. Tým pádom sa ideálny počet vlákien pre úlohu odvodí ako.

$$T_I = \begin{cases} X & N_Z \mod X = 0 \implies N_Y \mod X = 0, X = 1, .., \max(N_Z, N_Y) \\ Y & \max(N_Z \mod Y), Y = 1, .., \max(N_Z, N_Y) \end{cases} \tag{8.2}$$

Kde:
T_I - Ideálny počet vlákien pre danú úlohu
N_Z - Veľkosť 3. dimenzie úlohy
N_Y - Veľkosť 2. dimenzie úlohy

Vypočítaný počet ideálnych vlákien, je dobrý začiatok pre ďalšiu aproximáciu. Ako prvé sa plánovač snaží vyhľadať, či v znalostnej databázy existujú informácie (výsledky) vyhovujúce vypočítanému ideálnemu počtu vlákien.

Ak výsledky existujú a je ich viac ako 3, podobne ako v predchádzajúcich situáciách sa prichystajú aproximátorové objekty, vykoná sa aproximácia, výpočet dĺžky exekúcie (8.6.6) a výsledok sa navráti používateľovi. Táto situácia je najmenej pravdepodobná.

Ak ale výsledky neexistujú, započne sa ďalší spôsob aproximácie. Tento spôsob sa zameriava na najefektívnejšie behy, resp. aproximácia sa vykonáva na základe najefektívnejších behov. Na začiatku táto metóda načíta opäť pomocou repozitárovej metódy **get_threads_distinct()** list počtov vlákien, pre ktoré v znalostnej databázy existujú výsledky. Následne pre každý počet vlákien, kde počet vlákien je usporiadaný vzostupne,

načíta list výsledkov, ktoré boli spustené na tomto počte vlákien pomocou metódy **get_results_threads()**. Následne sa snaží z tohto množstva výsledkov, nájsť vhodný výsledok, ktorého rozmer úlohy sa blíži najefektívnejšiemu využitiu počtu vlákien. To znamená, že daná úloha výsledku najefektívnejšie využila počet jadier na ktorom bola spustená. Napríklad úloha $24 \times 24 \times 24$ zo znalostnej databázy bola spustená na 24 vláknach na 24 jadrovom procesore (Salomon), táto úloha najlepšie využíva výkon výpočtového uzlu, pretože doska $1 \times 1 \times 1$ beží na 1 vlákne a toto vlákno má pre seba 1 fyzické jadro. Takýto výsledok je pre najefektívnejšiu aproximáciu najlepší. Z výsledku, ktorý sa teda blíži najbližšie takémuto ideálnemu výsledku je uložená do listov, veľkosť úlohy (body na ose X) a dĺžka exekučného času (body na ose Y). Treba ale pripomenúť, že hodnoty listov musia byť stúpajúce, pretože so stúpajúcim počtom vlákien je očakávaný väčší rozmer úlohy, a zároveň s väčším rozmerom úlohy je očakávaný väčší exekučný čas. Ak by táto podmienka nebola použitá, mohla by nastať situácia, kde by pre veľký počet vlákien bola v databáze iba veľmi malá úloha a exekučný čas by bol veľmi malý, tým pádom by hodnoty listov predstavovali nemonotónnu funkciu a predikcia by bola zlá. Algoritmicky potom táto podmienka vyzerá nasledovne.

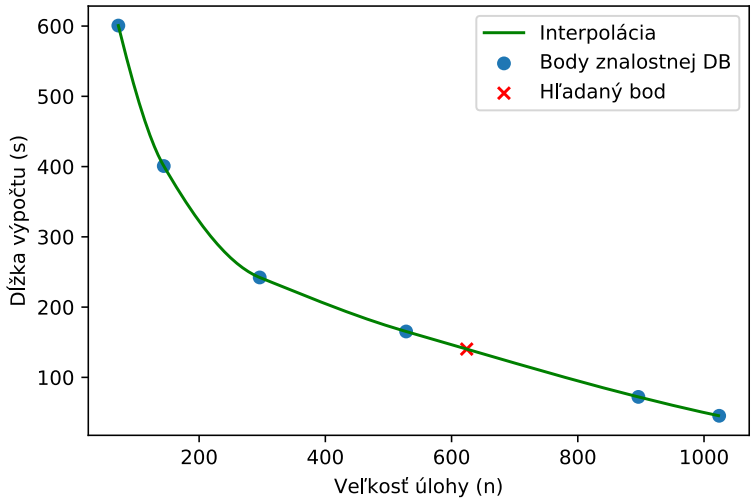
```
previousTask = -1
previousTime = -1
uniqueThreads = get_threads_distinct()

for threads in uniqueThreads:
    ...
    results = get_results_threads(threads)

    for result in results:
        gridPoints = result.nx * result.ny * result.nz
        cubicPoints = cbrt(gridPoints)
        ...
        if result.gridPoints > previousTask and
        result.execution_time >= previousTime and
        cubicPoints <= threads:
            tasksVector.append(gridPoints)
            execTimeVector.append(result.execution_time)
            previousTask = gridPoints
            previousTime = result.execution_time
```

Výpis 8.1: Algoritmus vyberania vhodných známych výsledkov pre najefektívnejšiu aproximáciu.

Týmto vzniknú dva listy, ktoré popisujú vzťah veľkosti úloh (body na ose X) k dobe ich výpočtu (body na ose Y), kde tieto úlohy bežia na najväčšom a najefektívnejšom možnom počte vlákien. Tieto listy sú následne predané aproximátorom, do ktorých sa vloží veľkosť aproximovanej (zadanej) úlohy, a pre každý aproximátor sa aproximuje výsledný exekučný čas, následne je ku aproximovanému exekučnému času pripočítaná časová poistka 8.6.6. Opäť sú počty vlákien a finálne exekučné časy zabalené do výsledných objektov a navrátené používateľovi.



Obr. 8.5: Výsledná interpolácia najviac efektívnych výsledkov vzhľadom na počet vlákien a veľkosti úlohy

Po tejto aproximácii ale taktiež môže nastať situácia, kedy výsledky môžu byť nesprávne a musia byť odfiltrované. Ak táto situácia nastane a výsledných objektov je 0, respektíve by plánovač nepredikoval nič, plánovač sa snaží nájsť aproximáciu pre najbližší známy počet jadier ku ideálnemu počtu jadier. Opäť pomocou metódy **get_threads_distinct()** vyhľadá známe počty vlákien v znalostnej databázy. Z tohto listu vlákien sa plánovač najskôr snaží nájsť najbližší vyšší alebo rovný počet známych vlákien k ideálnemu počtu vlákien. Ak sa toto vyhľadávanie nepodarí, plánovač sa snaží nájsť najbližší menší alebo rovný počet známych vlákien. Následne pre tento presný počet použije metódu **get_results_threads()** pre vytiahnutie dát z databázy a metódu **prepare_data()** pre ich prípravu na listy (body na osách), kde prvý list predstavuje známe rozmery úloh (body na ose X) a druhý list predstavuje exekučné časy týchto úloh z prvého listu (body na ose Y). Následne plánovač predá listy aproximátorom a pripraví ich pre výpočet. Podobne ako v iných plánovačoch, touto aproximáciou dostávame vzťah medzi rozmerom úlohy a dobou výpočtu na počte vlákien, ktorý je najbližší ideálnemu počtu vlákien. Opäť predaním aproximovaného (zadaného) rozmeru úlohy aproximátoru sa dostáva aproximovaný exekučný čas, ktorý musí opäť obsahovať časovú poistku podľa časti 8.6.6. Počet vlákien a exekučný čas sa ukladá do výsledného objektu a navracia sa používateľovi.

Ako je možné vidieť, tento plánovač pre MPI sa snaží vždy nájsť dvojicu počtu vlákien a exekučného času, aj keď pre ideálny počet vlákien exekučný čas nedokáže nájsť.

8.6.5 CUDA k-Wave (CudaScheduler)

OMP a CUDA verzie sú veľmi implementáciou podobné v rámci využitia zdrojov. CUDA verzia dokáže pracovať maximálne na jednej grafickej karte. Výpočetná sila procesora sa používa iba pri načítavaní dát a pri použití základných funkcii, tým pádom škálovanie úlohy na veľké počty vlákien a jadier je nepotrebné.

Na základe týchto faktov je možné OMP a CUDA verzie rovnako aproximovať a používať rovnaké metodiky. Tento plánovač funguje identicky s OMP plánovačom.

8.6.6 Finálny výpočet exekučného času

Pri odhadovaní exekučného času, je nutné aby plánovač vedel, koľko časových krokov obsahuje daná simulácia. Preto je argument `-n` nutný pri spúšťaní plánovača. Počet simulačných krokov výrazne ovplyvňuje, aký dlhý bude exekučný čas simulácie 5.2.

Preto aproximátory spomínané v implementačnej časti a listy s dátami predávané týmto aproximátorom vždy obsahujú časy iba jedného kroku, kde tento počet je zároveň minimum krokov simulácie. Na základe exekučného času jedného kroku je potom možné odvodiť exekučný čas pretože každý krok simulácie obsahuje rovnaký počet rovníc, výpočtov... Potom exekučný čas pre ľubovoľný počet krokov je možné odvodiť jednoduchým násobením.

Výpočet finálneho aproximovaného exekučného času potom vyzerá nasledovne.

$$T = A(N_S) \times N_T + S \tag{8.3}$$

Kde:
T - Výsledný aproximovaný exekučný čas
$A(x)$ - Aproximátor, ktorý obsahuje aproximačnú funkciu (interpoláciu alebo regresiu) pre jeden časový krok simulácie
N_S - Rozmer vstupnej úlohy
N_T - Počet časových krokov simulácie
S - Časová poistka
Je možné si všimnúť, že za výsledok aproximácie sa neudáva čisto iba čas aproximačnej funkcie, ale pričíta sa k výsledku aj istá časová poistka.

Časová poistka slúži pre vytvorenie časovej rezervy pre možnú nepresnosť aproximácie. Nepresnosť aproximácie môže byť zapríčinená rôznymi faktormi.

