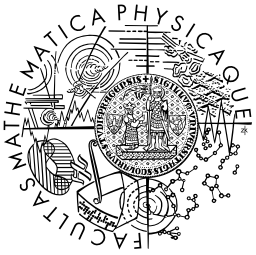


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Dominik Hrdý

PerfEval: Spojení unit testů s vyhodnocováním výkonu

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. Ing. Petr Tůma, Dr.

Studijní program: Informatika

Studijní obor: Systémové programování

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: PerfEval: Spojení unit testů s vyhodnocováním výkonu

Autor: Dominik Hrdý

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. Ing. Petr Tůma, Dr., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Abstrakt.

Klíčová slova: testování výkonnost

Title: PerfEval: Marrying unit testing with performance evaluation

Author: Dominik Hrdý

Department: Department of Distributed and Dependable Systems

Supervisor: prof. Ing. Petr Tůma, Dr., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: testing performance

Obsah

Úvod

Po více než dvacet let pomáhají unit testy udržovat kvalitu kódu v průběhu vývoje softwaru. Za tuto dobu bylo vyvinuto spoustu knihoven pro implementaci a spouštění unit testů. Verzovací nástroje jako GitLab, nebo GitHub umožňují v rámci vývoje software verzovat, spouštět unit testy a reagovat na jejich případná selhání.

Pro udržování korektnosti kódu využíváme pokrytí unit testy. Udržování korektnosti je dnes již naprosto běžnou součástí vývoje softwaru. Psaní výkonnostních testů ovšem tak běžné není. Práce s frameworky pro měření výkonu totiž není tak jednoduchá jako používání knihoven pro psaní unit testů, a proto jejich užití není tak časté.

Frameworky pro vyhodnocování výkonu softwaru jsou podobné knihovnám pro psaní unit testů. Frameworky jako JMH nebo BenchmarkDotNet pomáhají implementovat výkonnostní testy. Výkonnostní testy se implementují obdobně jako u unit testů obvykle pomocí anotací. Vyhodnocování naměřených výsledků dnes obvykle zahrnuje i ruční zhodnocení naměřených výsledků. Vyhodnocování výkonu totiž vyžaduje porovnat naměřený výkon s nějakou další referenční hodnotou. Měřící frameworky tyto referenční hodnoty nemají a ani je nikde získat nemohou, protože měří pouze jednu verzi softwaru. Nemohou tedy dělat zmíněné celkové vyhodnocení.

Z porovnání vyhodnocování unit testů a výkonnostních testů, není vyhodnocování výkonnostních testů tak jednoduché. Unit testy testují korektnost. Korektnost se prokáže tak, že na každý zadaný vstup program vrátí očekávaný výstup. Při vyhodnocování výkonu se změří nějaká data o běhu programu. Tato data ale sama o sobě nic neříkají a je nutné je zkoumat v kontextu.

Výsledkem této práce je nástroj nazvaný PerfEval. PerfEval je konzolová aplikace napsaná v programovacím jazyce Java. Perfeval umí pomocí argumentů na příkazové řádce vyhodnocovat výsledky výkonnostních testů.

PerfEval je schopen automatického hlášení výsledků výkonnostních testů. Umí porovnávat výsledky měření výkonu dvou verzí softwaru mezi sebou. PerfEval je také vhodný pro skriptování, protože o výsledcích informuje nejen výpisem na standardní výstup, ale také exit kódem. Nástroj podporuje zpracování výstupů frameworků BenchmarkDotNet a JMH ve formátu JSON, ale je rozšiřitelný i pro zpracování jiných frameworků, nebo formátů.

1. Kontext

1.1 Testování softwaru při vývoji

Při vývoji softwaru je vhodné vyvíjený software neustále testovat. Software lze testovat více způsoby, ale cílem testů je vždy otestovat některý z důležitých kvalitativních atributů jako je například korektnost, nebo výkon.

Pro testování korektnosti se obvykle používají unit testy. Jedná se většinou o krátké testovací funkce, které kontrolují jestli se testovaný kód chová požadovaným způsobem. Zjišťují tedy jestli kód na zadaný vstup vrátí očekávaný výstup. Ke kódu který není aktuálně vyvíjen, se obvykle nemění ani unit testy, a proto umožňují udržovat neustálou korektnost již hotového kódu. Zda-li je kód korektní se pomocí unit testů zjistí velmi jednoduše. Kód je korektní pokud vrátil očekávaný výstup.

Testování výkonu obvykle probíhá tak, že se použije nějaký vhodný měřící framework. Obvyklé frameworky pro měření výkonu mají vlastní pravidla, jak se mají oanotovat metody, které se mají měřit. Tyto frameworky umožňují měřit výkon softwaru podle různých metrik. Mezi tyto metriky se řadí čas, propustnost a například spotřeba paměti. Výsledky měření frameworky umí obvykle zaznamenat jak do strojově čitelného formátu, tak do formátu čitelného pro člověka. Výsledek měření je ale pouze sada čísel, kde jsou ke jménům testovaných metod přiřazeny naměřené hodnoty.

TODO: doplnit obrázek nějakého výstupu například BenchmarkDotNet, nebo JMH? možná by se hodilo nějak vhodně zasazený přímo textový výstup

Výsledky testování výkonu se tedy musejí vyhodnocovat tak, že se podrobně prozkoumá výsledná sada čísel. Oproti testování korektnosti, kdy test projde, nebo neprojde je testování výkonu výrazně složitější. Pohledem na samostatnou sadu dat se nedá určit zda-li je software dostatečně rychlý. Aby bylo možné ze sady určit něco vypovídajícího mohl by být stanoven limit výkonnosti. Například by se stanovilo, že metoda se nebude konat déle než 5 s. Tento přístup nemusí být vypovídající při dlouhodobém vývoji a při časech hluboko pod limitem. Proto je vhodné aby se datové sady, které testovací framework produkuje, testovaly navzájem mezi sebou. Porovnáním sad je totiž možné zjistit, jestli nedošlo k významným změnám výkonu při vývoji od předchozí verze. Frameworky samotné však tuto možnost obvykle nemají.

1.2 Průběžná integrace

Při vývoji softwaru se čato používají nástroje pro automatizování některých činností. Obvykle se automatizují činnosti jako jsou správa verzí, spouštění testů a jejich vyhodnocování.

Pro správu verzí při vývoji softwaru se obvykle používá technologie git. Jedná se o sytém pro správu verzí. Git je vhodný i pro práci ve velkých týmech. Umožňuje totiž členění projektu do větví. Každá větev je vhodná pro vývoj samostné části aplikace a následným spojováním větví pak dochází k propojení funkčností vyvinutých v jednotlivých větvích. Technologie si formou pamatování si změn udržuje přehled o průběžných verzích a umožňuje uživatelům (programátorům)

vrátit se vývoji zpět. Technologii je možné ovládat jednoduchými příkazy z příkazové řádky. Je tedy vhodná pro psaní krátkých skriptů.

Nástroj git umí jednoduše zpostředkovat průběžnou integraci (continuous integration). Průběžná integrace umožňuje dělat uživateli automatizované kroky při nahrávání nových verzí do větve nebo při spojování větví. Při průběžné integraci tedy jde o spouštění skriptu při některé ze zmíněných událostí. V rámci průběžné integrace je možné testovat software pomocí unit testů i benchmarků pro měření výkonu. Dále je možné použít jakékoli jiné příkazy příkazové řádky. Do průběžné integrace je tedy možné jednoduše zapojit téměř jakoukoli konzolovou aplikaci.

1.3 Scénář

Mějme programátora, který se rozhodne vyvíjet nový software. Projekt se mu začne rozrůstat, a tak aby mohl udržovat větší projekt napíše si unit testy. Pomocí unit testů si udržuje průběžnou korektnost již hotového kódu. Dále pojme podezření, že se jeho stávající kód začíná zpomalovat (klesá výkon). Proto se rozhodne použít framework pro testování výkonu a spustí testy pro měření výkonu svého kódu.

Programátor změří výkon na verzi před tím než pojmul podezření o zpomalování. Následně změří výkon aktuální verze. Výsledkem těchto měření jsou dvě tabulky s daty. V každé z nich je jméno testovací metody k níž je přiřazen výsledek měření. Protože ale benchmarkovací framework funguje jako stopky, tak není schopný porovnat dvě tabulky mezi sebou. Analogie ke stopkám je, že jsou schopny změřit čas prvního a druhého kola zvlášť, ale nejsou schopny je porovnat.

Programátor se tedy musí sám podívat, jestli výkon jeho softwaru klesá. Programátor by tedy potřeboval nástroj, který by uměl vzít výstupy měření z různých verzí a porovnal je mezi sebou. Zároveň by programátor jistě chtěl, aby byl na zhoršení výkonu upozorněn podobným způsobem jako na selhání unit testů tj. selháním průběžné integrace.

TODO: dodat příklad skriptu průběžné integrace

2. Problém vyhodnocování výkonu

2.1 Automatické vyhodnocování

V minulé kapitole byl zmíněn problém programátora s testováním výkonu. Programátor má unit testy, které testují korektnost jeho softwaru. Umí je efektivně spouštět s každou změnou pomocí průběžné integrace. Chtěl by, aby mohl podobně efektivně testovat i výkon svého softwaru.

S testováním výkonu je ale problém. Když se spustí měření výkonu, tak výsledkem je pouhá datová sada. Tato datová sada nevypovídá nic o změně průběžného výkonu, která je zajímavá. Právě podle změny ve výkonu softwaru je možné zjistit, jestli je nutné kód optimalizovat, protože dochází k významným zhoršením.

Při samotném měření výkonu je ale možné, že se některé hodnoty výrazně odchýlí od ostatních. Může tedy být nutné měření opakovat a z více naměřených hodnot dohromady usuzovat, jak se výkon v průběhu vývoje mění. K analýze takto získaných hodnot se ale hodí využít statistických metod, které si i se zmíněnými odchylkami umí poradit.

Vyvinutý systém PerfEval řeší programátorův problém. Jedná se o nástroj, který může zakomponovat do své průběžné integrace tak, aby byl pokles výkonu hlášen. Nástroj při spuštění průběžné integrace porovná dvě poslední verze a případně oznámí zhoršení výkonu. Na základě tohoto hlášení může selhat celá průběžná integrace, což je obdobné chování, jako se očekává při selhání unit testů.

2.2 Výsledky měření výkonu

Z počátku je dobré se zamyslet nad tím, jak můžou výsledky měření výkonu softwaru vypadat. První domněnky o tom, že stačí měřit pouze čas se ukázaly jako mylné. Testovací frameworky totiž umožňují mimo měření času také měření frekvence nebo například spotřeby paměti. Předpokládat se tedy dá jen to, že pokud vezmu dva výsledky měření výkonu ze dvou různých verzí, tak budou reprezentovány stejnou fyzikální veličinou a v lepším případě budou mít i stejnou fyzikální jednotku.

