POPIS PROJEKTU

Cílem projektu bylo vytvořit aplikaci, která dokáže identifikovat *k* nejlepších procesorů podle zadaných parametrů a zadané agregační funkce pomocí algoritmu threshold. Aplikace také obsahuje naivní implementaci top-k algoritmu. Oba algoritmy jsem rozšířil o možnost řazení parametru vzestupně či sestupně.

ZPŮSOB ŘEŠENÍ

Naivní řešení top-k algoritmu spočítá hodnotu agregační funkce pro každý procesor. Tyto hodnoty následně seřadí a vrátí *k* nejlepších.

Top-k threshold algoritmus funguje v následujících krocích:

1. Pro každý zadaný parametr vytvořím seřazený seznam podle daného parametru.
2. Ze seřazených seznamů paralelně přečtu *k* nových procesorů a vypočítám jejich hodnotu agregační funkce.
3. Uložím si *k* nejlepších procesorů.
4. Vypočítám threshold jakožto maximální možnou hodnotu agregační funkce ještě nepřečtených procesorů.
5. Pokud prvních *k* uložených procesorů má hodnotu agregační funkce větší než je threshold, ukončím algoritmus a vracím těchto *k* procesorů. Jinak jdu zpátky na 2. krok.

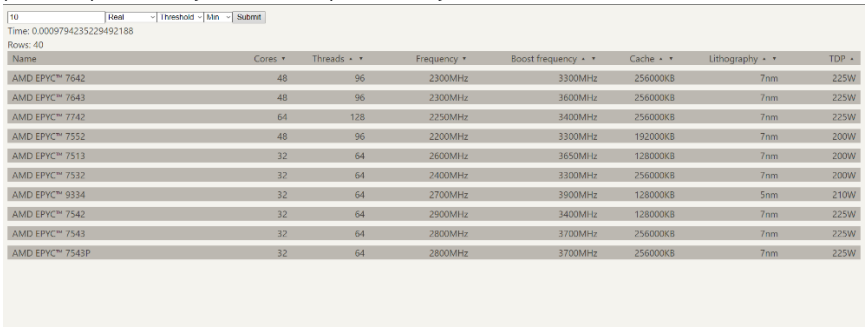
Pro jednoduchou manipulaci s daty ukládám procesory do minimové haldy.

IMPLEMENTACE

Backend aplikace je implementován v jazyce Python pomocí frameworku Django a Django REST. Pro perzistenci dat používám relační databázi Postgresql. Frontend je napsaný v Typescriptu s použitím knihovny React. Sestavení aplikace provádím pomocí Dockeru. Pro spuštění tak stačí pouze zadat příkaz *docker-compose up* a aplikace se spustí na adrese http://localhost:3000.

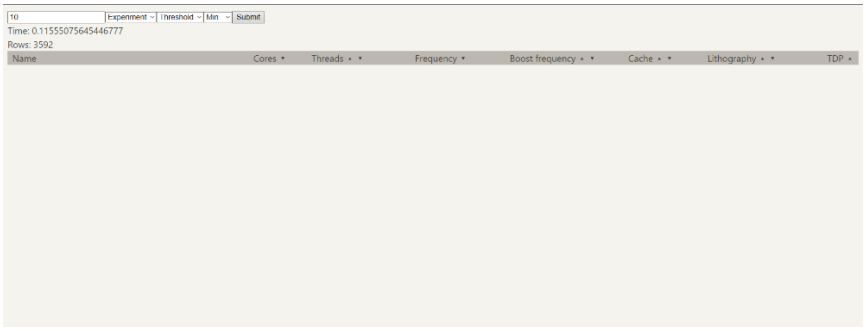
PŘÍKLAD VÝSTUPU

Uživatel si přál najít 10 nejlepších procesorů. Použil k tomu algoritmus threshold a agregační funkci minimum. Procesory chtěl mít seřazené sestupně podle počtu jader a frekvence a vzestupně podle TDP. Aplikace ukáže nalezené procesory včetně času běhu a počtu přečtených řádků potřebných k identifikaci.