- Nevhodný rozmer úlohy (prvočíselný rozklad úlohy)
- Zátaž klastru
- Nepresný výsledok aproximačnej funkcie (interpolácie alebo regresie)
- Oscilácia aproximačnej funkcie

Časová rezerva sa snaží pripočítaním času tieto nepresné odhady spresniť. Táto časová rezerva je vypočítaná pre každý výsledok aproximátoru nasledovne.

$$R = \frac{A(N_S) \times N_T}{\sigma_A \times N_T} \tag{8.4}$$

Kde:
R - Pomer predikovaného exekučného času aproximátora (jeho funkcie)
A - Aproximačná funkcia, ktorá berie na vstup rozmer úlohy
N_S - Rozmer úlohy
σ_A - Smerodajná odchýlka aproximačnej funkcie - smerodajná odchýlka známych vybraných výsledkov
N_T - Počet časových krokov simulácie
Tento pomer R zobrazuje pomer odhadovaného exekučného času aproximátora ku smerodajnej odchýlke aproximačnej funkcie, presnejšie známych vybraných hodnôt, ktoré boli vybrané zo znalostnej databázy, a z ktorých sa vykonáva aproximácia (interpolácia alebo

regresia). Tento pomer rozhodne, aká veľká časová rezerva sa pripočíta ku odhadovanému exekučnému času. Pre pomer potom platí.

$$R = \begin{cases} 1 & R > 1 \\ R & R \leq 1 \end{cases} \tag{8.5}$$

Potom je možné vypočítať finálnu časovú rezervu.

$$S = A(N_S) + R \times A(N_S) \tag{8.6}$$

8.6.7 Spoločné vlastnosti plánovačov

Konkrétne implementované plánovače vykonávajú a majú zopár vlastností, ktoré je nutné spomenúť pre ich úplné pochopenie.

Vhodné a nevhodné rozmery úlohy

Spomínané plánovače, ktoré používajú dáta zo znalostnej databázy rozlišujú, aké dáta treba z databázy vybrať. Spomínaná metóda **get_results_threads()** obsahuje ako voliteľný parameter Booleanovskú hodnotu z metódy **has_good_shape()**, kde na základe tejto hodnoty budú vybrané výsledky, ktoré majú, alebo nemajú vhodný rozmer spomínaný v časti 8.5. Z toho vyplýva, že ak na vstup plánovača príde úloha, ktorá má zlý rozklad prvočísiel, jej odhad vykonaný plánovačom bude vykonaný iba na základe úloh so zlým rozkladom. Rovnako to platí aj naopak. Čiže všetky listy, dáta predávané aproximátorom majú vždy len vlastnosť vhodných alebo nevhodných rozmerov úlohy.

Filtrovanie zlých aproximácii

Po dokončení odhadu plánovač filtruje nevhodné aproximácie pomocou metódy **filter_wrong_approximations()**. Výsledky sú odfiltrované pri výskyte zápornej hodnoty v predikcii exekučného času.

8.6.8 Získavanie dát

Pretože plánovač pracuje na základe dát, je nutné získať prvé dáta na základe ktorých bude plánovač pracovať, ďalšie dáta už môžu byť vkladané od používateľov. Pre tento problém boli vytvorené dva moduly **Agent** a **Node**.

Agent

Tento modul na základe predaných informácii vytvorí pole pracovných skriptov. Informácie sa predávajú ako parametre vstupu skriptu. Parametre sú nasledujúce.

- **-f** - Cesta k vstupným súborom simulácie (úlohám) na ktorých bude zbieranie dát prebiehať
- **-t** - Počet vlákien, na ktorých bude úloha prebiehať
- **-b** - Cesta k binárnej verzii k-Wave, ktorá bude spustená

Na začiatku Agent kontroluje či sú tieto cesty platné. Potom prebieha dotazovanie používateľa na ďalšie parametre simulácie, na základe ktorých Agent potom generuje pracovný skript. Tieto parametre sú.

- Názov klastru patriaceho IT4I - aby mohol byť uložený v databáze
- Počet jadier jedného uzla klastra - pre výpočet potrebných uzlov
- Typ fronty klastra - aby Agent mohol nastaviť, v ktorej fronte sa vygenerovaný skript spustí
- Doba alokácie - čas potrebný pre priebeh zberu dát
- Binárna verzia k-Wave - pre ukladanie do databázy a nastavenie modulov na klastri
- Adresa tunela SSH - tunel pre pripojenie do znalostnej databázy
- Port tunela SSH - port tunela pre pripojenie do znalostnej databázy

Je možné si všimnúť, že Agent ako parameter berie adresu SSH tunela. Tento SSH tunel je nutné vytvoriť na prihlasovacom uzle na danom klastri IT4I, pretože výpočtový uzol podporuje pripojenie iba v lokálnej sieti. Tunel musí smerovať od IT4I prihlasovacieho uzla, ku koncovému zariadeniu na ktorom beží inštancia znalostnej databázy. Na výpočtovom uzle konkrétny vygenerovaný pracovný skript od Agenta vytvára ďalší tunel, ktorého adresa a port je zadaná používateľom pri generovaní skriptu. Týmto sa vytvorí druhý tunel od výpočtového uzlu klastra k prihlasovaciemu uzlu a vznikne vzájomné pripojenie od výpočtového uzlu až k databáze.

Na základe všetkých týchto informácii Agent vytvára bash skript, ktorý po vytvorení môže používateľ zadať do fronty pomocou **qsub**. V tomto skripte sú ďalej uvedené kroky pre nastavenie klastra a nakoniec hlavné spustenie modulu Node, ktorý je vysvetlený ďalej.

Modul Agent môže byť spustený na ľubovoľnom počítači ale uverejňovanie skriptu už musí byť vykonané na klastri.

Node

Modul Node vykonáva exekúciu zadaných simulácii (úloh) už na danom klastri, keď mu plánovač predá riadenie a výpočtové zdroje.

Proces tohto modulu je invokovaný pracovným skriptom. Node začne postupne spúšťať k-Wave binárnu verziu, kde na vstupe mu Node predáva úlohy, ktoré boli predané v module Agent ako parameter. Tieto úlohy (simulácie) púšťa iba na 100 krokov. To znamená že Node spúšťa vždy jeden beh ku jednej úlohe (simulácii). Po dokončení takéhoto behu, začne Node spracovávať výsledný súbor simulácie, kde z výsledného súboru načíta dáta.

- Veľkosť úlohy
- Počet krokov
- Parametre - lineárnosť, absorbcia, homogenita
- Trvanie doby exekúcie
- Počet vlákien

Následne sú tieto informácie z výstupu uložené do modelu databázy **ResultsModel** ako riadok v tabuľke **Results**, viz. sekcia 7.3.1. Node spomínanú operáciu vykonáva pre každú úlohu v priečinku uvedenom v module Agent pri generovaní pracovného skriptu. Pripojenie do databázy je realizované cez spomínaný SSH tunel v sekcii Agent.

Skutočné získanie dát

Skutočné získanie dát prebehlo pomocou spomínaných modulov. Vstupné súbory pre testovanie boli vytvorené pomocou MATLAB modulu **k-Wave-Tester**. Vytvorené vstupné súbory boli prenesené na klastre, kde následne boli vytvorené pracovné skripty. Skripty boli spustené a výsledky boli uložené do databázy.

Pretože nie je možné pokryť každú variantu úlohy, úlohy pre náber výsledkov boli vygenerované logicky vzhľadom na viacero vlastností. Z pohľadu prvočíselných násobkov.

- Úlohy s vhodným prvočíselným rozkladom
- Úlohy s nevhodným prvočíselným rozkladom

Majorita úloh pre náber výsledkov do znalostnej databázy bola s vhodným prvočíselným rozkladom, dôvod prečo je odôvodnený na začiatku kapitoly 9. Menšia časť nevhodných úloh bola otestovaná z dôvodu, aby plánovač vedel odpovedať na úlohy takýchto rozmerov. Hlavný účel je ale predikovať úlohy vhodných rozmerov. Ďalej z pohľadu pokrytia rozmerov úlohy.

- Malé úlohy - $64 \times 64 \times 64, 128 \times 128 \times 128, 256 \times 256 \times 256$
- Stredné úlohy - $512 \times 512 \times 512, 728 \times 728 \times 728$
- Veľké úlohy - $896 \times 896 \times 896, 1024 \times 1024 \times 1024$

Tieto rozmery sa snažia pokryť čo najväčšiu množinu úloh, a teda čo najlepšie pokryť priestor úloh, a tým pádom vytvoriť základné "záchytné" body pre aproximáciu. Po diskusii s vedúcim práce a vlastných testoch sa zistilo, že úlohy väčšie než $1024 \times 1024 \times 1024$ sú menej obvyklé, a potrebujú veľmi veľké výpočtové zdroje.

Ďalší pohľad na úlohy môže byť z hľadiska vlastnosti média, tieto vlastnosti sú popísané v časti 5.2. Pri naberaní skutočných výsledkov bola pre každú veľkosť úlohy spomínanú v predchádzajúcom vymenovaní rozmerov, spustená každá možná variácia úlohy z pohľadu týchto vlastností. Úloha môže mať 3 skúmané vlastnosti (5.2), kde každá vlastnosť má 2 stavy. To znamená napríklad, že pre jednu úlohu veľkosťou $64 \times 64 \times 64$ bolo spustených 8 rôznych úloh.

$$N = 2 \times 2 \times 2 = 8 \tag{8.7}$$

Kde:
N - Počet možností jednej úlohy.

Dôležitý faktor pri naberaní výsledkov je počet vlákien. Rovnako je nutné pokryť aj tento logický priestor. Tu sa už počty vlákien rozdeľujú podľa binárnej verzie k-Wave.

k-Wave OMP Pretože OMP verzia dokáže pracovať iba na jednom fyzickom procesore a na všetkých jeho fyzických jadrách, je jednoduché pokryť tento logický priestor vlákien.