Protože testovací frameworky nepoužívají žádné identifikátory testů, tak je nejpřímější řešení jejich rozpoznávání používat jména testovacích metod jako identifikátor. Tato jména poskytují ve výsledcích měření jak framework BenchmarkDotNet, tak framework JMH. Z dokumentace frameworku Criterion (Heisler, 2024) pro měření výkonu v jazyce R se název metody ve výsledcích nachází také. Z toho lze usoudit, že použití jména metody jako identifikátoru může být dostatečně obecné.

V případě měření výkonu u jazyků, které jsou kompilované metodu JIT je nutné být obezřetný. Je nutné všímat si, jaká data byla naměřena. Jazyky kompilované metodou JIT mohou při měření podléhat tzv. zahřívací fázi. Jedná se o fázi, kdy kód ještě není plně optimalizovaný, ale již se provádí a může být měřen. V závislosti na použitém měřícím frameworku je pak nutné naměřená

data vhodně filtrovat. V případě, že by se data před a po optimalizaci nacházela v jednom statistickém souboru by mohla být znehodnocena.

2.2.1 Výstup měření BenchmarkDotNet

BenchmarkDotNet je framework určený k měření výkonu programů na platformě .NET. V důsledku toho, že měří programy na platformě .NET je schopen měřit výkon programů napsaných v programovacích jazycích C#, F# a Visual Basic. Podrobnosti o tomto měřícím frameworku je možné nalézt v dokumentaci (Akinshin, 2024).

Při měření výkonu pomocí BenchmarkDotNet se měřené hodnoty vypisují na standardní výstup včetně konečného shrnutí. Mimo standardní výstup se ještě výsledky měření ukládají do strojově zpracovatelných formátu jako je například JSON, nebo CSV.

(DODAT PŘÍLOHU)

Měření výkonu programu v jazyce C# pomocí frameworku BenchmarkDotNet má za výstup již statisticky zpracované hodnoty. Aby bylo možné sledovat jednotlivé naměřené hodnoty je nutné zvolit již při psaní testů správný exportér, který tuto funkci podporuje. Dále je naměřené hodnoty filtrovat, protože C# je kompilovaný metodou JIT.

BenchmarkDotNet je možné nakonfigurovat tak, aby se v souboru s výsledky nacházely podrobnosti o prostředí jako je operační systém, verze platformy .NET, jméno, typ a parametry procesoru. Dále aby se v souboru s výsledky nacházely výsledky jednotlivých provedených měření. Výsledek měření u sebe má informaci o jméně testovací metody, zpracované statistické údaje a naměřené hodnoty z různých módů a iterací měření.

TODO: přidat jako přílohu výstupy BenchmarkDotNet odkazovat se do nich a do dokumentace BDN

2.2.2 Výstup měření JMH

JMH je framework pro měření výkonu, který umožňuje pomocí anotací definovat výkonostní testy pro programy v jazyce Java. Z průzkumu (Stefan a kol., 2017) vyplývá, že se jedná o nejpoužívanější framework pro měření výkonu pro projekty vyvíjené v jazyce Java.

JMH obdobně jako BenchmarkDotNet poskytuje výsledek měření jako tabulku na standardní výstup. Dále poskytuje výstup také v podobě strojově zpracovatelných formátů jako jsou například XML, nebo JSON. O výstup v této podobě je nutné zažádat pomocí argumentů na příkazové řádce při spouštění měření. Další podrobnosti o frameworku JMH je možné nalézt v dokumentaci (Shipilëv, 2024).

(DODAT PŘÍLOHU)

Ve výstupním souboru měření pomocí JMH lze nalézt informace o stroji na kterém probíhalo měření. Jedná se především o název stroje a verzi operačního systému. Dále zde lze nalézt verzi Javy, ve které probíhalo měření. V souboru je možné vidět také ostatní parametry měření, jako je zahřívací doba a počet zahřívacích iterací. Zahřívací iterace jsou zde uvedeny, protože Java je stejně jako C# jazyk kompilovaný metodou JIT. Je tedy nutné vástupní data vhodně filtrovat.

Jednotlivé naměřené hodnoty jsou ve výstupním souboru dostupné i ve výchozím nastavení JMH. Naměřené hodnoty jsou reprezentovány pomocí metriky. V rámci metriky jsou uloženy informace o fyzikální jednotce, která se měřila, a sady naměřených hodnot. Pro každý běh (fork), přičemž běhů může být v jednom spuštění JMH více, se objeví jedna sada měřených hodnot u metriky.

2.3 Použití statistických metod pro analýzu dat

Pro vyhodnování výkonu je využito metod testování hypotéz. Ve statistickém testování hypotéz se snažíme zamítnout nulovou hypotézu. V případě zamítnutí nulové hypotézy se předpokládá, že platí alternativní hypotéza. Popis testování hypotéz v této kapitole se řídí skripty Pravděpodobnost a statistika 1 (Šámal, 2023).

Při testování hypotéz rozeznáváme chyby I. a II. druhu. Chyba I. druhu znamená, že jsme nulovou hypotézu zamítli, i když platí. Chyba II. druhu znamená, že jsme ji nezamítli, ale ona neplatí.

V našem případě porovnávání výkonu bude nulová hypotéza tvrzení, že výkon obou verzí je stejný. Jako alternativní hypotézu budeme uvažovat, že výkony obou verzí jsou různé. Naměřené hodnoty jedné testovací metody každé z verzí budeme považovat za spojitou náhodnou veličinu. Pokud hodnoty těchto veličin mají stejné rozdělení, tak budeme tvrdit, že i výkon obou verzí je stejný.

Chyba I. druhu tedy v našem případě znamená, že jsme prohlásili, že výkony dvou verzí jsou různé, ačkoli jsou stejné. Pravděpodobnost chyby I. druhu je obvyklý parametr statistického testu. Pravděpodobnost chyby I. druhu bude dále značen jako parametr α. Parametr α je součástí konfigurace systému PerfEval.

2.3.1 Welchův dvouvýběrový T-test

Welchův dvouvýběrový t-test se používá jako statitistika při testování hypotéz. Tato statitistika předpokládá, že náhodné veličiny jsou nezávislé a jejich rozdělení se blíží normálnímu rozdělení. Nezávislost náhodných veličin je dána vlastnostmi experimentu (Turčicová, 2021) a její zajištění je mimo doménu řešeného problému. PerfEval tedy v případě použití možnosti T-test možnou závislost zanedbává.

TODO: není náhodou lepší místo CLV použít silný zákon velkých čísel

Normalitu náhodných veličin je možné zajistit díky centrální limitní větě (CLV). Se zajištěním normality nám pomůže samotná struktura naměřených výsledků. Výsledky měření obsahují běhy. Běhy obsahují jednotlivé naměřené hodnoty. Naměřené hodnoty představují vzorky náhodné veličiny. Pokud budeme v rámci T-testu namísto naměřených hodnot uvažovat průměry jednotlivých běhů, tak se podle CLV bude rozdělení těchto průměrů blížit normálnímu rozdělení.

Dvouvýběrový T-test je dnes běžnou součástí standardních matematických knihoven. Použití T-testu v programu je tedy velmi jednoduché, protože není potřeba zamýšlet se nad korektností psaného kódu. V případě systému PerfEval se T-test provádí tak, že se z naměřených hodnot vyrobí Studentovo T-rozdělení. Z tohoto rozdělení se zjistí intervalový odhad střední hodnoty.

Nakonec se již jen zkoumá zda-li tento interval obsahuje nulu ($\delta = 0$). Pokud interval nulu neobsahuje, pak lze tvrdit, že s pravděpodobností $1 - \alpha$ nulová hypotéza neplatí.

2.3.2 Percentilový hierarchický bootstrap

Bootstrap je statistická metoda využívající tzv. resamplování. Stejně jako u dvouvýběrového T-testu se předpokládá nezávislost náhodných veličin. Podmínka nezávislosti bude v systému zanedbána, protože ji není možné zaručit. Bootstrap se jako metoda používá v případě, kdy o náhodných veličinách není možné určit téměř žádné silné předpoklady. Díky této vlastnosti je bootstrap pro testování hypotéz o výkonu verzí softwaru použit.

Algorithm 1: Bootstrap1D

```
Input: measurements, iterationCount
Output: bootstrapppedSamples
n = length(measurements);
samples = [];
for i = 0; i < iterationCount; i += 1 do
    sum = 0;
    for j = 0; j < n; j = j + 1 do
        index = random() mod n;
        sum += measurements[index];
    samples.add(sum/n);
return samples;
```

Interval spolehlivosti se z výstupu výše uvedeného algoritmu určuje velmi snadno. Podle percentilového bootstrapu se interval spolehlivosti nalezne tak, že se vezme $\frac{\alpha}{2}$-tý a $\frac{1-\alpha}{2}$-tý percentil z resamplovaného souboru. Tyto dvě hodnoty budou představovat hranice intervalu.

Zkoumanou náhodnou veličinou je rozdíl dvou náhodných veličin. Tyto dvě náhodné veličiny jsou dány měřením výkonnosti dvou verzí softwaru. Nulová hypotéza, která je vyvracena, tvrdí, že obě veličiny mají stejné rozdělení. Pokud tedy interval spolehlivosti neobsahuje nulu, tak můžeme nulovou hypotézu vyvrátit, protože s pravděpodobností $1 - \alpha$ neplatí.

Naměřené vzorky však nejsou prostý statistický soubor. Jedná se o hierarchický soubor dat. Každé jedno měření se skládá z jednoho, nebo více běhů. Každý běh se skládá z jednoho, nebo více naměřených údajů. Vytváření bootstrapového statistického souboru tedy vypadá trochu odlišně.

Algoritmus výše ukazuje, jak vypadá výběr nového statistického souboru. Vyberou se náhodné běhy z jednotlivých výsledků měření. Z těchto běhů se získá 1D bootstrap. Novým prvkem vytvářeného statistického souboru se stane rozdíl středních hodnot těchto bootstrapů.

2.3.3 Co dělat v případě nevyvrácení hypotézy?

V případě nevyvrácení nulové hypotézy nám statistické testy nedávají žádnou informaci. Nicméně je stále nutné se rozhodnout zda-li nulová hypotéza platí. Je

Algorithm 2: Bootstrap2D

```
Input: runs1, runs2, iterationCount
Output: bootstrapppedSamples
n = length(runs1);
m = length(runs2);
samples = [];
for i = 0; i < iterationCount; i += 1 do
    index = random() mod n;
    samples1 = Bootstrap1D(runs1[index], iterationCount);
    index = random() mod m;
    samples2 = Bootstrap1D(runs2[index], iterationCount);
    diff = mean(samples1)-mean(samples2);
    samples.add(diff);
return samples;
```

nutné se ale stále vyvarovat chybě II. druhu. Proto v tomto případě budeme považovat nulovou hypotézu za platnou, pokud bude interval spolehlivosti dostatečně úzký. Pokud má tedy interval spolehlivosti dolní mez 1 a horní mez 3, pak je jeho délka 2. Odhadovaný průměr by byl také 2. Relativní úzkost intervalu by tedy byl poměr průměru a délky, tedy 1.