10 Real Threshold Min Submit
Time: 0.000979423522949218
Rows: 40

Name	Cores	Threads	Frequency	Boost frequency	Cache	Lithography	TDP
AMD EPYC™ 7642	48	96	2300MHz	3300MHz	256000KB	7nm	225W
AMD EPYC™ 7643	48	96	2300MHz	3600MHz	256000KB	7nm	225W
AMD EPYC™ 7742	64	128	2250MHz	3400MHz	256000KB	7nm	225W
AMD EPYC™ 7552	48	96	2200MHz	3300MHz	192000KB	7nm	200W
AMD EPYC™ 7513	32	64	2600MHz	3650MHz	128000KB	7nm	200W
AMD EPYC™ 7532	32	64	2400MHz	3300MHz	256000KB	7nm	200W
AMD EPYC™ 9334	32	64	2700MHz	3900MHz	128000KB	5nm	210W
AMD EPYC™ 7542	32	64	2900MHz	3400MHz	128000KB	7nm	225W
AMD EPYC™ 7543	32	64	2800MHz	3700MHz	256000KB	7nm	225W
AMD EPYC™ 7543P	32	64	2800MHz	3700MHz	256000KB	7nm	225W

Pokud chceme s algoritmy experimentovat, stačí si přepnout na datovou sadu experiment. V tomto režimu se vypisuje pouze čas a počet řádků.

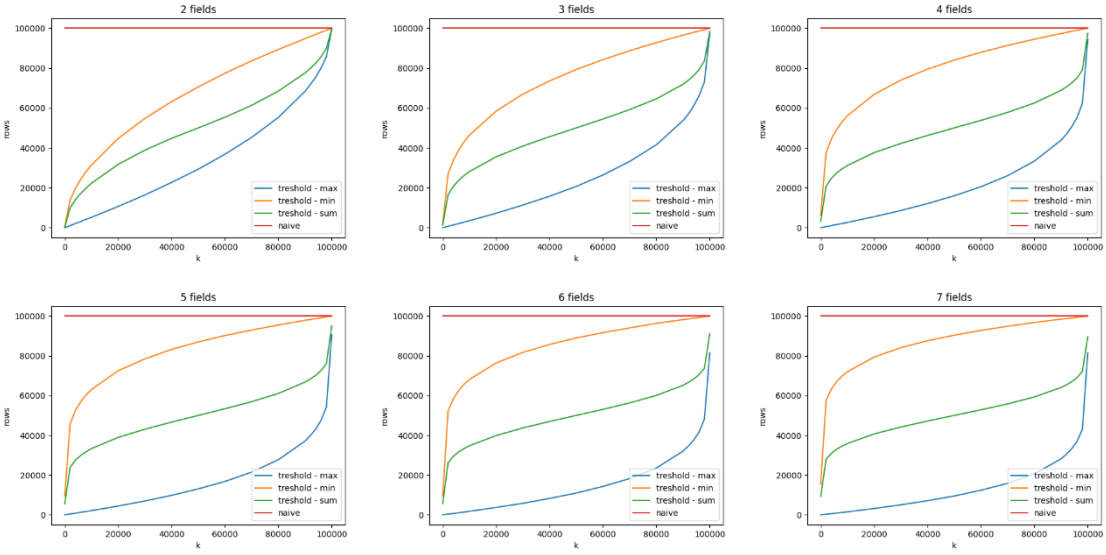


EXPERIMENTÁLNÍ SEKCE

Algoritmy jsem se pokusil porovnat podle počtu potřebných řádků a doby běhu v závislosti na velikosti *k*. Při porovnávání jsem také sledoval jejich chovaní závislé na použité agregační funkci a počtu zvolených parametrů. Experiment probíhal nad databází o 100 000 procesorech s náhodně vygenerovanými parametry.