Tým pádom pre klaster Salomon, ktorý obsahuje procesory s 24 fyzickými jadrami boli úlohy spustené s vláknami v počtoch od $1, ..., 24$. Ďalej pre klaster Anselm, ktorý obsahuje procesory s 16 fyzickými jadrami boli rovnako úlohy spustené s vláknami v počtoch $1, ..., 16$. Takto sa jednoducho dali pokryť všetky možnosti. Všetky spomínané rozmery úloh boli spustené na spomínaných počtoch vlákien.

k-Wave CUDA CUDA verzia pracuje na grafickej karte a silu procesora využíva iba na zopár jednoduchých úloh, preto nie je nutné merať výsledky na veľkých počtoch vlákien a zároveň grafických kariet. Meranie preto podobne prebehlo iba na jednom uzle ako pri OMP.

k-Wave MPI Pri MPI sa situácia z pohľadu vlákien výrazne mení. Úloha môže byť spustená na viacerých uzloch, procesoroch. V tejto možnosti sa teoreticky dá použiť na výpočet celý klaster, resp. všetky uzly (v praxi skoro nereálne).

Tu vzniká podobný problém ako pri určovaní, aké veľkosti úloh sa použijú na náber dát. Pokryť každý počet vlákien je takmer nereálne a nepraktické. Preto výber počtu uzlov bol následovný.

- Malé počty vlákien/uzlov - 24 vlákien (1 uzol Salomon) / 16 (1 uzol Anselm) vlákien až 264 (11 uzlov Salomon) / 256 (16 uzlov Anselm) vlákien po násobkoch 24 (Salomon) a 16 (Anselm), lacnejšia cena, zvládnutejšie a viac pravedpodobné že aproximácie pre tieto počty budú žiadané
- Väčšie počty vlákien/uzlov - 528 **a** 1032 vlákien, veľmi drahé a menej pravdepodobné že používateľ bude chcieť vykonávať výpočet na takom veľkom počte vlákien, uzlov

Pri vykonávaní merania a naberania dát sa objavil problém, kde pri MPI verzii a pri veľkom rozmere úlohy (napr. $726 \times 726 \times 726$), sa k-Wave zasekne pri kontaktovaní ostatných procesorov.

Kapitola 9

Experimentálna časť

Táto kapitola obsahuje porovnanie aproximácii, presnosť odhadov a porovnanie aproximácii so skutočnosťou. Pre experimentálne účely boli vytvorené dve testovacie sady. Tieto sady obsahujú množinu vstupných simulácii pre k-Wave (všetky verzie). Pri vytváraní týchto sád sa dávalo na pozor, aby ani jedna simulácia nemala rozmer úlohy, ktorý už existuje v znalostnej databáze. Inak by aproximátory odpovedali známym výsledkom.

Prvá testovacia sada obsahuje úlohy, ktoré majú veľkosti vhodných rozmerov, presnejšie ich prvočíselný rozklad je vhodný pre k-Wave binárne verzie 5.1.

Druhá testovacia sada obsahuje úlohy, ktoré nemajú veľkosti vhodných rozmerov, čiže prvočíselný rozklad nie je vhodný pre k-Wave binárne verzie 5.1.

Po debate s vedúcim práce, sme došli k názoru že nie je nutné, aby bol plánovač schopný presne alebo približne aproximovať exekučný čas a počet vlákien pre takto nevhodné úlohy. Ako bude v tabuľkách ďalej vidieť, tieto úlohy zaberajú niekoľkonásobne viac času, ako vhodné úlohy, pričom rozdiel rozmerov medzi takýmito dvoma úlohami môže byť minimálny. Taktiež sa prišlo na to, že vstupnú úlohu s nevhodnými rozmermi je možné vyplniť číslami z originálnej hrany úlohy (replikácia) na vhodný rozmer. Týmto je možné potom zaručiť presnejšiu aproximáciu a rýchlejší beh úlohy. Takto umelo vyplnené čísla v rozmeroch simulácie, presnejšie v jej bodoch (hodnotách), neovplyvnia výsledok simulácie. Pre demonštráciu je možné použiť úlohu rozmeru $177 \times 177 \times 177$.

$$177 \times 177 \times 177 = 189 \times 189 \times 189* \tag{9.1}$$

* - Rekonštruovaná úloha $189 \times 189 \times 189$ obsahuje navyše rozmery $12 \times 12 \times 12$, kde tieto rozmery obsahujú hodnoty hrany replikované z originálnej úlohy.

Všetky nasledujúce spomínané simulácie (úlohy), ktoré je vidieť v testovacej sade a tabuľkách sa skladajú vždy zo 100 časových krokov.

9.1 Odhad pre k-Wave OMP

Nasledujúci obrázok 9.1 zobrazuje priebeh aproximácii všetkých aproximátorov. Priebeh zobrazuje aproximácie rozmerov úloh od veľkosti $32 \times 32 \times 32$ až po $1024 \times 1024 \times 1024$, pre úlohy vhodných rozmerov (prvočíselný rozklad). Tieto aproximácie sú pre binárnu verziu k-Wave OMP, klaster Salomon a 24 vlákien. Aproximácie sa na prvý pohľad správajú veľmi podobne. V tabuľke 9.1 je ale vidieť, že aproximácie naberajú iné hodnoty.

Pre zadaný počet vlákien v týchto meraniach najpresnejšie predikoval aproximátor vykonávajúci kubický spline, hneď po ňom Akima spline a PCHIP interpolácia. Tieto 3 aproximátory sa ale medzi sebou líšili v stotinách sekúnd, čo je zanedbateľné vzhľadom na to,

že pri zadávaní úlohy na superpočítač sa ešte musí pridať určitá časová rezerva, napr. pre načítanie dát, exportovanie dát, načítanie modulov... Je dôležité pripomenúť že rozmery v tabuľkách majú vhodný prvočíselný rozklad, čiže ich aproximácia je vykonávaná na základe rovnako vhodných známych výsledkov, sekcia 8.5.

Po týchto aproximátoroch sa umiestnil lineárny spline, ktorý predviedol chvályhodné výsledky. Najhoršie dopadli aproximátory využívajúce regresiu, kde sú výsledky horšie, nepresnejšie. Posledný výsledok v tabuľke ($600 \times 600 \times 600$) pridáva istú časovú rezervu a necháva približne 70 - 80 sekúnd naviac, táto rezerva súvisí s veľkosťou úlohy. Ako je vidieť, rezerva na menších úlohách nechá malú rezervu. Na väčších úlohách, nechá veľkú rezervu vždy, v prípade keby sa niečo zlé stalo, napr. zaseknutie, načítavanie dát.

Druhá tabuľka 9.2 zobrazuje porovnanie aproximátorov, ktoré vykonávajú predikciu času a predikciu počtu optimálnych vlákien pre binárnu verziu k-Wave OMP a klaster Salomon pre prvú testovaciu sadu - vhodné rozmery úloh.

Tu je vidieť, že aproximátory (spôsoby predikcie) sa už úplne nezhodujú, pokiaľ ide o predikciu dvojice exekučného času a počtu vlákien vzhľadom na najlepšiu cenu. Pri malých úlohách sa zhody nájdu. Rozdiely začínajú pri stredných a vyšších rozmeroch úloh. Pri veľkých úlohách sa už ale aproximátory začínajú zhodovať na rovnakom počte vlákien, pretože na menších počtoch vychádza cena vyššia (pretože výpočet trvá dlho).

Najlepšie z týchto odhadov dopadol aproximátor vykonávajúci interpoláciu Akima splinom, kde pri niektorých úlohách odhadol veľmi presný exekučný čas (úloha $100 \times 100 \times 100$). Veľmi tesne za týmto aproximátorom je aproximátor vykonávajúci kubický spline, ktorý sa v niektorých prípadoch zmýlil (úloha $100 \times 100 \times 100$), ale v niektorých na druhej strane podal podobný výkon ako Akima spline (úloha $100 \times 200 \times 100$, $40 \times 405 \times 405$).

Ako tretí aproximátor sa umiestnil PCHIP interpolátor. Táto metóda vytvára predikcie pripomínajúce zmiešanie kubického splinu a Akima spline.

Štvrtý aproximátor je lineárny spline, ktorý vo väčších úlohách nadobúda ale väčšie rozdiely od reálneho času, pretože lineárne funkcie rastú konštantne, kde pri veľkých úlohách vznikajú väčšie rozdiely, a tým interpoľačná funkcia rastie prudšie.

Na poslednom mieste sú aproximátory vykonávajúce regresie. Tu sú výsledky veľmi nespoľahlivé a nepresné, pretože spôsob aproximácie pomocou regresie je vykonávaný inak a regresia je presnejšia na základe viac známych bodov. Tým pádom v budúcnosti používania plánovača a naberaní výsledkov, by sa mohla presnosť tejto metódy zlepšiť.

Tretia tabuľka 9.3 zobrazuje porovnanie aproximátorov, ktoré vykonávajú rovnako predikciu času a počtu optimálnych vlákien pre binárnu verziu k-Wave OMP a klaster Salomon, použitá je druhá testovacia sada - nevhodné rozmery úloh.

Tu je možno vidieť, ako silno záleží na rozmere úlohy a na jej prvočíselnom rozklade. Úloha $269 \times 269 \times 269$, ktorá má zlý celočíselný rozklad trvá $6.29\times$ dlhšie ako blízko vzdialená úloha $200 \times 200 \times 200$, ktorá má vhodný rozmer, prvočíselný rozklad.

V tomto meraní najlepšie dopadli aproximácie pomocou lineárneho a Akima splinu. Je ale vidieť, že aproximátory majú problémy s aproximáciou, raz exekučný čas preceňujú a raz podceňujú. To je zapríčinené tým, že úlohy nevhodného rozmeru môžu mať prvočíselný rozklad ľubovoľné prvočíslo, a v skutočnosti nie je možné pokryť všetky veľké prvočísla znalostnou databázou. Čím väčšie prvočíslo v prvočíselnom rozklade, tým dlhšie úloha trvá.