V případě většího množství vzorků je možné zužovat interval spolehlivosti. Vztah mezi úzkostí a počtem vzorků odpovídá $O(\frac{1}{\sqrt{n}})$, kde n je počet vzorků. Z daného množství vzorků je tedy možné odhadnout, kolik vzorků je ještě zapotřebí změřit.

V případě, že je interval dostatečně úzký prohlásíme, že nulová hypotéza platí. V případě, že interval není dostatečně úzký, prohlásíme, že vzorků není dost.

PerfEval tedy v konečném důsledku rozlišuje tři základní výsledky porovnání výkonu verzí. Test končí s kladným výsledkem pokud platí nulová hypotéze dle kritérií výše. Test končí s kladným výsledkem také když platí je nulová hypotéza vyvrácena, ale výkon novější verze je lepší. V ostatních případech test neprojde. Další dva možné výsledky testu tedy budou značit selhání. Druhý výsledek testu značí, že nulová hypotéza neplatí, a zároveň že došlo ke zhoršení výkonu. Třetí výsledek značí, že se nulovou hypotézu nepodařilo vyvrátit, ale počet naměřených výsledků je příliš malý.

3. Systém PerfEval

3.1 Popis systému

Systém PerfEval je konzolová aplikace, jejíž činnost je ovládaná příkazy a parametry na příkazové řádce. Příkazy umožňují inicializovat systém, přidávat nové výsledky měření a vyhodnocovat mezi sebou výsledky měření dvou verzí.

3.1.1 Průběh vyhodnocování

Hlavní funkcionalitu pro vyhodnocování poskytuje třída EvaluateCLICommand. Celý úkol vyhodnocování výkonu byl rozdělen do tří oddělených kroků.

Jako první dojde k naparsování výsledků měření dvou porovnávaných verzí. Parsování probíhá pomocí implementace rozhraní MeasurementParser, který umí naparsovat potřebný formát výsledků měření, které se budou porovnávat. Výsledkem parsování vzniknou dvě instance třídy Samples, která reprezentuje vzorky výsledků měření.

Následně se tyto dvě instance porovnají pomocí metody compareSets na třídě PerformanceEvaluator. Instance třídy PerformanceEvaluator k porovnání dvou instancí Samples využívá implementace rozhraní StatisticTest. Konkrétně tato implementace rozhoduje o tom, jaký statistický test se při porovnávání vzorků použije.

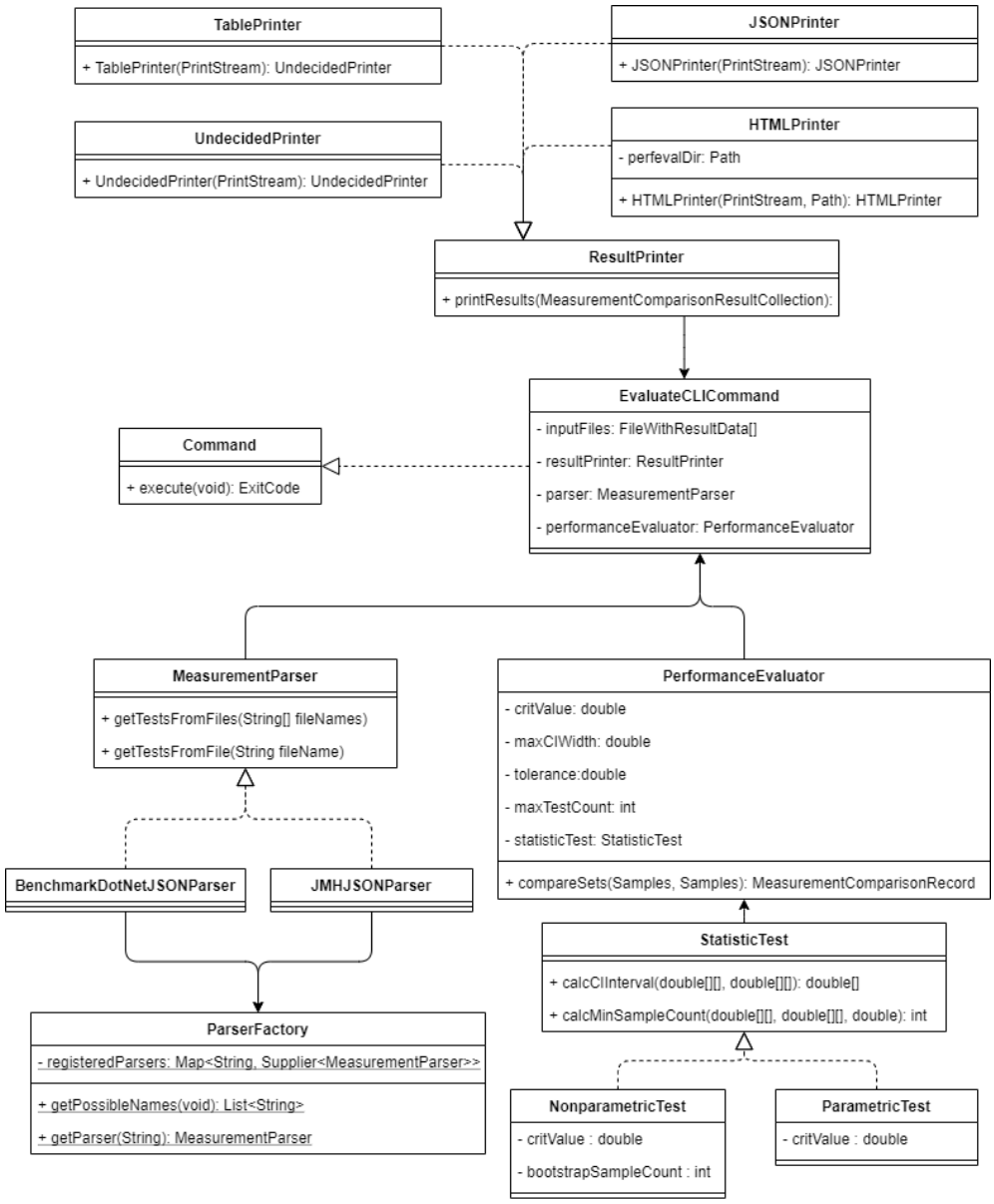
Statistický test, který implememtuje rozhraní StatisticTest, musí mít metody calcCIInterval a calcMinSampleCount. Metoda calcCIInterval vrací intervalový odhad rozdílu náhodných veličin, které jsou reprezentovány konkrétními naměřenými vzorky.

Pokud tento interval neobsahuje nulu, pak lze s požadovanou přesností a pravděpodobností usuzovat, že rozdělení náhodných veličin jsou různá. Pokud jsou rozdělení různá, pak se jako kladný výsledek testu připouští zlepšení výkonu, nebo pokud je nový průměr v dané toleranci.

Pokud interval obsahuje nulu, tak je vyhodnocení výsledku testu výkonu složitější. Pokud je interval spolehlivosti dostatečně úzký, tak považujeme rozdělení náhodných veličin za stejná a test výkonu dopadl kladně. Pokud interval spolehlivosti není dostatečně úzký, tak test výkonu neprošel. Následně se pomocí metody calcMinSampleCount určí kolik měření by bylo potřeba, aby byl interval spolehlivosti dostatečně úzký. Spočtený počet vzorků s informací o tom, zda-li je, nebo není možné je změřit, bude přidán k výsledkům porovnávání.

Posledním krokem při vyhodnocování je zavolání metody printResults na implementaci rozhraní ResultPrinter. Tato metoda vypíše instanci MeasurementComparisonResultCollection, která reprezentuje všechny výsledky jednotlivých testů výkonu. Po vypsání výsledků požadovaným způsobem dle dodané implementace ResultPrinter se nastaví správný exit kód. V případě, že všechny testy dopadly kladně, pak bude exit kód 0. V případě, že alespoň v jednom případě došlo ke zhoršení, pak bude exit kód 1. V případě, že dojde k nějaké výjimce, tak exit kód bude větší než 100. Touto výjimkou může být například neexistence některého ze souborů s naměřenými výsledky v důsledku jeho smazání, nebo přesunutí.

Na následujícím obrázku je k vidění architektura části systému právě kolem třídy EvaluateCLICommand, která zajišťuje průběh vyhodnocování testů výkonu.



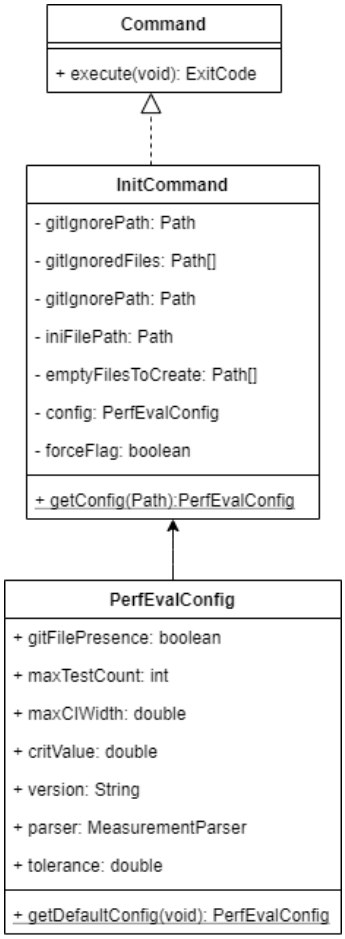
Obrázek 3.1: Objektový návrh časti PerfEvalu pro porovnávání výsledků měření

Hlavní myšlenka návrhu je tok dat programem. Po zpracování příkazové řádky dostane instance EvaluateCLICommand informace o soubourech se kterými bude pracovat. Při konstrukci dostane správné implementace ResultPrinter, MeasurementParser a instanci třídy PerformanceEvaluator se správnou instancí StatisticTest. Data plynou programem tak, že se naparsují pomocí implementace třídy MeasurementParser. Naparsované výsledky se pomocí statistického testu vyhodnotí mezi sebou. Výsledky porovnání se vypíšou pomocí implementace rozhraní ResultPrinter.

3.1.2 Inicializace systému PerfEval

Před použitím systému PerfEval k vyhodnocování je nutné jej inicializovat. Inicializace umí rozpoznat zda-li se nachází v kořenovém adresáři Git repozitáře. Rozpoznávání probíhá tak, že se hledá soubor s názvem .git.

Inicializace probíhá tak, že se zjistí přítomnost .git souboru. Z příkazové řádky se zjistí jaký parser (implementace třídy MeasurementParser) pro naměřené hodnoty se bude používat. Vyrobí výchozí konfigurace. Do výchozí konfigurace se dodají informace o git repozitáři a o parseru. Nakonec se vyrobí složka se v adresáři, kde uživatel program spustil vyrobí složka .performance a v ní soubory .gitignore a config.ini. Soubor config.ini obsahuje konfiguraci systému PerfEval pro jeden projekt jehož výkon mezi verzemi se bude porovnávat.