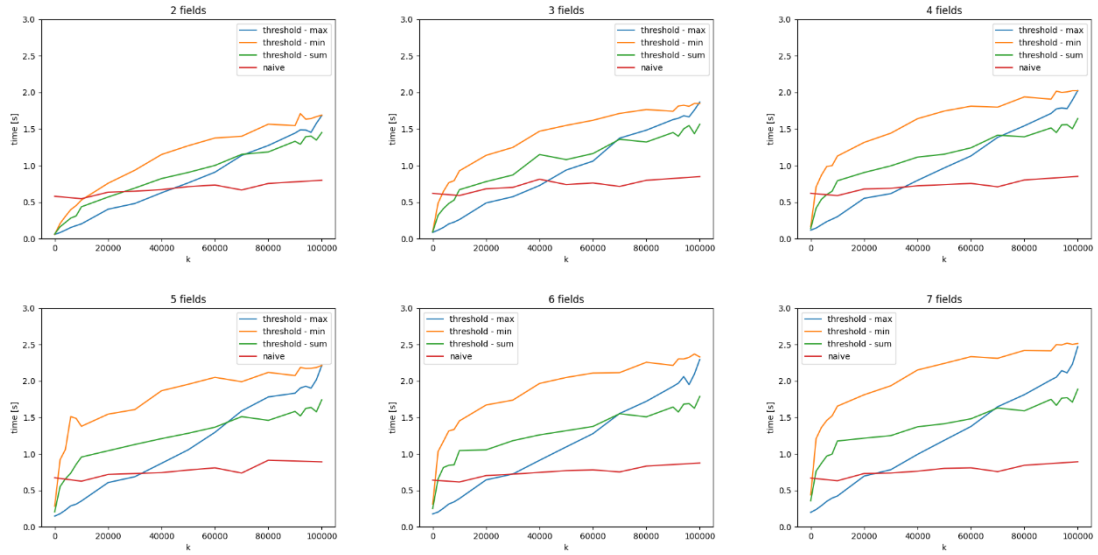
POROVNÁNÍ POČTU PŘEČTENÝCH ŘÁDKŮ

Můžeme si všimnout, že naivní implementace přečte vždy všechny řádky. To z povahy dané implementace není překvapivé. Zajímavé ovšem je chování algoritmu threshold v závislosti na agregační funkci. Zatímco počet přečtených řádků pro agregační funkci minimum roste logaritmicky, průběh agregační funkce maximum je exponenciální. Také jde vidět zvětšující se zakřivení všech průběhů při označení více parametrů. Zjistili jsme tedy, že počet řádků potřebných k identifikaci za pomocí algoritmu threshold je silně závislý na typu agregační funkce a počtu označených parametrů. Dále jsme zjistili, že nejméně řádků se přečte při hledání pomocí maxima a nejvíce za pomoci minima.



POROVNÁNÍ DOBY BĚHU

Přestože algoritmus threshold potřebuje k identifikaci nejlepších objektů mnohem méně řádků než naivní implementace, je patrné, že rychlejší je pouze při nízké selektivitě (malé *k*). Tato hranice se navíc s množstvím vybraných parametrů snižuje. To je zapříčiněno vytvářením seřazených seznamů pro každý parametr a udržováním objektů v haldě. Algoritmus threshold proto není vhodný pro nalezení velkého množství objektů řazených podle mnoha parametrů.



DISKUZE

Projekt jsem se snažil vytvořit co nejrobustněji, tedy tak, jak by se nejspíše implementoval v praxi (rozdělení na backend/frontend, REST API, dedikovaná databáze). I přesto zde existují některé nedostatky, které by bylo potřeba před reálným spuštěním odstranit. Vytváření seřazených seznamů pro jednotlivé parametry při každém zavolání je značně neefektivní a dost se to odráží na výsledné časové složitosti algoritmu. Při reálném nasazení bychom si v tomto případě vytvořili databázové indexy pro každý parametr a nemuseli je tak opakovaně řadit uvnitř algoritmu.

ZÁVĚR

Toto téma mě velmi bavilo, a to jak samotné implementování algoritmů, tak i UI projektu. Dále mě zaujala následná experimentální sekce a porovnávání algoritmů navzájem. Překvapilo mě rozdílné chování algoritmu threshold při různých agregačních funkcích. Přestože jsem v tomto projektu používal framework Django poprvé, vše se s ním dělalo velice snadně a rozhodně bych ho pro podobné projekty zvolil znova.