Najlepšie ale dopadol aproximátor pracujúci na základe lineárneho splinu, čo sa dá odôvodniť faktom, že lineárny spline dokáže stúpať veľmi prudko, lineárne, a neosciluje tak silno ako kubický spline. Taktiež, keď priamka medzi známym predposledným a posledným bodom lineárneho spline stúpa, krivka za posledným známym bodom bude vždy stúpať, čiže extrapolácia bude vždy stúpať. Toto je možné vidieť na poslednom zázname v tabuľke,

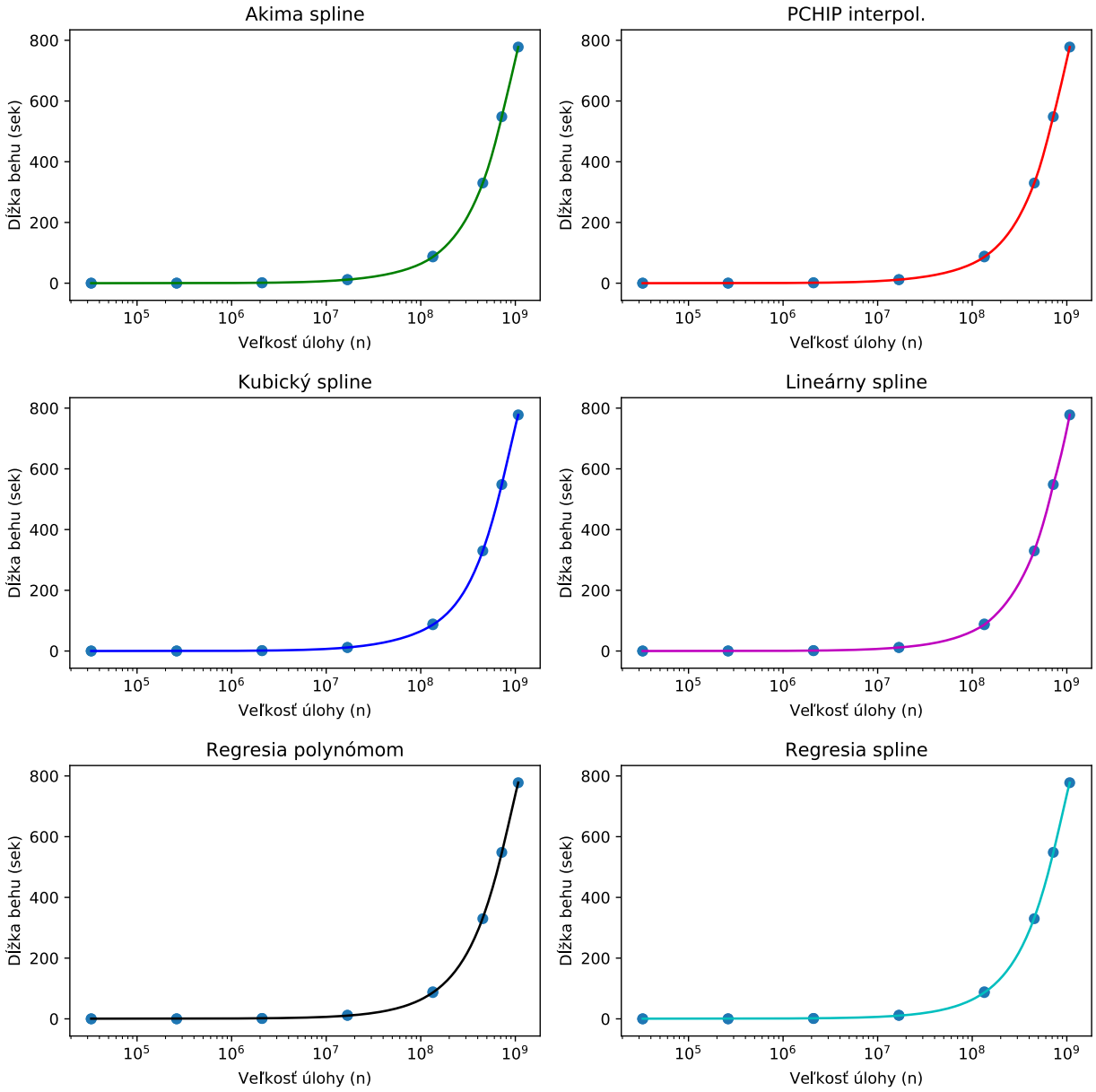
kde úloha $719 \times 719 \times 719$ je extrapolovaná, a lineárny spline extrapoluje vhodný odhad, pričom ostatné interpolácie silno oscilujú.

9.1.1 Zhrnutie

Zo skúmania sa najlepšia metóda určuje ťažko. Pri úlohách s rozmermi, ktoré majú vhodné prvočíselné násobky a zadaný počet vlákien je najvhodnejšia metóda aproximácie pomocou kubického splinu, poprípade je možné použiť Akima spline.

Simulácie, pri ktorých je nezadaný počet vlákien a majú vhodné rozmery, je najpresnejšia metóda Akima spline, podobná náhrada môže byť aj kubický spline.

Pri simuláciách, ktoré majú nevhodné rozmery je najpresnejšia dvojica metód Akima spline a lineárny spline.



Obr. 9.1: Porovnanie priebehov aproximátorov pre OMP k-Wave. Pevne zadaných 24 vlákien na vstup plánovača. Diskrétne body zobrazujú známe výsledky zo znalostnej databázy

Tabuľka 9.1: Tabuľka porovnávajúca odhady aproximátorov (plánovača) pre 24 jadier (zadaných) na prvej testovacej sade. k-Wave OMP - Salomon klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom	Počet vlákien
$40 \times 98 \times 40$	**0.09**	**0.11**	**0.11**	**0.11**	**0.11**	0.69	0.69	24
$100 \times 100 \times 100$	**0.53**	0.60	**0.59**	0.62	0.60	1.19	1.19	24
$100 \times 200 \times 100$	**1.08**	1.25	**1.24**	1.25	1.25	1.77	1.78	24
$98 \times 40 \times 600$	**1.37**	**1.48**	**1.48**	1.49	**1.48**	1.99	1.99	24
$100 \times 200 \times 200$	**2.41**	2.62	2.61	2.67	**2.59**	2.98	2.98	24
$40 \times 405 \times 405$	**4.12**	4.46	**4.42**	4.51	**4.42**	4.53	4.53	24
$200 \times 200 \times 200$	**4.60**	5.52	5.46	5.56	5.48	**5.42**	**5.42**	24
$600 \times 405 \times 294$	**42.13**	**53.78**	55.74	54.17	54.12	53.8	53.8	24
$448 \times 448 \times 448$	**50.86**	69.31	71.48	70.35	69.76	69.91	69.91	24
$405 \times 600 \times 600$	**100.24**	126.62	**125.35**	127.6	126.28	131.7	131.8	24
$600 \times 600 \times 600$	**133.21**	221.50	**213.7**	227.7	219.64	232.88	232.88	24

Tabuľka 9.2: Tabuľka porovnávajúca odhady aproximátorov (plánovača) iba pre zadanú veľkosť úlohy - prvá testovacia množina. k-Wave OMP - Salomon klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$40 \times 98 \times 40$	16: **0.10**	16: 0.08	16: 0.08	16: 0.08	16: 0.08	18: 0.05	18: 0.05
$100 \times 100 \times 100$	16: **0.56**, 22: **0.48**	16: 0.56	22: 0.40	16: 0.60	22: 0.42	12: 0.07	12: 0.07
$100 \times 200 \times 100$	16: **1.24**	16: 1.24	16: 1.24	16: 1.25	16: 1.24	16: -0.19	16: -0.19
$98 \times 40 \times 600$	16: **1.64**	16: 1.51	16: 1.50	16: 1.50	16: 1.49	16: 0.10	16: 0.10
$100 \times 200 \times 200$	16: **2.65**, 8: **4.03**	16: 2.78	16: 2.74	16: 2.78	16: 2.66	8: 2.11	8: 2.11
$40 \times 405 \times 405$	16: **4.73**	16: 4.78	16: 4.71	16: 4.78	16: 4.61	16: 3.79	16: 3.79
$200 \times 200 \times 200$	16: **4.60**	16: 5.91	16: 5.84	16: 5.92	16: 5.74	16: 5.07	16: 5.07
$600 \times 405 \times 294$	16: **46.62**, 19: **45.58**, 20: **43.08**	19: 58.35	16: 63.68	16: 64.76	16: 63.27	20: 51.99	20: 51.99
$448 \times 448 \times 448$	16: **60.47**, 19: **56.36**, 20: **54.58**	19: 75.45	16: 83.49	16: 85.04	16: 83.05	20: 68.94	20: 68.94
$405 \times 600 \times 600$	20: **110.44**	20: 133.98	20: 133.18	20: 135.54	20: 134.27	20: 131.33	20: 131.33
$600 \times 600 \times 600$	20: **126.06**	20: 229.51	20: 223.69	20: 239.67	20: 230.43	20: 20: 236.25	20: 236.25

Tabuľka 9.3: Tabuľka porovnávajúca odhady aproximátorov (plánovača) iba pre zadanú veľkosť úlohy - druhá testovacia množina. k-Wave OMP - Salomon klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$177 \times 177 \times 177$	24: **8.06**	24: 8.07	24: 8.28	24: 8.02	24: 8.17	24: 8.28	24: 8.28
$177 \times 213 \times 177$	24: **9.19**	24: 9.63	24: 9.83	24: 9.60	24: 9.75	24: 10.17	24: 10.17
$213 \times 393 \times 177$	24: **20.78**	24: 21.76	24: 21.32	24: 21.86	24: 21.46	24: 22.93	24: 22.93
$269 \times 269 \times 269$	24: **28.95**	24: 31.16	24: 31.16	24: 31.16	24: 31.16	24: 29.66	24: 29.66
$514 \times 466 \times 117$	24: **45.58**	24: 42.16	24: 42.05	24: 42.15	24: 42.19	24: 41.33	24: 41.33
$466 \times 514 \times 213$	24: **51.69**	24: 70.14	24: 69.48	24: 70.96	24: 70.24	24: 71.50	24: 71.50
$393 \times 393 \times 393$	24: **84.48**	24: 85.49	24: 84.90	24: 85.82	24: 85.50	24: 85.43	24: 85.43
$419 \times 419 \times 419$	24: **97.12**	24: 108.19	24: 118.65	24: 105.58	24: 108.87	24: 106.82	24: 106.82
$393 \times 514 \times 466$	24: **129.45**	24: 141.77	24: 225.91	24: 137.18	24: 150.90	24: 151.86	24: 151.86
$514 \times 514 \times 514$	24: **159.61**	24: 146.07	24: 787.53	24: 201.17	24: 235.77	24: 309.79	24: 309.79
$719 \times 719 \times 719$	24: **545.08**	24: -6800.39	24: 27048.96	24: 563.50	24: -1073.00	24: 5493.30	24: 5493.30

9.2 Odhad pre k-Wave CUDA

Táto podkapitola, podobne ako pre OMP, zobrazuje odhady plánovača, presnejšie špecifických aproximátorov na testovacej sade vstupných simulácii. Testovacia množina simulácii pri tomto type binárnej verzie má menšie rozmery ako množina pre OMP, pretože klaster Anselm má menšiu veľkosť pamäte RAM a tým pádom nezvládne väčšie rozmery. Približný odhad tohto maximálneho rozmeru je $350 \times 350 \times 350$.