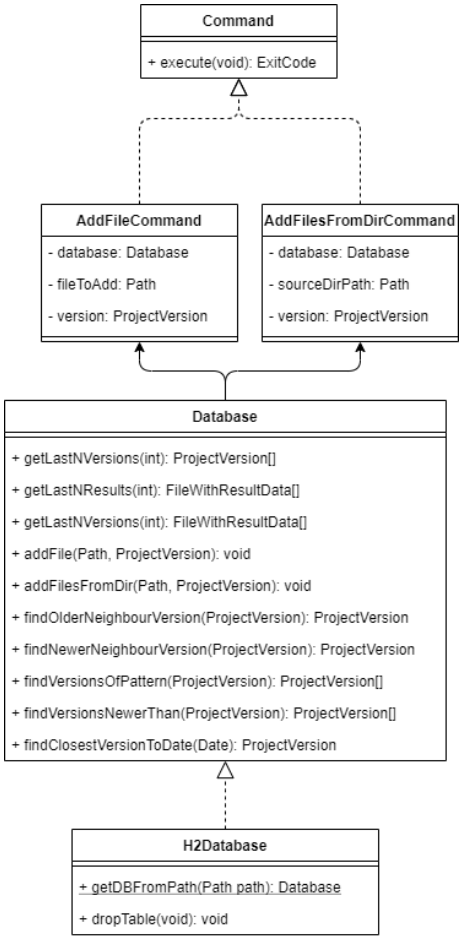
Obrázek 3.2: Objektový návrh časti PerfEvalu pro inicializaci

Z obrázku je patrné, že třída InitCommand úzce spolupracuje s třídou PerfEvalConfig. Třída PerfEvalConfig reprezentuje obsah souboru config.ini. Provedení metody execute na třídě InitCommand provede inicializaci systému PerfEval výše zmíněným způsobem.

3.1.3 Přidávání nových výsledků testů

Uživatel musí každý výsledek měření výkonu zaznamenat do systému PerfEval. Pokud výsledek měření zaznamenán, nebude s ním PerfEval vůbec pracovat.

Přidávat je možné soubory samostatně, nebo z adresáře. Při přidávání souborů z adresáře budou přidány všechny soubory a to i soubory zanořené ve vnitřních adresářích zadaného adresáře.



Obrázek 3.3: Objektový návrh časti PerfEvalu pro přidávání výsledků měření

Na obrázku je vidět, že za oběma příkazy pro přidávání výsledků měření stojí jedna implementace rozhraní Database. Jediná existující implementace tohoto rozhraní využívá technologie H2 databáze. Jedná se o technologii, která umožňuje vést si databázi lokálně v rámci souboru a pokládat na ní klasické SQL dotazy.

Implementace rozhraní Database pomocí technologie H2 je třída H2Database. V rámci databáze pro jeden projekt, který PerfEval spravuje jsou lokální soubory pro H2 databázi uloženy v adresáři .performance. Databáze pro správu dat o výsledcích měření má jen jednu tabulku. V této tabulce jsou informace o cestě k souboru a informace o verzi.

3.2 Rozbor alternativ v řešení

Při vývoji systému PerfEval došlo k několika rozhodnutím, které výrazně ovlivnily jeho fungování a architekturu. Nejdůležitější rozhodnutí, která byla v průběhu výoje provedena jsou v následujících částech podrobně vysvětleny.

3.2.1 Grafické rozhraní

PerfEval je aplikace používaná z příkazové řádky. Od počátku plánování bylo zamýšlené, že se bude ovládat pomocí příkazové řádky. Je určen pro porovnávání výsledků testování výkonu dvou verzí softwaru. Pomocí exit kódu signalizuje, zda-li nedošlo ke zhoršení výkonu.

PerfEval měl mít ale i grafickou část. V grafické části by si mohl uživatel prohlížet dlouhodobý vývoj výkonu jeho testovaného softwaru. K dispozici by mu v rámci tohoto rozhraní mělo být porovnání více verzí v podobě grafů. Zároveň by si uživatel mohl omezovat zobrazovaná období pro detailnější zkoumání.

Protože základní aplikace bez grafického rozhraní byla složitější než zpočátku zdálo, tak grafické rozhraní aplikace nebylo realizováno.

3.2.2 Spouštění testů systémem PerfEval

V počátku vývoje bylo nutné se rozhodnout jakým způsobem bude systém přijímat a zpracovávat výsledky testů. V úvahu přicházela varianta, že uživatel provede měření výkonu sám, a druhá varianta, že uživatel systému vysvětlí jakým způsobem se testování výkonu spouští. Pokud by byla zvolena varianta, kdy PerfEval spouští testy sám, tak by bylo nutné nalézt dostatečně univerzální způsob spouštění testů.

Aplikace a benchmarky pro měření výkonu mohou být jak konzolové, tak grafické aplikace. Pokud by PerfEval měl měření provádět sám, tak by téměř určitě nebyl schopen pracovat s grafickými aplikacemi, ale byl by schopen spouštět programy s parametry na příkazové řádce.

Dále by bylo nutné vysvětlit jak vypadá výstup spouštěných testů. Když pomineme formát, tak je nutné zjistit, kam program, který provádí měření výsledky ukládá. Benchmarkovací systém BenchmarkDotNet například vypisuje výsledky měření v podobě tabulky na standardní výstup a zároveň ukládá strojově čitelné výsledky do speciálního k tomu určeného adresáře.

Nakonec by to vypadalo tak, že při inicializaci systému uživatel zadá jak spustit test výkonu a kam se uloží výsledek. Tímto způsobem by došlo k tomu, že PerfEval by začal určovat, jak mají vypadat programy jejichž výstupy přijímá. Na druhou stranu použití stávajícího řešení tyto nevýhody nemá. Jediné, co musí uživatel zajistit je, že jeho způsob měření zvládne uložit výsledek do souboru, který PerfEval umí zpracovat.

Stávající řešení je tedy složitější v tom, že uživatel musí testování výkonu spouštět sám. Poté, co spustí testování výkonu a dostane soubor s výsledky je přidá do databáze výsledků systému PerfEval a poté spustí PerfEval s příkazem evaluate pro porovnání výsledků testování výkonu. Kdyby se použila varianta s tím, že si PerfEval spouští testy sám, tak by uživatel testování a porovnání výkonu zvládl jedním spuštěním aplikace.

3.2.3 Použité statistické metody

Pro porovnání dvou výsledků měření se používají statistické metody. Statistické metody se používají ke zjištění, jestli výsledky měření považované za náhodné veličiny mají stejné rozdělení, popřípadě jestli je vzorků dostatečné množ-

ství. V systému PerfEval jsou implementovány statistické metody bootstrap a T-test.

Pomocí bootstrapu dochází k iluzi, že vzorků je mnohem více, než je jich ve skutečnosti změřeno. Protože měření vzorků je samo o sobě dlouhotrvající proces, tak se použití této statistické metody vyplatí. Bootstrap má ale tu nevýhodu, že jeho výpočet může oproti prostému zkoumání vzorků trvat delší dobu.

T-test je tedy možné zvolit namísto bootstrapu kvůli tomu, že je rychlejší. Oproti náhodnému vybírání vzorků z dvoudimenzionální sady dat, kterým vzorky odpovídají, dochází k pouhému dosazení hodnot do vzorce.

Uživatel by se tedy při volbě testu měl rozhodovat na základě toho kolik má změřených vzorků a kolik má času na porovnání výsledků měření. Pokud má uživatel naměřených vzorků málo a větší množství času k vyhodnocení doporučuje se použít bootstrap. Pokud má uživatel větší množství vzorků a málo času je pro něho lepší použít T-test.

3.2.4 Forma ukládání informací o měřeních

Systém, který provnává výsledky měření, by měl mít nějaké informace o tom, k jaké verzi se měření vztahuje, nebo kde se soubor s výsledky nachází. Proto bylo při vývovji nutné se zamyslet nad tím jakými způsoby lze tyto informace získat. Systém PerfEval potřebuje o výsledcích měření vědět, kde jsou uloženy a ke které verzi softwaru bylo měření provedeno.

První zvažovaná varianta byla, že by existovala složka do které by uživatel vkládal výsledky měření do adresáře určeného systémem PerfEval, nebo do adresáře zadaného při inicializaci systému. Po každém měření by tedy uživatel výsledek akorát vložil do správné složky a o nic víc by se nemusel starat.

Toto řešení má několik nevýhod. Omezuje uživatele v tom, jak musí výsledky testů ukládat. Dále by se špatně určovalo ke které verzi bylo měření provedeno, protože toto se ve výsledcích měření běžnými frameworky neudává. Pravděpodobně by proto vznikla hierarchie v tomto adresáři a uživatel by například pomocí pojmenovávání složek určoval ke které verzi se měření vztahuje.

Celkově tedy tento způsob poskytuje jednoduché zjištění, kde se výsledek testu nachází. Je to pevně dané. Není ale možné jednoduše zjistit verzi, protože je nutné projít adresářovou strukturu, což může být při velkém množství dat pomalé.

Druhá zvažovaná varianta byla taková, že by PerfEval znal celý obsah souborů s výsledky testování. Konkrétně by v nějakém svém adresáři měl uložený celý výstup měření ve svém vlastním formátu, dooplněné o vlastní poteřbná metadata. Při zaznamenání nových výsledků by tedy došlo k načtení celého souboru s výsledky jeho přepracování do formátu, kterému by PerfEval rozumněl, a znovuuložení do svého adresáře s výsledky měření.

V rámci tohoto řešení je jednoduché zjistit k jaké verzi výsledky měření patří i kde je soubor uložen. Místo uložení souboru je pevně dané jako u prvního způsobu a verze je udaná v metadatech uložených přímo mezi předzpracovanými výsledky. Opět ale dochází k prohledávání souborů na disku jako u první varianty. Aby byl nalezen výsledek měření pro požadovanou verzi bylo by nutné nalézt soubor s výsledky měření k této verzi mezi všemi výsledky měření, které systém eviduje. Lze docílit částečného zrychlení, pokud by se tento systém ukládánní doplnil o nějakou variantu cache. Nicméně při testování dvou verzí z nichž jedna

není uložena v cache by došlo k pomalému prohledávání celé struktury.

Tento způsob tedy Nakonec také nebyl použit, z důvodu pomalého procházení dat uložených v pevné paměti.

Poslední uvažované řešení je strukturované ukládání metadat o výsledcích měření. Využije se databáze o jedné tabulce, kde je uvedena cesta k výsledku měření a verze ke které byl výsledek změřen. Uživatel do systému nahlásí, že má nový výsledek měření, kde je uložen, a k jaké verzi softwaru je měření provedeno. Systém si tato data pouze poznamená do databáze. Při vyhodnocení pak pomocí databázových dotazů nalezen snadno všechny soubory s výsledky měření k potřebné verzi.

Toto řešení poskytuje jak rychlé vkládání nových výsledků, tak rychlé vyhledávání výsledků měření k dané verzi. Zároveň není nutné žádné předzpracování souborů, a tudíž není třeba je při vkládání číst.