Prvý obrázok 9.2, zobrazuje odhadovaný priebeh aproximátorov pre k-Wave CUDA. Najlepšie z týchto grafov vyzerá aproximácia pomocou Akima spline, hneď za ňou nasleduje aproximácia PCHIP interpoláciou. Nesprávne ale vyzerá kubický spline, kde v tejto situácii dochádza ku zlej vlastnosti tejto interpolácie, a to ku "overshooting", presnejšie ku oscilácii.

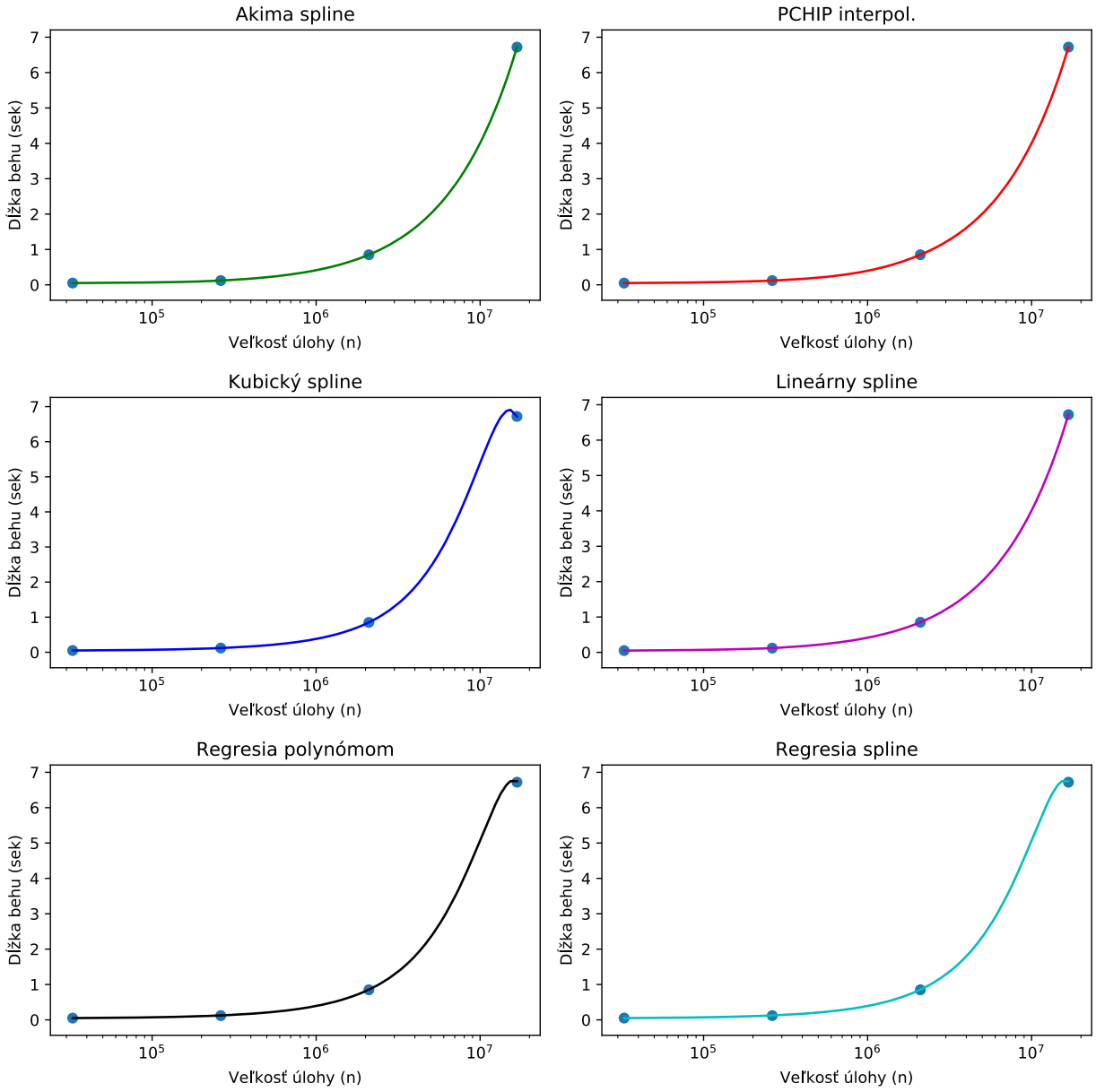
Tabuľka 9.4, zobrazuje aproximácie pre prvú testovaciu sadu porovnávané s realitou. Počet vlákien bol uvedený pri vstupe, takže plánovač odhaduje časy behov pre pevný (zadaný) počet vlákien. Z týchto aproximácii je vidieť, že najlepšie odhady zvládli Akima spline a PCHIP interpolácia, ktorých výsledky sa veľmi podobajú realite. Následne je vidieť jav oscilácie pri kubickom spline, kde interpoľačná funkcia vďaka polynómom začína stúpať rýchlejšie ako spomínané interpolácie, pri úlohe $200 \times 200 \times 200$ polynom osciluje smerom nahor rýchlejšie. V poslednom zázname ($300 \times 300 \times 300$) už polynom osciluje nadol, až presiahne do zápornej časti osi Y. Lineárny spline vyzerá ako stredná cesta, na ktorú sa dá ale menej spoľahnúť. Najhoršie opäť dopadli oba spôsoby regresie, čo môže byť opäť zapríčinené nedostatkom znalostných dát alebo spôsobom aproximácie.

Nasledujúca tabuľka 9.5 zobrazuje aproximácie pre prvú testovaciu sadu porovnávané s realitou, kde aproximátory odhadujú dvojicu počet vlákien a exekučný čas úlohy. Ako vidieť, plánovač vždy odhaduje počet vlákien ako jedno. Toto potvrdzuje tvrdenie v časti 5.1, kde k-Wave CUDA pracuje iba na grafickej karte a na procesore vykonáva iba minimum úloh (načítanie dát, uloženie dát...). Plánovač vždy odhadol jedno vlákno, pretože väčšia cena za rovnaký výpočetný výkon je nelogický odhad podľa vzorca 8.1. Toto potvrdzujú aj výsledky aproximácii, ktoré sú veľmi podobné ako pre jedno vlákno tak, i pre 16 vlákien. Z tohto merania sa dá opäť vyvodiť, že aproximácia Akima splinom a PCHIP interpoláciou vykazuje najlepšie výsledky aproximácie. Pri väčších úlohách ($300 \times 300 \times 300$) už je vidieť, že Akima spline a PCHIP interpolácia naberá väčšieho rastu ako pri OMP, tým obieha skutočný čas a necháva časovú rezervu, ktorá nie je tak veľká, a môže v niektorých situáciach byť nápomocná používateľovi.

Ďalšia tabuľka 9.6 porovnáva aproximácie pre druhú testovaciu sadu, presnejšie pre nevhodné rozmery úloh. Je možné vidieť, že pri zbere dát sa oplatilo nabrať zopár nevhodných behov pretože plánovač vďaka ním dokáže vcelku presne vytvoriť predikciu aj pre takéto úlohy. V týchto výsledkoch a medzi aproximátormi nie je vidieť až taký silný rozdiel, pretože takýchto výsledkov nie je veľa a aproximácia prebieha medzi zopár bodmi. Vzhľadom na to, že veľký rozdiel medzi aproximátormi nenastal, z pohľadu na výsledky aproximátorov použité v iných verziách k-Wave je pre zachovanie najmenšej oscilácie a presnosti najlepší výber dvojica aproximátorov Akima spline a PCHIP interpolácia.

9.2.1 Zhrnutie

Pre tento typ implementácie k-Wave je z meraní vidieť, že najlepšie výsledky podáva dvojica Akima spline a PCHIP interpolácia, a to pre oba typy vstupu. Použiteľný je aj lineárny spline ako stredná cesta. Najhoršie z experimentov opäť vychádza dvojica vykonávajúca regresiu.



Obr. 9.2: Porovnanie priebehov aproximátorov pre CUDA k-Wave. Pevne zadané 1 vlákno na vstup plánovača. Diskrétne body zobrazujú známe výsledky zo znalostnej databáze

Tabuľka 9.4: Tabuľka porovnávajúca odhady aproximátorov (plánovača) pre 16 jadier (zadaný) na prvej testovacej sade (vhodné rozmery). k-Wave CUDA - Anselm klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom	Počet vlákien
$40 \times 98 \times 40$	**0.10**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	16
$100 \times 100 \times 100$	**0.54**	**0.47**	0.43	**0.47**	0.45	0.43	0.43	16
$100 \times 200 \times 100$	**0.95**	**1.03**	**1.03**	1.04	1.04	1.04	1.04	16
$98 \times 40 \times 600$	**1.20**	**1.27**	1.31	**1.27**	**1.27**	1.32	1.32	16
$100 \times 200 \times 200$	**1.67**	2.54	3.08	2.54	**2.53**	3.11	3.11	16
$40 \times 405 \times 405$	**3.58**	5.12	6.79	5.13	**5.11**	6.44	6.44	16
$200 \times 200 \times 200$	**3.91**	**6.41**	8.55	6.42	**6.41**	8.05	8.05	16
$300 \times 300 \times 300$	**11.40**	**21.67**	-7.30	21.61	**21.69**	-2.94	-2.94	16

Tabuľka 9.5: Tabuľka porovnávajúca odhady aproximátorov (plánovača) iba pre zadanú veľkosť úlohy, prvá testovacia sada (vhodné rozmery). k-Wave CUDA - Anselm klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$40 \times 98 \times 40$	1: **0.10**	1: 0.09	1: 0.09	1: 0.09	1: 0.09	1: 0.09	1: 0.09
$100 \times 100 \times 100$	1: **0.47**	1: 0.47	1: 0.43	1: 0.47	1: 0.45	1: 0.43	1: 0.43
$100 \times 200 \times 100$	1: **0.95**	1: 1.04	1: 1.03	1: 1.04	1: 1.04	1: 1.04	1: 1.04
$98 \times 40 \times 600$	1: **1.20**	1: 1.27	1: 1.31	1: 1.27	1: 1.27	1: 1.32	1: 1.32
$100 \times 200 \times 200$	1: **1.67**	1: 2.54	1: 3.08	1: 2.54	1: 2.53	1: 2.95	1: 2.95
$40 \times 405 \times 405$	1: **3.58**	1: 5.12	1: 6.79	1: 5.13	1: 5.11	1: 6.44	1: 6.44
$200 \times 200 \times 200$	1: **3.90**	1: 6.41	1: 8.55	1: 6.42	1: 6.41	1: 8.05	1: 8.05
$300 \times 300 \times 300$	1: **11.39**	1: 21.67	1: -7.35	1: 21.61	1: 21.69	1: -2.94	1: -2.94

Tabuľka 9.6: Tabuľka porovnávajúca odhady aproximátorov (plánovača) pre 16 jadier (zadaný) na druhej testovacej sade (nevhodné rozmery). k-Wave CUDA - Anselm klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$177 \times 177 \times 177$	1: **13.17**	1: 16.88	1: 16.92	1: 16.59	1: 16.66	1: 17.33	1: 17.33
$177 \times 213 \times 177$	1: **16.36**	1: 21.65	1: 21.44	1: 20.88	1: 21.62	1: 20.15	1: 20.15
$197 \times 197 \times 197$	1: **24.34**	1: 24.58	1: 24.58	1: 24.58	1: 24.58	1: 22.44	1: 22.44
$213 \times 393 \times 177$	1: **42.69**	1: 37.24	1: 35.77	1: 38.14	1: 36.88	1: 37.73	1: 37.73
$269 \times 269 \times 269$	1: **48.31**	1: 49.01	1: 49.01	1: 49.01	1: 49.01	1: 49.34	1: 49.34

9.3 Odhad pre k-Wave MPI

Nasledujúci obrázok 9.3, zobrazuje priebeh všetkých aproximácii rôznych úloh od veľkosti $32 \times 32 \times 32$ až po $625 \times 625 \times 625$. Z obrázku je vidieť, že kubický spline, regresia polynómom a spline regresia veľmi silno oscilujú a tým prakticky poskytujú nepoužiteľné (záporné) alebo moc vysoké výsledky. Použitím tak vysokých výsledkov, presnejšie veľký exekučný čas by zapríčinil, že úloha by príliš dlho čakala vo fronte v plánovači. Týmto pádom použitie týchto aproximátorov nie je možné.

O dosť lepšie vyzerá priebeh Akima spline, lineárneho spline a PCHIP interpolácie. Vzhľadom na to, že tento plánovač používa viacero metód na odhad exekučného času, je možné vidieť, že exekučné časy od aproximátorov pri veľkých úlohách sa začínajú líšiť. Je možné si taktiež všimnúť, že odhad pre túto binárnu verziu je najzložitejší zo všetkých.

Bližšie presné výsledky je možné vidieť v tabuľke 9.7. Táto tabuľka zobrazuje odhady pre pevne zadaný počet vlákien a vhodné rozmery úloh, presnejšie pre 192 vlákien. Tieto behy boli spustené na klastri Salomon, z čoho potom vyplýva že 192 vlákien je ekvivalent 8 výpočetných uzlov. Osem výpočetných uzlov bolo vybraných, pretože naznačuje strednú cestu pri výbere ceny a efektivity výpočtov. V tabuľke je vidieť, ako začína oscilovať kubický spline a aproximátory využívajúce regresiu. Úspešné sú ale aproximácie vykonané Akima spline, PCHIP interpoláciou a lineárnym spline. Pri veľkých úlohách ale vidieť, ako už stúpa rozdiel aproximátorov, a ako sa interpolácie vzájomne líšia. Najstabilnejšie správanie predstavuje Akima spline a PCHIP interpolácia. Je vidieť, že predikčné funkcie pre exekučný čas týchto interpolácii začínajú opäť stúpať prudšie ako skutočný čas, a tým pádom nechávajú časovú rezervu.

Najhoršie dopadli opäť aproximátory pomocou regresie a kubického splinu. Všetky tri tieto aproximátory silne oscilujú, a ich výsledky sú skoro nepoužiteľné.

Ďalšia tabuľka 9.9 zobrazuje aproximácie plánovača bez zadaného počtu vlákien a vhodné rozmery. Čiže plánovač aproximuje najlepšiu dvojicu počet vlákien a exekučný čas úlohy na týchto vláknach. Je vidieť, že aproximátory sa zhodujú na počtoch vlákien vždy, pretože tento počet je odvodený zo vzorca 8.2. Hviezdou označené výsledky, nemohli byť vypočítané interpoláciou známych výsledkov pre známy počet vlákien, preto boli vypočítané interpoláciou najefektívnejších behov.

Opäť najlepšími aproximátormi sa stali aproximátory pracujúce na základe Akima splinu, PCHIP interpolácie a lineárneho spline. Pri väčších úlohách sa ale odhad aproximátorov začína rozlišovať, kde u úlohy $600 \times 600 \times 600$ lineárny spline, lineárne a prudko stúpa. PCHIP interpolácia a Akima interpolácia sú si veľmi podobné, no z testovacej množiny podal celkovo najlepšie výsledky Akima spline.

Najhoršie opäť dopadli aproximátory využívajúce regresiu a kubický spline, kde ako je vidieť, aproximátory aj pre iné počty vlákien silno oscilujú, a podávajú nepoužiteľné výsledky.

Tabuľka 9.8 zobrazuje predikcie plánovača pre druhú testovaciu sadu - pre úlohy so zlým rozmerom, resp. s veľkým prvočíselným rozkladom. Na vstup plánovača bolo zadaných 192 vlákien (8 uzlov - Salomon), pre ktoré plánovač vykonával predikciu exekučného času. Hviezdou označené výsledky v tabuľke značia výsledky, ktoré boli aproximované na základe dvojitej interpolácie popísanej v implementačnej časti plánovača.

Z hodnôt predikcii je možno vidieť, že aproximácie pre nevhodné rozmery úloh sú ťažko aproximovateľné. To je z dôvodu, že vhodné rozmery úloh sú prvočísla spomínané v 5.1, kde tieto čísla je jednoduché pokryť znalostnou databázou a taktiež znalostná databáza sa nimi zaoberá. Na druhej strane, tieto nevhodné rozmery, čiže nevhodné prvočísla v rozklade, sú

všetky ostatné prvočísla pričom počet týchto zvyšných prvočísiel je nekonečno. Čím väčšie prvočíslo je, tým dlhšie exekučný čas trvá.

Z týchto aproximácii najlepšie dopadla dvojica lineárny spline a Akima spline. Akima spline pre tieto výsledky síce osciluje, ale oscilácia je tlmená. Aj keď dochádza k tlmeniu oscilácie, exekučné časy sú pomerne vysoké v porovnaní s realitou. Napriek tomu tieto výsledky sú stále použiteľné a je možné na základe nich spúšťať exekúciu. Lineárny spline ale dopadol najlepšie, jeho nie veľká strmosť interpoľačnej funkcie zaručila väčšie predikované exekučné časy, kde ale na druhej strane tieto exekučné časy nie sú prehnané, a mohli by byť použité pre exekúciu.

Avšak ostatné aproximátory ukazujú podobne veľkú a silnú osciláciu, kde tieto časové predikcie sú skoro nepoužiteľné.

V tabuľke 9.10 sú záznamy predikcie plánovača pre druhú testovaciu sadu, kde plánovač predpovedal dvojicu počet vlákien a exekučný čas simulácie na tomto počte vlákien. Tu je predikcia ešte problematickejšia, pre úlohy zlých rozmerov neexistuje ideálny počet vlákien. Plánovač sa snaží vybrať iba ten najlepší možný, aký dokáže nájsť. Majorita výsledkov v tejto tabuľke je vypočítaná najefektívnejšiou aproximáciou. Ako je popísané v implementácii, najefektívnejšia aproximácia počíta s najlepšími rozmermi úloh v pomere ku počtu vlákien. Preto je možné vidieť, že plánovač preceňuje silu výpočetných uzlov a vždy trafí o trocha menší exekučný čas výpočtu, ako je reálny. Toto sa dá ale vyriešiť, upozornením používateľa, aby si pripočítal istú časovú poistku na základe rozmerov úlohy.

Z tejto predikcie najlepšie dopadla trojica aproximátorov vykonávajúca Akima spline, PCHIP interpolácia a lineárny spline. PCHIP interpolácia podáva najlepšie a realite najbližšie výsledky. Na druhej strane Akima spline má slabší rast interpoľačnej funkcie, preto podáva menšie časové predikcie. Lineárny spline má v niektorých častiach trocha prudší rast, čo môže naznačovať menšiu osciláciu, ale výsledky sú taktiež použiteľné.