4. Uživatelská dokumentace systému PerfEval

4.1 Instalace

TODO: dodat, až se to bude vědět

4.2 Dostupné příkazy

Tato část práce se zabývá jednotlivými příkazy systému PerfEval a jejich parametry. V jednotlivých podkapitolách je vysvětleno k čemu se daný příkaz používá. V podkapitolách se nachází také informace o volitelných a povinných parametrech jednotlivých příkazů.

4.2.1 Příkaz init

Příkaz init slouží k inicializaci systému v rámci aktuálního pracovního adresáře. Systém PerfEval po spuštění hledá v pracovním adresáři složku .performance. Pokud složka není nalezena a nebyl zadán příkaz init, tak končí s chybou, že systém není inicializovaný.

Povinné argumenty:

`benchmark-parser` Nastavení parseru, který se bude pro tento projekt používat. Jméno parseru je zadávané jako parametr tohoto příznaku. Parser se volí podle použitého testovacího frameworku a výstupního formátu.

Volitelné argumenty:

`force` Příznak, který vynutí inicializaci i v případě, že je systém v adresáři již inicializovaný.

4.2.2 Příkaz index-new-result

Příkaz index-new-result slouží k přidání výsledků měření výkonu do databáze. Databáze je pro každý projekt zvlášť a je tedy možné na jednom zařízení systémem PerfEval spravovat více projektů. Při přidávání informací o souboru s výsledky je nutné zadat cestu k tomuto souboru. Verze ke které byly výsledky změřeny může být zadaná také, nebo ji systém zkusí odhadnout podle git repozitáře.

Povinné argumenty:

`path` Parametr tohoto příznaku udává cestu k souboru s výsledky měření.

Volitelné argumenty:

`version` Parametr tohoto příznaku udává textovou reprezentaci verze SW, která se měřila.

`tag` Parametr tohoto příznaku udává tag verze měření.

4.2.3 Příkaz index-all-results

Příkaz index-all-results slouží k přidání více výsledků měření výkonu do databáze. Při přidávání informací o souborech s výsledky je nutné zadat cestu k adresáři s těmito výsledky. Budou přidány všechny (i zanořené) soubory v tomto adresáři. Verze ke které byly výsledky změřeny může být zadaná také, nebo ji systém zkusí odhadnout podle git repozitáře.

Povinné argumenty:

`path` Parametr tohoto příznaku udává cestu k adresáři s výsledky měření.

Volitelné argumenty:

`version` Parametr tohoto příznaku udává textovou reprezentaci verze SW, která se měřila.

`tag` Parametr tohoto příznaku udává tag verze měření.

4.2.4 Příkaz evaluate

Příkaz evaluate porovnává dvě poslední zaznamenané verze, které byly změřeny. Verze k porovnání je možné specifikovat také manuálně pomocí příznaků. Výstupem je tabulka, nebo JSON s výsledky porovnání. V případě, že alespoň u jednoho porovnání došlo ke zhoršení výkonu, bude selhání signalizováno exit kódem 1.

Volitelné argumenty:

`new-version` Parametr udává textovou reprezentaci verze, která se má při porovnání považovat jako novější.

`new-tag` Pouze soubory s tímto tagem budou použity k porovnání jako novější.

`old-version` Parametr udává textovou reprezentaci verze, která se má při porovnání považovat jako starší.

`old-tag` Pouze soubory s tímto tagem budou použity k porovnání jako starší.

`t-test` T-test bude při statistickém porovnání použitý místo bootstrapu.

`json-output` Formát výstupu bude JSON.

`html-output` Výstup bude ve fromátu HTML stránky.

`html-template` Při použití příznaku html-output je možné ještě jako argument tohoto příznaku dodat novou HTML šablonu pro vypsání výsledků.

4.2.5 Příkaz list-undecided

Příkaz evaluate porovnává dvě poslední zaznamenané verze, které byly změřeny. Verze k porovnání je možné specifikovat také manuálně pomocí příznaků. Výstupem jsou dva sloupce oddělené znakem tabulátoru. V prvním sloupci je název testovací metody. Ve+druhém sloupci je počet měření, které jsou potřeba, aby bylo možné s dostatečnou pravděpodobností říct, že je výkon stejný.

Volitelné argumenty:

`new-version` Parametr udává textovou reprezentaci verze, která se má při porovnání považovat jako novější.

`new-tag` Pouze soubory s tímto tagem budou použity k porovnání jako novější.

`old-version` Parametr udává textovou reprezentaci verze, která se má při porovnání považovat jako starší.

`old-tag` Pouze soubory s tímto tagem budou použity k porovnání jako starší.

`t-test` T-test bude při statistickém porovnání použitý místo bootstrapu.

4.2.6 Příkaz list-results

Příkaz list-results vypíše informace o souborech uložených v databázi. Příkaz nemá žádné argumenty. Výstupní formát je tabulka s informacemi o souborech. Poskytuje jednoduchý přehled o souborech v databázi PerfEvalu.

4.3 Konfigurační soubor

Po spuštění systému PerfEval s příkazem init dojde k vytvoření složky .performance v pracovním adresáři. Ve vytvořené složce bude konfigurační soubor config.ini. Tento soubor obsahuje nastavení spojené s používáním systému PerfEval při vyhodnocování výkonu jednoho projektu. Změnou hodnot v konfiguračním souboru je možné omezeně změnit chování systému PerfEval.

Hodnoty v konfiguračním souboru

`critValue` Určuje pravděpodobnost chyby I. druhu při testování hypotézy, že výkony verzí jsou stejné.

`maxCIWidth` Určuje maximální relativní šířku intervalu spolehlivosti

`minTestCount` Určuje minimální počet testů (běhů), který bude dodán.

`maxTestCount` Určuje maximální počet testů (běhů), který je uživatel schopný změřit.

`tolerance` Určuje maximální pokles výkonu (relativně vůči starší verzi), který nezpůsobí selhání.

`git` Určuje jestli projekt podléhá správě verzí pomocí technologie git. Nabývá hodnot TRUE a FALSE.

`parserName` Jméno parseru, který bude použit při zpracovávání souborů s výsledky měření.

5. Programátorská dokumentace systému PerfEval

5.1 Architektura systému

Architekturu systému PerfEval je možné rozdělit do dvou částí. První část systému tvoří parser příkazové řádky a tzv. setup třídy. Druhou částí jsou tzv. command třídy, které provádí skutečně požadovanou činnost. V této části dokumentace nemusí být některé podrobnosti o implementaci, které je možné najít v automaticky generované dokumentaci JavaDoc. Jedná se například o argumenty zmiňovaných metod.

Vnitřní struktura běhu je velice jednoduchá. Metoda main má tři jednoduché úkoly. Zavolat metodu getCommand na třídě Parser. Tato metoda vrátí objekt typu Command, na kterém metoda main zavolá metodu execute. Metoda execute při návratu vrací enumerátor typu ExitCode. Na objektu ExitCode pak na konci metoda main zavolá metodu exit, která ukončuje program s požadovaným exit kódem. Metody getCommand a execute mohou skončit s výjimkami ParserException a PerfEvalCommandFailedException. Pro tyto případy tyto výjimky obsahují položku exitCode na které metoda main opět zavolá metodu exit.

5.1.1 Parser a setup třídy

Hlavní část třídy Parser tvoří slovník commandPerSetup a metoda getCommand. Ve slovníku jsou k jednotlivým řetězcovým klíčům přiřazeny objekty typu Supplier, které mají za úkol vracet instance typu CommandSetup. Klíče jsou skutečné řetězce zadávané uživatelem. Parser pak na základě příkazu zadaného na příkazové řádce zkonstruuje skrze Supplier získaný ze slovníku správnou instanci třídy CommandSetup. Na této instanci se zavolá metoda setup, která vrací objekt typu Command, podle definice rozhraní CommandSetup. Metoda getCommand je volaná z metody main, která tvoří hlavní metodu programu.

Třída Parser předává do metody setup objekty OptionSet a PerfEvalConfig. Třída PerfEvalConfig reprezentuje globální konfiguraci systému PerfEval podle konfiguračního souboru. Parser tedy skrze statické metody této třídy zařídí přečtení konfiguračního souboru do objektu PerfEvalConfig. Objekt OptionSet reprezentuje parametry a značky zadané uživatelem na příkazovou řádku. Jedná se o objekt z knihovny joptsimple. Rozpoznávány jsou všechny značky blíže specifikované ve třídě SetupUtilities.

SetupUtilities je třída obsahující statické položky a řetězcové konstanty. Tyto položky a konstanty slouží pro zpracování argumentů příkazové řádky. Setup třídy a Parser využívají jednotlivé metody této třídy. Protože některé setup třídy využívají stejných metod ze třídy SetupUtilities, tak jsou umístěny právě ve třídě SetupUtilities.

InitSetup je třída, která má za úkol připravit instanci InitCommand. Protože se jedná o setup třídu, tak shání informace pro práci Commandu, který konstruuje. Poté, co v rámci metody setup sežene všechny potřebné informace zkonstruuje InitCommand a vrátí jej. Tímto je InitCommand připraven k práci.

IndexNewSetup je třída pro přípravu instance AddFileCommand. IndexNewSetup dodává při konstrukci AddFileCommandu implementaci rozhraní Database, cestu k přidávanému souboru a instanci objektu ProjectVersion. ProjectVersion popisuje verzi softwaru pro kterou byl nově přidávaný výsledek změřen. Zkonstruovanou instanci třídy AddFileCommand vrací metoda setup.

IndexAllSetup je třída pro přípravu instance AddFilesFromDirCommand. IndexAllSetup dodává při konstrukci AddFilesFromDirCommandu implementaci rozhraní Database, cestu ke složce, ze které se budou přidávat soubory, a instanci objektu ProjectVersion. ProjectVersion popisuje verzi softwaru pro kterou byly nově přidávané výsledky změřeny. Zkonstruovanou instanci třídy AddFilesFromDirCommand vrací metoda setup.

EvaluateSetup je třída která připravuje EvaluateCLICommand. Musí v databázi najít soubory s výsledky měření, které bude EvaluateCLICommand porovnávat. Dále vyrábí implementace rozhraní StatisticTest a ResultPrinter, podle příznaků z příkazové řádky. EvaluateSetup musí také vyrobit instanci objektu PerformanceEvaluator. Tuto instanci vyrobí z nastavení v objektu PerfEvalConfig a ze zmíněné implementace StatisticTest. EvaluateCLICommand je zkonstruovaný z vyrobeného PerformanceEvaluatoru, ResultPrinteru, vstupních souborů s výsledky a MeasurementParseru. MeasurementParser je součástí objektu PerfEvalConfig.

UndecidedSetup konstruuje EvaluateCLICommand způsobem z předchozího odstavce. Jediný rozdíl je, že se použije pevně stanovený ResultPrinter typu UndecidedPrinter. Tato implementace CommandSetup vznikla proto, aby se vypsání informace o nerozhodných testech vypisovala pomocí odlišného příkazu.

Třída ListResultsSetup připravuje ListResultsCommand. Jediná činnost metody setup spočívá ve vyrobení instance rozhraní Database. Tuto instanci předá konstruktoru třídy ListResultsCommand. Vyrobenou instanci třídy ListResultsCommand vrací.

5.1.2 Command třídy

PerfEval je systém ovládaný pomocí příkazů (commandů) z příkazové řádky. Každému z příkazů odpovídá některá z implementací rozhraní Command. Implementace třídy Command musejí mít implementovanou jedinou metodu execute.

Třída AddFileCommand má za úkol přidat nový výsledek měření do databáze. Vnitřní položky třídy jsou fileToAdd, database a version. Metoda execute tedy volá metodu addFile na dodané implementaci databáze. Předávané paramtery jsou fileToAdd a version.

Třída AddFilesFromDirCommand má za úkol přidat všechny výsledky měření ve složce do databáze. Vnitřní položky třídy jsou sourceDirPath, database a version. Metoda execute tedy volá metodu addFilesFromDir na dodané implementaci databáze. Předávané paramtery jsou sourceDirPath a version.

Třída InitCommand má za úkol v adresáři, odkud byl PerfEval z příkazové řádky spuštěn, inicializovat systém PerfEval. Inicializace systému PerfEval probíhá v několika krocích. Nejprve se vytvoří adresář .performance. Následně se do .gitignore (v případě neexistence bude vytvořen) souboru doplní ignorace souborů a adresářů systému PerfEval. Nakonec je z parametrů dodaných při konstrukci a z výchozích parametrů vytvořen konfigurační soubor config.ini. V případě, že

systém je v adresáři již inicializovaný a nebyl dodán příznak násilné inicializace, tak systém nebude inicializovaný.

Třída EvaluateCLICommand má za úkol porovnat výsledky měření výkonu dvou různých verzí. Soubory s výsledky dvou verzí a parser k nim jsou dodané při konstrukci. Při konstrukci je dodaný objekt implementace rozhraní StatisticTest. Zmíněný objekt slouží ke statistickému porovnání výsledků měření. Při konstrukci je dále dodaná implementace rozhraní ResultPrinter. Implementace tohoto rozhraní rozhoduje o tom, jakým způsobem budou prezentovány výsledky porovnání. Třída v rámci volání metody execute nechá naparsovat soubory s výsledky měření do objektů typu Samples. Seznamy těchto objektů nechá porovnat pomocí statistického testu. Výsledkem je objekt MeasurementComparisonResultCollection. Tento objekt je nakonec pomocí dodané implementace rozhraní ResultPrinter prezentováno uživateli.

Třída ListResultsCommand a její metoda execute slouží k prostému vypsání obsahu databáze s výsledky měření. Databáze obsahuje informace o výsledcích měření jako je cesta k souboru a popis verze. Tyto údaje vypíše metoda print na třídě FileInfoPrinter do přehledné tabulky na standardní výstup.

5.1.3 Implementace rozhraní MeasurementParser

Rozhraní MeasurementParser je určeno ke zpracování souborů s výsledky měření výkonu. Třída MeasurementFactory vrací správnou instanci MeasurementParseru na základě dodaného jména (řetězce). Jméno parseru je uloženo v konfiguračním souboru.

Implementace rozhraní mohou při zpracovávání souborů vyhazovat runtime výjimku MeasurementParserException. Výjimka je odchytávaná mimo implementaci MeasurementParser v metodě execute třídy EvaluateCLICommand. Výjimku je tedy bezpečné vyhazovat.

Aktuálně dostupné implementace rozhraní MeasurementParser jsou JMHJSONParser a BenchmarkDotNetJSONParser. JMHJSONParser umí zprácovavat výstup frameworku JMH ve formátu JSON. BenchmarkDotNetJSONParser umí zprácovavat výstup frameworku BenchmarkDotNet ve formátu JSON.

5.1.4 Implementace rozhraní StatisticTest

Rozhraní StatisticTest definuje jaké metody mají mít implementace statistických testů pro porovnání dvourozmětných polí typu double. Jedná se o metodu, která vrací interval spolehlivosti. Interval spolehlivosti je intervalový odhad střední hodnoty pro rozdíl dvou náhodných veličin. Dále musí umět vrátit odhad minimálního počtu vzorků, které jsou zapotřebí pro zajištění dostatečně úzkého intervalu.

Implementace tohoto rozhraní jsou dvě. První je třída ParametricTest, která používá Welchův T-test, který je podrobněji popsán v kapitole 2.3.1. Druhou implementací je NonparamtericTest, který využívá hierarchického bootstrapu. Tento bootstrap je podrobněji popsaný v kapitole 2.3.2.

5.1.5 Implementace rozhraní Database

Rozhraní Database definuje funkce, které jsou požadovány od systému, který má uládat metadata o výsledcích měření. Jeho jediná implementace využívá technologie H2 embedded databáze. Technologie umožňuje vyhledávat položky v lokální databázi pomocí standardních SQL příkazů.

5.1.6 Struktura konfiguračního souboru

Konfigurační soubor systému PerfEval se nachází ve složce .performance a má název config.ini. V souboru jsou dvě sekce. Jedná se o sekce statistic a project.

V sekci statistic je možné nastavit pět různých paramterů. Prvním parametrem je critValue, který odpovídá velikosti chyby I. druhu při statistickém testu. K přímému ovlivnění statistických testů slouží ještě parametr maxCIWidth a tolerance. Parametr maxCIWidth říká, jaká může být relativní šířka intervalu spolehlivosti při nevyvrácení hypotézy, aby test prošel s kladným výsledkem. Parametr tolerance říká, o kolik může klesnout výkon v poměru s předchozí verzí. Parametr maxTestCount udává kolik je možné provést maximálně měření jedné verze. Parametr minTestCount udává kolik minimálně by mělo být dodaných naměřených výsledků (celkový počet běhů) pro porovnání výkonů.

V sekci project je možné nastavit používání git repozitáře (parametr git) pomocí hodnot TRUE a FALSE. Dalším parametrem je parserName. Tento parametr definuje parser, kterou bude vracet ParserFactory.

5.2 Rozšiřitelnost a její omezení

Systém PerfEval byl od počátku projektován tak, aby byl rozšiřitelný. Hlavní jádro celé aplikace tvoří vyhodnocování výkonu. V různých částech návrhu se vyskytují místa, kde je možné významným způsobem doplnit a změnit chování celé aplikace.

Nejdůležitější ze zmiňovaných rozšíření je rozšíření o datový formát. Tato možnost dělá z PerfEvalu poměrně univerzální nástroj. Činí ho totiž méně závislým na použitém měřícím systému a jeho výstupním formátu.

Rozšiřitelnost systému PerfEval není neomezená. Rozšíření, která nebudou zmíněná v této kapitole, pravděpodobně nebudou možná, nebo budou vyžadovat mnohem více času pro vývoj. Naopak pro rozšíření v této kapitole existuje návod jak systém jednoduše rozšířit.

5.2.1 Rozšíření o datový formát

Vezměmě opět příklad našeho programátora z první kapitoly. Programátor si napsal výkonnostní testy. Testy napsal pomocí nástroje, který nepodporuje PerfEval. Nicméně progrmátor ví, že systém PerfEval dělá přesně to, co potřebuje. Jediný problém je tedy ve vysvětlení svého datového formátu systému.

PerfEval je od počátku zamýšlen pro rozšíření v tomto místě. Programátor tedy musí prozkoumat, jak správně zkonstruovat třídu Samples. Třída Samples totiž reprezentuje jedno spuštění testovací metody měřícího frameworku. Všechny hodnoty naměřené jednou metodou jsou tedy uloženy zde.

Programátor musí naimplementovat vlastní implementaci rozhraní MeasurementParser. V tomto rozhraní je důležitá metoda getTestsFromFiles. Tato metoda na vstupu přijme všechny soubory s výsledky měření jedné verze. Výstupem je list objektů typů Samples. Pro každou metodu (měřící test), který se v souborech nachází, se v listu vyskytuje pouze jedna instance typu Samples.

Pokud jsem tedy měření pomocí frameworku spustil dvakrát se stejnými měřícími metodami, tak se ve výsledných Samples každá objeví právě jednou. Naměřené hodnoty budou všechny u této jedné instance Samples.

Poslední krok tohoto rozšíření po naimplementování rozhraní MeasurementParser je jeho registrace. Registrace probíhá tak, že se přidá jeden řádek do statického konstruktoru třídy ParserFactory. Na řádku bude přidání položky do objektu s názvem registeredParsers. Tento objekt je typu HashMap a přiřazuje k sobě Supplier, který vrací MeasurementParser a název typu String. Přidávaná položka je tedy String odpovídající názvu parseru a reference na metodu, která umí parser zkonstruovat.

Použití nového parseru je pak jednoduché. Při inicializaci systému PerfEval příkazem init se jako argumentu benchmark-parser použije jméno nového parseru. Dokonce jej začne hlásit v nabídce i chybová hláška v případě, že žádný parser není při inicializaci zadán.

5.2.2 Rozšíření o filtr

Pokud uživatel chce PerfEvalu změnit pořadí výpisu testů na výstupu, může použít filtr. Pokud mu žádný z připravených nevyhovuje, tak může implementovat nový filtr. Tento filtr je objektem typu Comparator, který porovnává instance MeasurementComparisonRecord. Pomocí tohoto komparátoru, pak dojde k setřídění vypisovaných prvků.

Dále je nutné přepsat metodu resolvePrinterForEvaluateCommand takovým způsobem, aby rozponávala i nový druh filtru podle příkazové řádky. Tuto metodu je možné nalézt ve třídě SetupUtilities. Posledním krokem je přidání nové značky do parseru příkazové řádky v metodě createParser. Komparátor se pak může předávat objektům typu ResultPrinter při konstrukci. O jejich dalším použití si tedy tyto objekty rozhodují samy.

5.2.3 Rozšíření o statistický test

Může se stát, že uživatel systému PerfEval má o svých datech nějaké informace, které by chtěl při vyhodnocování zohlednit. Může si tedy naprogramovat vlastní implementaci rozhraní StatisticTest.

Po naprogramování vlastní implementace rozhraní StatisticTest, pak jen stačí ve třídě SetupUtilities změnit chování metody resolveStatisticTest, tak aby rozpoznnávala novou implementaci podle příkazové řádky. Posledním krokem je přidání nové značky do parseru příkazové řádky v metodě createParser.

5.2.4 Rozšíření o možnost výpisu

Pro vypisování výsledků porovnání slouží rozhraní ResultPrinter. Pokud by si uživatel chtěl naimplementovat vlastní způsob vypisování, tak stačí implmetnovat jedinou jeho metodu PrintResults.

Přidání nového ResultPrinteru je podobné jako u přidávání nové implementace rozhraní StatisticTest. Stačí změnit implementaci metody resolvePrinterForEvaluateCommand ve třídě SetupUtilities. Úprava by měla být provedena tak, aby metoda uměla vrátit i novou implementaci. Pokud by bylo zapotřebí nového argumentu na příkazové řádce, tak je nutné jej také správně naimplementovat do metody createParser.