Naopak ostatné aproximátory nepodali dobrý výkon a naberajú veľkej oscilácie, kde táto oscilácia robí časové výsledky až nepoužiteľné.

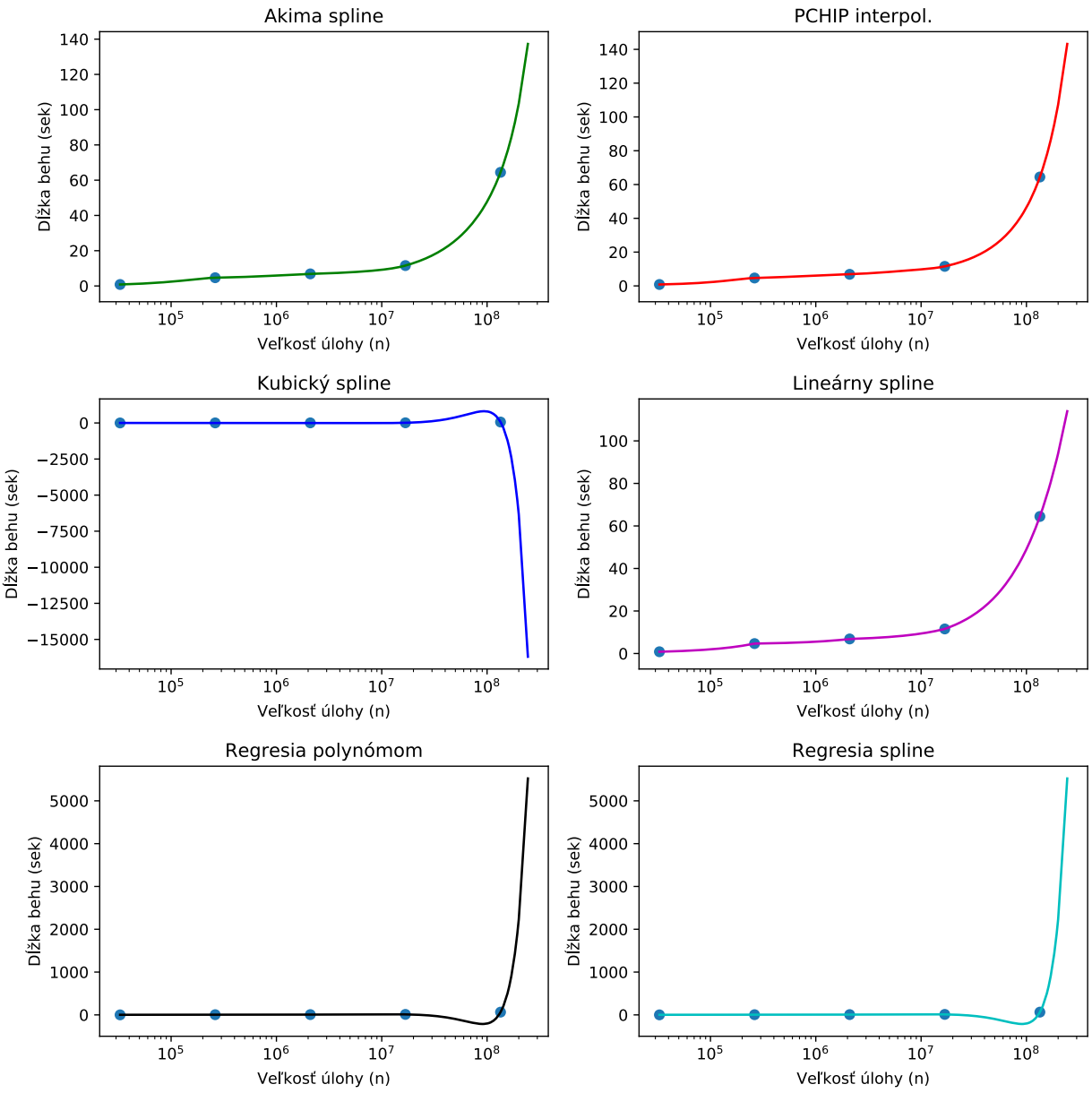
9.3.1 Zhrnutie

Pre k-Wave MPI je z výsledkov vidieť, že odhad je najnáročnejší. Plánovač podáva najlepšie výsledky so zadaným počtom vlákien, resp. používateľ žiada predikciu pre určitý počet jadier. Tento počet jadier by mal byť vhodne vybraný, a mať vhodnú veľkosť. Pri zadanom počte vlákien a vhodnej veľkosti úlohy najlepšie aproximácie vykonával Akima spline. Pri nevhodnej veľkosti úlohy, najlepšiu aproximáciu času vykonal aproximátor pracujúci na lineárnom spline.

Pri nezadanom počte vlákien a vhodnom rozmere úlohy, podal najlepší výkon aproximátor vykonávajúci PCHIP interpoláciu a taktiež podal najlepší výkon aj pri nevhodnom rozmere úlohy.

Na základe týchto výsledkov pre nevhodné rozmery úloh, je vhodné podotknúť že dvojitá interpolácia je náročný výpočet, a vo výsledkoch vznikajú väčšie nepresnosti ako vo výsledkoch najefektívnejšej aproximácii.

Najhoršie výsledky podávali aproximátory pracujúce na kubickom spline a regresii. Tieto aproximátory v majorite testov silne oscilovali, až došlo do štádia, kde ich výsledky boli nepoužiteľné kvôli veľmi veľkému alebo zápornému exekučnému času. Aproximácia pre túto k-Wave binárnu verziu je najnepresnejšia.



Obr. 9.3: Porovnanie odhadov aproximátorov pre MPI k-Wave - Salomon klaster. Pevne zadaných 192 vlákien na vstupe. Diskrétne body zobrazujú známe behy zo znalostnej databázy

Tabuľka 9.7: Tabuľka porovnávajúca odhady aproximátorov (plánovača) pre 192 jadier - 8 uzlov (zadaný) na prvej testovacej sade (vhodné rozmery). k-Wave MPI - Salomon klaster

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom	Počet vlákien
$40 \times 98 \times 40$	**2.47**	4.10	3.45	3.26	3.86	2.96	2.96	192
$100 \times 100 \times 100$	**3.61**	7.39	14.32	6.84	7.66	5.63	5.63	192
$100 \times 200 \times 100$	**5.01**	8.77	9.60	8.65	8.74	8.87	8.87	192
$98 \times 40 \times 600$	**8.96**	8.96	6.95	8.96	9.07	10.00	10.00	192
$100 \times 200 \times 200$	**7.44**	9.79	-12.5	9.81	10.44	15.05	15.05	192
$128 \times 448 \times 98$	**8.48**	10.57	-13.3	10.67	11.57	19.31	19.31	192
$100 \times 100 \times 600$	**7.77**	10.75	-12.25	10.87	11.80	20.17	20.17	192
$40 \times 405 \times 405$	**10.34**	11.02	-15.3	11.18	12.13	21.34	21.34	192
$200 \times 200 \times 200$	**9.36**	11.72	-14.25	11.97	12.89	23.69	23.69	192
$128 \times 600 \times 200$	**16.01**	16.07	6.13	16.32	16.30	20.33	20.33	192
$200 \times 400 \times 784$	**51.47**	62.51	1133.68	64.50	58.87	-2001.02	-2001.02	192
$600 \times 405 \times 294$	**62.37**	70.06	1350.55	72.35	66.37	-180.2	-180.2	192
$448 \times 448 \times 448$	**70.14**	86.42	1619.25	88.99	83.14	-75.3	-75.3	192
$405 \times 600 \times 600$	**109.15**	141.10	-320.5	139.97	142.32	524.66	524.66	192
$600 \times 600 \times 600$	**173.32**	229.19	-785.3	202.11	239.08	6438.02	6438.02	192
$784 \times 400 \times 784$	**206.79**	277.50	-1212.6	229.42	289.62	11377.50	11377.50	192

Tabuľka 9.8: Tabuľka porovnávajúca odhady aproximátorov (plánovača) pre 192 jadier - 8 uzlov (zadaný) na druhej testovacej sade (nevhodné rozmery). k-Wave MPI - Salomon klaster (* - sú označené aproximácie, ktoré boli vykonané dvojitou interpoláciou)

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom	Počet vlákien
$177 \times 177 \times 177$	**10.59**	14.48*	10018.16*	184.92*	5886.33*	237.45*	237.45*	192
$177 \times 177 \times 393$	**17.74**	203.31*	8202.08*	182.59*	4985.98*	233.64*	233.64*	192
$177 \times 177 \times 514$	**27.77**	305.01*	7291.49*	181.29*	4522.61*	231.61*	231.61*	192
$177 \times 393 \times 514$	**39.69**	516.52*	3671.38*	174.54*	2554.99*	-10.66*	-10.66*	192
$177 \times 514 \times 466$	**45.21**	1018.22*	2815.67*	172.25*	2040.56*	-13.43*	-13.43*	192
$213 \times 514 \times 393$	**49.28**	1012.23*	2742.86*	172.04*	1995.33*	-13.53*	-13.53*	192
$466 \times 393 \times 393$	**76.80**	405.80*	3181.70*	156.25*	512.25*	-2.68*	-2.68*	192
$393 \times 466 \times 466$	**99.48**	273.93*	1777.30*	145.68*	192.41*	-15.72*	-15.72*	192
$514 \times 393 \times 514$	**100.72**	61.83*	147.55*	127.93*	104.16*	48.93*	48.93*	192
$514 \times 514 \times 514$	**125.98**	391.64*	672.42*	234.49*	261.42*	212.90*	212.90*	192

Tabuľka 9.9: Tabuľka porovnávajúca odhady aproximátorov (plánovača) iba pre zadanú veľkosť úlohy - prvá testovacia sada (vhodné rozmery). k-Wave MPI - Salomon klaster (* - sú označené aproximácie, ktoré boli vykonané najviac efektívnou aproximáciou)