5.2.5 Změna použitého databázového systému

V důsledku dlouhého rozhodování se o tom, jak se budou informace o výsledcích měření ukládat vzniklo rozhraní Database. Rozhraní má spoustu metod. Pokud by se uživatel rozhodl změnit způsob ukládání dat o měřeních, tak by musel implementovat celé toto rozhraní. Po implementaci rozhraní pak už jen stačí změnit metodu constructDatabase ve třídě SetupUtilities, která vrací instanci objektu typu Database.

5.2.6 Rozšíření o příkaz

Každý příkaz PerfEval se skládá ze dvou tříd. Jedná se o třídu implementující rozhraní CommandSetup a o třídu implementující rozhraní Command. Pokud by uživatel chtěl doplnit nějaký nový příkaz, který by v kontextu systému dával smysl, tak je to možné. Dobrý příklad pro reprezentování nového příkazu bude vyhodnocení výkonu s grafickým výstupem.

Na rozdíl od stávajícího vyhodnocování by grafické vyhodnocování potřebovalo údaje z více než dvou posledních verzí. Proto by příkaz evaluate-graphical třída rozpoznala jako a vytvořila instanci třídy EvaluateGraphicalSetup. Na této třídě, implementaci CommandSetup, by pak zavolala metodu setup. Metoda setup na třídě EvaluateGraphicalSetup by měla za úkol z konfigurace PerfEvalu a příkazové řádky vyrobit objekt EvaluateGraphicalCommand. Tento objekt, který by implementoval rozhraní Command, by pak metoda getCommand na třídě Parser vrátila.

Zbytek programu by se již nezměnil metoda main ve třídě Main by vykonala dodaný příkaz. Spustila by standardním způsobem metodu execute na instanci objektu typu Command.

Zaregistrování nového příkazu by probíhalo přidáním nové položky do statického konstruktoru třídy Parser. Položka mapy commandPerSetup má obdobnou strukturu jako v případě rozšíření o nový MeasurementParser. Dodal by se řádek s názvem příkazu typu String a z reference na bezparametrický konstruktor implementace rozhraní CommandSetup. Název příkazu odpovídá příkazu, kterým se bude volat z příkazové řádky.

5.2.7 Omezená rošiřitelnost ve vyhodnocování

Jeden z nejhorších požadavků na rozšíření systému by bylo rozšíření v oblasti vyhodnocování. Jedná se zejména o změnu implementace třídy PerformanceEvaluator. Tato třída utváří celkovou vyhodnocovací logiku. Její rozšiřitelnost je omezená na implementace rozhraní StatisticTest.

Omezenou změnu chování vyhodnocování výkonnostních testů provést lze. Omezené změny ve vyhodnocování se provádí změnami hodnot uvnitř config.ini souboru.

6. Ukázka práce

Výsledkem této práce je tedy implementace systému pro porovnávání výkonu verzí softwaru pojmenovaný PerfEval. V této kapitole bude krok za krokem ukázáno, jak se systémem pracovat.

6.1 Používání systému PerfEval

V následujících dvou ukázkách bude vysvětleno, jak používat PerfEval. První ukázka bude cílit na nastínění, co+nejjednoduššího použití. Druhá se bude snažit o složitější použití systému PerfEval.

Jednoduché použití PerfEvalu

Následující kód popisuje obvyklou posloupnost příkazů práce se systémem PerfEval. Na začátku je nutné jej inicializovat. PerfEval bude inicializován pro použití JMHJSONParseru. Následně se přidají výsledky měření několika (alespoň dvou) různých verzí. Nakonec program vypíše jestli výkonnostní testy prošly, nebo ne.

```
perfeval init --benchmark-parser JMHJSONParser
perfeval index-all-results --path path/to/dir/with/
   results --version x.y.z
perfeval index-all-results --path path/to/another/
   dir/with/results --version x.y.z2
perfeval evaluate && echo "Performance␣test␣passed"
    | exit 0
echo "Performance␣test␣failed" | exit 1
```

Návrh skriptu pro doměření výsledků

Následující kód nastiňuje možnost využití příkazu list-undecided. Výstupem tohoto příkazu jsou dva tabulátorem oddělené sloupce. V prvním sloupci jsou názvy metod, pro něž systém eviduje málo naměřených běhů. Ve druhém sloupci je uveden tento počet. Příkaz je určený pro skriptování, proto není dodaná žádná další hlavička.

V případě, že výstupem není žádný výpis, tak je hodnot u všech testovacích metod naměřeno dostatek. Druhou alternativou je, že v důsledku nastavení v konfiguračním souboru systém vyhodnotil, že není možné požadovaný počet testů doměřit. Následné vyhodnocení pak bude vyžadovat kontrolu uživatelem, Protože systém PerfEval bude vyhodnocení považovat za nevyhovující.

Skript projde všechny řádky výpisu. Pokud je výpis prázdný, tak skončí. V následujícím kódu je celá situace velmi zjednodušena. Nalezene se maximální počet testů, který je zapotřebí změřit. Pro tento maximální počet se změří výkony obou verzí znovu. Tudíž může dojít i k měření pomocí metod, které nejsou zapotřebí. Výsledky těchto měření se zaznamenají do systému PerfEval. Po doběhnutí všech

měření skript skončí. Vyhodnotí mezi sebou výsledky těchto verzí a skončí. Parametry $1 a $2 jsou stará a nová verze k porovnání. Předpokládá se, že příkaz measure provede měření verze zadané jako první argument a výsledek uloží do souboru specifikovaného jako druhý argument.

```
#!/bin/bash

# Function to find the maximum value in the second
   column
find_max() {
    max=0
    while IFS=$'\t' read -r _ val; do
        if [[ $val =~ ^[0-9]+?$ ]]; then
            if (( $(echo "$val > $max" | bc -l) ));
               then
                max=$val
            fi
        fi
    done
    echo "$max"
}

# Main script
index=1
while true; do
    output=$(perfeval list-undecided --old-version
       "$1" --new-version "$2")
    if [[ -n "$output" ]]; then
        max=$(echo "$output" | find_max)
        for ((i=1; i<=$max; i++)); do
            result_file="old_version_$index"
            measure "$1" "$result_file"
            perfeval index-new-result --path "
               $result_file" --version "$1"

            result_file="new_version_$index"
            measure "$2" "$result_file"
            perfeval index-new-result --path "
               $result_file" --version "$2"
            ((index++))
        done
    else
        perfeval evaluate --old-version "$1" --new-
           version "$2"
        exit $?
        break
    fi
done
```

6.2 Nasazení systému v praxi

Pro prezentování fungování PerfEvalu s reálnými daty jsem si vybral softwarový projetk crate. Tento projekt je dostupný na GitHubu (Motl a kol. (2024)) a jeho autoři používají k vyhodnocování výkonu framework JMH. Pomocí testů od autorů bylo provedeno měření několika verzí. Byly vybírány zejména verze jejichž commit message se zmiňovala o zlepšení výkonu a verze před tímto zlepšením.

Projekt byl z výše zmíněného Githubu naklonován do lokálního adresáře. Následně došlo k nalezení commitů u nichž zpráva tvrdí, že došlo k výraznému zlepšení výkonu.

```
commit b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929
Author: Andrei Dan <andrei@crate.io>
...skipping...
    using DocValues over source lookup will greatly
       increase performance.
    the reasons for doing a source lookup instead were
       mostly related to the
    pre-DocValues era were the fieldcache was used
       instead.

commit 968a5fcae558f904fb2ebdc747bae1dadc567cfb
Author: Sebastian Utz <su@rtme.net>
Date:   Fri Jan 12 12:37:58 2018 +0100

    re-enables StringTypeTest JMH benchmark by moving
       it to benchmarks module

    was a left over from the recent JMH benchmarks
       refactoring
```

U obou verzí došlo měření výkonu. Měření projektu jsou součástí projektu, takže došlo pouze ke spuštění JMH testů a uložení výsledků ve formátu JSON. Po naměření byly porovnány výše uvedené verze.

Path	Date of creation	Date of commit	Version
../test-results/crate-f6c3d156e826c811d2da170b8ae36f67be102e54-result.json	2024-02-27 18:13:03.935	2024-03-04 19:10:25.29	f6c3d156e826c811d2da170b8ae36f67be102e54
../test-results/crate-5ad19461d9bbdfd812586612f3598ef91fb1d529-result.json	2024-02-27 00:41:28.277	2024-03-04 19:10:57.949	5ad19461d9bbdfd812586612f3598ef91fb1d529
../test-results/crate-30081686c6e38b3b20817d86a3a38eb652c8625a-result.json	2024-02-26 21:35:45.253	2024-03-04 19:11:31.396	30081686c6e38b3b20817d86a3a38eb652c8625a
../test-results/crate-df0635dec721861d1f9f5fbde4e9e8647bfebc13-result.json	2024-02-26 18:41:02.424	2024-03-04 19:11:55.306	df0635dec721861d1f9f5fbde4e9e8647bfebc13
../test-results/crate-1866a331f90aaaeabc6a789a2c11cee2aac5c611-result.json	2024-02-26 15:49:43.865	2024-03-04 19:12:18.869	1866a331f90aaaeabc6a789a2c11cee2aac5c611
../test-results/crate-8785042ff7bb70d9e5e6463e1c92baedc5a89205-result.json	2024-02-26 12:18:50.026	2024-03-04 19:12:43.886	8785042ff7bb70d9e5e6463e1c92baedc5a89205
../test-results/crate-95d5566b6e25f32276cf52ba1df200663fba0ecc-result.json	2024-02-26 01:03:35.439	2024-03-04 19:13:11.05	95d5566b6e25f32276cf52ba1df200663fba0ecc
../test-results/crate-e73db94230c7eaa5c8120037c83fac920e647d0a-result.json	2024-02-25 22:09:01.861	2024-03-04 19:13:33.664	e73db94230c7eaa5c8120037c83fac920e647d0a
../test-results/crate-789fd702bd2a04a07a8faefca62f89bbdefc9609-result.json	2024-02-24 18:33:50.328	2024-03-04 19:14:18.01	789fd702bd2a04a07a8faefca62f89bbdefc9609
../test-results/crate-b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929-result.json	2024-02-24 21:43:23.631	2024-03-04 19:14:40.932	b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929
../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb
../test-results/crate-56757de9bec8a06c6c5c376ad84905cb58b31ead-result.json	2024-02-23 23:46:11.076	2024-03-04 19:15:44.777	56757de9bec8a06c6c5c376ad84905cb58b31ead
../test-results/crate-4814fd4fb7a1faf974e6325c7e30d49464d9486a-result.json	2024-02-23 21:00:35.654	2024-03-04 19:16:07.948	4814fd4fb7a1faf974e6325c7e30d49464d9486a
../test-results/crate-d72839ebdb88f81f6b48c38f84ce9fb22941b20a-result.json	2024-02-23 18:13:11.051	2024-03-04 19:16:29.385	d72839ebdb88f81f6b48c38f84ce9fb22941b20a