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$40 \times 98 \times 40$	48: **0.73**	48: 1.89	48: 1.66	48: 1.60	48: 1.79	48: 1.49	48: 1.49
$100 \times 100 \times 100$	120: **4.05**	120: 5.10	120: 11.59	120: 5.02	120: 5.04	120: 3.81	120: 3.81
$100 \times 200 \times 100$	120: **4.08**	120: 4.63	120: 5.49	120: 4.72	120: 4.70	120: 4.88	120: 4.88
$98 \times 40 \times 600$	48: **3.74**	48: 4.38	48: 3.68	48: 4.38	48: 4.39	48: 4.61	48: 4.61
$100 \times 200 \times 200$	216: **9.65**	216: 9.46*	216: 1.36*	216: 9.32*	216: 10.05*	X	X
$40 \times 405 \times 405$	408: **40.23**	408: 11.84	408: -1.87	408: 11.34	408: 12.67	X	X
$200 \times 200 \times 200$	216: **10.98**	216: 13.23	216: -1.71	216: 12.50	216: 13.99	X	X
$120 \times 600 \times 200$	216: **14.30**	216: 18.75*	216: 11.38*	216: 17.91*	216: 18.66*	X	X
$200 \times 400 \times 784$	408: **36.36**	408: 35.67*	408: 188.94*	408: 36.00*	408: 39.78*	X	X
$600 \times 405 \times 294$	312: **47.08**	312: 38.45*	312: 197.11*	312: 39.33*	312: 42.66*	X	X
$448 \times 448 \times 448$	456: **47.90**	456: 46.71*	456: 194.27*	456: 46.67*	456: 48.49*	X	X
$405 \times 600 \times 600$	600: **51.00**	600: 73.65*	600: 24.93*	600: 74.32*	600: 72.28*	X	X
$600 \times 600 \times 600$	600: **57.22**	600: 127.16*	600: 214.25*	600: 133.13*	600: 117.09*	X	X
$784 \times 400 \times 784$	408: **95.26**	408: 153.59*	408: 728.01*	408: 162.36*	408: 139.83*	X	X

Tabuľka 9.10: Tabuľka porovnávajúca odhady aproximátorov (plánovača) iba pre zadanú veľkosť úlohy - druhá testovacia sada (nevhodné rozmery). k-Wave MPI - Salomon klaster (* - sú označené aproximácie, ktoré boli vykonané najviac efektívnou aproximáciou)

Rozmer úlohy	Skutočný čas	Akima spline	Kubický spline	Lineárny spline	PCHIP interpol.	Spline regresia	Regresia polynómom
$177 \times 177 \times 177$	192: **9.90**	192: 10.87*	192: -1.19*	192: 10.53*	192: 11.68*	X	X
$177 \times 177 \times 393$	144: **18.95**	144: 15.47	144: 956.28	144: 41.30	144: 18.88	144: 131.56	144: 131.56
$177 \times 177 \times 514$	192: **20.92**	192: 19.72*	192: 17.45*	192: 19.41*	192: 19.65*	X	X
$177 \times 393 \times 514$	264: **35.21**	264: 26.91*	264: 102.28*	264: 26.31*	264: 29.33*	X	X
$177 \times 514 \times 466$	480: **63.74**	480: 29.14*	480: 130.41*	480: 28.62*	480: 32.18*	X	X
$213 \times 514 \times 393$	408: **45.30**	408: 29.35*	408: 132.94*	408: 28.84*	408: 32.44*	X	X
$466 \times 393 \times 393$	408: **44.95**	408: 38.62*	408: 197.41*	408: 39.53*	408: 42.83*	X	X
$393 \times 466 \times 466$	480: **59.56**	480: 43.09*	480: 197.43*	480: 44.81*	480: 47.05*	X	X
$514 \times 393 \times 514$	264: **99.92**	264: 50.11*	264: 173.17*	264: 52.47*	264: 53.06*	X	X
$514 \times 514 \times 514$	528: **89.22**	528: 66.99*	528: 60.04*	528: 67.08*	528: 66.80*	X	X

Kapitola 10

Záver

Cieľom práce bolo vytvoriť aplikáciu, nástroj (plánovač), ktorý dokáže vhodne aproximovať exekučné časy a počet vlákien pre simulácie spúšťané na toolboxe k-Wave. Na začiatku bolo potrebné sa zoznámiť s dnešnými plánovačmi na superpočítačoch, hlavne na klaster architektúre a zoznámiť sa s klastermi od skupiny IT4Innovations. Ďalej bolo nutné nájsť vhodné metódy pre odhad na základe získaných znalostí, dát.

V práci sa nachádza podrobné riešenie tohto problému. Implementácia, návrh tohto plánovača a experimentácia sa podarila. Bolo implementovaných niekoľko algoritmov pre vykonanie odhadov, ktoré využívajú interpoláciu a regresiu. Algoritmy sa správajú rozdielne, kde ich správanie je definované zadaným alebo nezadaným počtom vlákien a typom binárnej verzie k-Wave. Najpresnejšie, respektíve najlepšie odhady vznikajú pre malé až väčšie úlohy, ktoré majú vhodný rozmer z pohľadu prvočíselných rozkladov. Implementácia v kóde sa snaží byť čo najjednoduchšia na pochopenie, a obsahuje jednoduché rozhrania, pre implementáciu ďalších možných častí plánovača, ako napríklad databázové modely, aproximátory, repozitáre... Taktiež bola vytvorená vhodná databáza pre uchovávanie výsledkov, kde zároveň táto databáza je zakontajnerizovaná pomocou Docker. Z pohľadu metód aproximácie, najlepšie výsledky poskytovala trojica interpolačných metód Akima spline, kubický spline a PCHIP interpolácia. Každá interpolácia sa javila v niečom špeciálna, pričom ale nie je možné vybrať jednu najlepšiu metódu a tou sa vždy riadiť.

Práca v praxi môže byť využitá ako komponent do systému, ktorý samostatne spúšťa úlohy na klasteri a operuje okolo nich. Tento plánovač by rozhodoval automaticky o dĺžke exekúcie a počtoch vlákien pre každú zadanú simuláciu. Takýto systém existuje už vo vývoji s názvom k-Dispatch a je možné že bude tento plánovač používať.

V blízkej dobe by sa mohlo dať znalostnú databázu obohatiť ešte viac známymi behmi z rôzneho pohľadu konfigurácii. Na to by bolo ale treba viac jadro-hodín od klastru, čo pre veľké výpočty stojí nemalé sumy. Pre príklad, od začiatku práce až po koniec bolo doteraz spotrebovaných 293 388 jadro-hodín dokopy na oboch klastroch.

Vo vzdialenej dobe, by sa na zlepšenie dalo pozrieť z pohľadu predikcie, kde by sa jednotlivé aproximátory mohli v plánovači váhovať podľa histórie predikcii, ich presnosti a následne určovať ich dôveryhodnosti. Taktiež, by sa mohol plánovač poskytovať ako SaaS (Software-as-a-Service).

Z práce som si odniesol veľa znalostí z pohľadu náročných výpočtov a programovania. Taktiež prácu s superpočítačom, klastermi a plánovačom. A hlavne som spoznal nové metódy pre získavanie dát a zlepšil si kritické myslenie z pohľadu na veľké množstvo dát.

Literatúra

[1] Akima, H. A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. *J. ACM*. First. New York, NY, USA: ACM. október 1970, zv. 17, č. 4, s. 589–602. DOI: 10.1145/321607.321609. ISSN 0004-5411. Dostupné z: http://dx.doi.org/10.1145/321607.321609.

[2] Center, I. N. S. *The infrastructure of IT4Innovations*. VŠB – Technical University of Ostrava, 2017.

[3] Fajmon, B., Hlavičková, I., Novák, M. a Vítovec, J. *Numerická matematika a pravděpodobnost*. First. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ ÚSTAV MATEMATIKY, 2014.

[4] Fritsch, F. N. a Carlson, R. E. Monotone Piecewise Cubic Interpolation. *SIAM Journal on Numerical Analysis*. First. 1980, zv. 17, č. 2, s. 238–246. DOI: 10.1137/0717021. Dostupné z: https://doi.org/10.1137/0717021.

[5] Li, B. a Lu, P. The Evolution of Supercomputer Architecture: A Historical Perspective. In: Xu, W., Xiao, L., Li, J. a Zhang, C., ed. *Computer Engineering and Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, s. 145–153. ISBN 978-3-662-49283-3.

[6] London, Q. M. U. of. *HPC Introduction* [online]. 2019 [cit. 05.01.2020]. Dostupné z: https://docs.hpc.qmul.ac.uk/intro/.

[7] SAS: Analytics, B. I. a Management, D. *SAS High-Performance Analytics tip #1: How it differs from SAS Grid & SAS In-Memory Analytics* [online]. 2016. 2016 [cit. 05.01.2020]. Dostupné z: https://communities.sas.com/t5/SAS-Communities-Library/SAS-High-Performance-Analytics-tip-1-How-it-differs-from-SAS/ta-p/244538.

[8] University, I. *Understand measures of supercomputer performance and storage system capacity* [online]. 2020. 01.02.2020 [cit. 10.02.2020]. Dostupné z: https://kb.iu.edu/d/apeq#measure-flops.

[9] Works, P. *PBS Programmer's Guide*. Altair Engineering, júl 2015.

[10] Yoo, A. B., Jette, M. A. a Grondona, M. SLURM: Simple Linux Utility for Resource Management. In: Feitelson, D., Rudolph, L. a Schwiegelshohn, U., ed. *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, s. 44–60. ISBN 978-3-540-39727-4.

Príloha A

Obsah DVD

Priložené DVD obsahuje nasledujúce súbory:

- `xsasak01.pdf` - text práce
- `scheduler_src` - priečinok obsahujúci zdrojové kódy plánovača a všetky vypracované grafy
- `text_src` - zdrojové kódy textu práce v jazyku LaTeX
- `scheduler_src/README` - textový súbor formátu `Markdown`, obsahujúci návod na spustenie plánovača