Obrázek 6.1: Obsah databáze systému PerfEval pro projekt crate

Z výsledků porovnání je vidět, že došlo k nějakému zlepšení výkonu. I přes to, že ve dvou případech došlo ke zlepšení v řádu stovek procent, tak test selhal s kódem 1. Toto selhání je zapříčiněno přísliš malým počtem změřených běhů pro dvě měřené metody. Jedná se o metodu benchmarkRuntimeMethod a o metodu benchmarkMemoryMXBeanMethod.

```
old version: 968a5fcae558f904fb2ebdc747bae1dadc567cfb
new version: b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929
```

Name	NewAverage	OldAverage	Change [%]	Lower CI bound	Upper CI bound	Comparison result	Comparison verdict
iccS.measure_Long_parseLong_invalid	1 us/op	4 us/op	+180.20	+1.191	+3.609	different distribution	OK
iccS.measure_StringUti~ParseLong_invalid	0.000 us/op	0.000 us/op	+80.88	+0.005	+0.006	different distribution	OK
iceeccL.measureConsumeL~neBatchIterator	209 ms/op	217 ms/op	+3.60	+0.156	+14.049	different distribution	OK
iceeaG.measureGroupingO~NumericDocValues	349 ms/op	479 ms/op	+37.14	+87.304	+173.723	different distribution	OK
icejC.benchmarkRuntimeMethod	3 us/op	4 us/op	+26.69	-6.477	+8.258	not enough samples (5 samples needed)	NOT OK
iceedmS.measurePagingIteratorWithRepeat	16 ms/op	16 ms/op	+1.73	+0.116	+0.417	different distribution	OK
iccS.measure_Long_parseLong_valid	0.000 us/op	0.000 us/op	+79.10	+0.015	+0.015	different distribution	OK
iceeaG.measureGroupBySumLong	592 ms/op	578 ms/op	-2.34	-19.612	-8.156	different distribution	OK
ict.StringTypeTest.booleanConversion	1922379545 ops/s	1054958427 ops/s	+82.22	-901042066.473	-837411099.961	different distribution	OK
iceeaG.measureGroupByMinString	603 ms/op	751 ms/op	+24.57	+9.100	+456.400	different distribution	OK
icoaI.benchmarkIntervalAvgAggregation	271 us/op	269 us/op	-0.84	-13.303	+8.921	same distribution	OK
icejC.benchmarkMemoryMXBeanMethod	37 us/op	47 us/op	+25.67	-75.669	+94.796	not enough samples (5 samples needed)	NOT OK
iceeccO.measureLoadAndC~neBatchIterator	6 s/op	6 s/op	+2.58	+0.072	+0.254	different distribution	OK
icoaH.benchmarkHLLPlusPlusMurmur128	314 us/op	319 us/op	+1.44	-6.867	+15.934	same distribution	OK
icdjR.measureConsumeBlockNestedLoopJoin	102 ms/op	148 ms/op	+45.13	+6.507	+96.762	different distribution	OK
icdjR.measureConsumeHashInnerJoin	2 ms/op	3 ms/op	+4.68	-0.015	+0.239	same distribution	OK
iceedmS.measurePagingIt~orWithoutRepeat	10 ms/op	11 ms/op	+10.14	+0.680	+1.243	different distribution	OK
icoaH.benchmarkHLLAggregationOffHeap	479 us/op	509 us/op	+6.19	-7.892	+67.556	same distribution	OK
iceT.currentTimeMillisNextGen	0.00 us/op	0.00 us/op	+232.61	+0.071	+0.080	different distribution	OK
icdjR.measureConsumeHa~thHashCollisions	0.0 ms/op	0.0 ms/op	+15.90	+0.026	+0.095	different distribution	OK
icoaH.benchmarkHLLAggregationOnHeap	429 us/op	452 us/op	+5.28	+20.637	+24.598	different distribution	OK
iceeaG.measureGroupingOnNumericDocValues	316 ms/op	323 ms/op	+2.21	+1.793	+12.195	different distribution	OK
icoaH.benchmarkHLLPlusPlusMurmur64	294 us/op	297 us/op	+0.89	-27.193	+32.555	same distribution	OK
icdjR.measureConsumeNestedLoopLeftJoin	107 ms/op	115 ms/op	+7.06	-2.964	+26.753	same distribution	OK
icoaI.benchmarkIntervalSumAggregation	613 us/op	644 us/op	+5.02	+13.635	+48.373	different distribution	OK
iceedmS.measurePagingIteratorWithSort	474 ms/op	573 ms/op	+20.97	+65.419	+129.678	different distribution	OK
iceT.currentTimeMillisOriginal	0.000 us/op	0.000 us/op	+6.29	+0.001	+0.001	different distribution	OK
iceeaG.measureGroupingOnLongArray	196 ms/op	346 ms/op	+76.22	+134.003	+165.160	different distribution	OK
iccS.measure_StringUti~ryParseLong_valid	0.00 us/op	0.00 us/op	+84.98	+0.014	+0.019	different distribution	OK
icdjR.measureConsumeNestedLoopJoin	81 ms/op	86 ms/op	+6.42	+2.919	+7.778	different distribution	OK
iceedmS.measureListSort	468 ms/op	495 ms/op	+5.82	+18.919	+37.849	different distribution	OK
iceeaA.measureAggregateCollector	155 us/op	158 us/op	+2.31	-3.287	+9.097	same distribution	OK

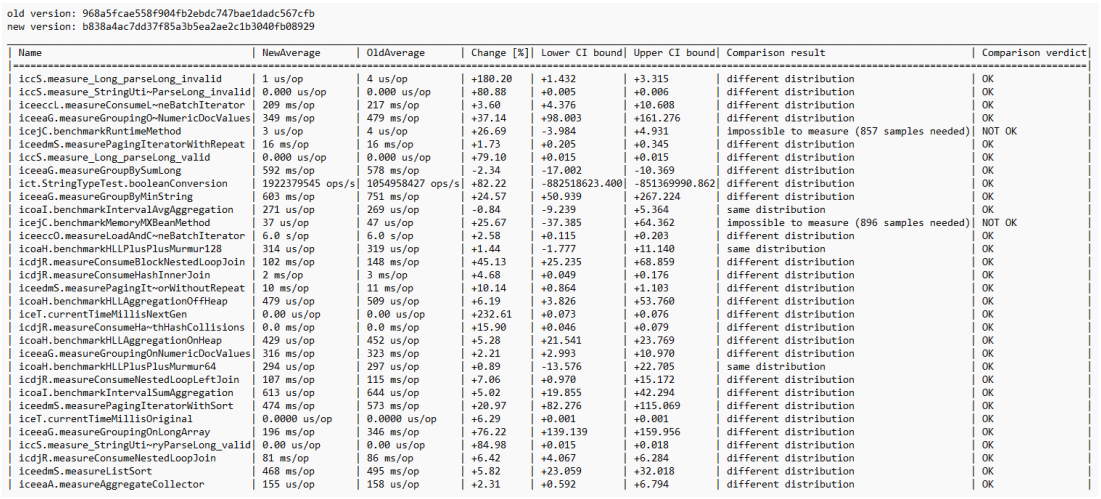
Obrázek 6.2: Výsledek porovnání mezi commity

V důsledku malého počtu změřených běhů bude měření dalších běhů simulované. Nové výsledky měření budou simulovány tak, že se již přidané soubory s výsledky přidají vícekrát. Toto vícenásobné přidání se projeví v databázi PerfEvalu obsahující informace o výsledcích měření tak, že se zde stejný řádek bude vyskytovat vícekrát.

	Path	Date of creation	Date of commit	Version	Tag
1.	../test-results/crate-f6c3d156e826c811d2da170b8ae36f67be102e54-result.json	2024-02-27 18:13:03.935	2024-03-04 19:10:25.29	f6c3d156e826c811d2da170b8ae36f67be102e54	
2.	../test-results/crate-5ad19461d9bbdfd812586612f3598ef91fb1d529-result.json	2024-02-27 00:41:28.277	2024-03-04 19:10:57.949	5ad19461d9bbdfd812586612f3598ef91fb1d529	
3.	../test-results/crate-30081686c6e38b3b20817d86a3a38eb652c8625a-result.json	2024-02-26 21:35:45.253	2024-03-04 19:11:31.396	30081686c6e38b3b20817d86a3a38eb652c8625a	
4.	../test-results/crate-df0635dec721861d1f9f5fbde4e9e8647bfebc13-result.json	2024-02-26 18:41:02.424	2024-03-04 19:11:55.306	df0635dec721861d1f9f5fbde4e9e8647bfebc13	
5.	../test-results/crate-1866a331f90aaaeabc6a789a2c11cee2aac5c611-result.json	2024-02-26 15:49:43.865	2024-03-04 19:12:18.869	1866a331f90aaaeabc6a789a2c11cee2aac5c611	
6.	../test-results/crate-8785042ff7bb70d9e5e6463e1c92baedc5a89205-result.json	2024-02-26 12:18:50.026	2024-03-04 19:12:43.886	8785042ff7bb70d9e5e6463e1c92baedc5a89205	
7.	../test-results/crate-95d5566b6e25f32276cf52ba1df200663fba0ecc-result.json	2024-02-26 01:03:35.439	2024-03-04 19:13:11.05	95d5566b6e25f32276cf52ba1df200663fba0ecc	
8.	../test-results/crate-e73db94230c7eaa5c8120037c83fac920e647d0a-result.json	2024-02-25 22:09:01.861	2024-03-04 19:13:33.664	e73db94230c7eaa5c8120037c83fac920e647d0a	
9.	../test-results/crate-789fd702bd2a04a07a8faefca62f89bbdefc9609-result.json	2024-02-24 18:33:50.328	2024-03-04 19:14:18.01	789fd702bd2a04a07a8faefca62f89bbdefc9609	
10.	../test-results/crate-b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929-result.json	2024-02-24 21:43:23.631	2024-03-04 19:14:40.932	b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929	
11.	../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb	
12.	../test-results/crate-56757de9bec8a06c6c5c376ad84905cb58b31ead-result.json	2024-02-23 23:46:11.076	2024-03-04 19:15:44.777	56757de9bec8a06c6c5c376ad84905cb58b31ead	
13.	../test-results/crate-4814fd4fb7a1faf974e6325c7e30d49464d9486a-result.json	2024-02-23 21:00:35.654	2024-03-04 19:16:07.948	4814fd4fb7a1faf974e6325c7e30d49464d9486a	
14.	../test-results/crate-d72839ebdb88f81f6b48c38f84ce9fb22941b20a-result.json	2024-02-23 18:13:11.051	2024-03-04 19:16:29.385	d72839ebdb88f81f6b48c38f84ce9fb22941b20a	
15.	../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb	
16.	../test-results/crate-b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929-result.json	2024-02-24 21:43:23.631	2024-03-04 19:14:40.932	b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929	
17.	../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb	
18.	../test-results/crate-b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929-result.json	2024-02-24 21:43:23.631	2024-03-04 19:14:40.932	b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929	
19.	../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb	
20.	../test-results/crate-b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929-result.json	2024-02-24 21:43:23.631	2024-03-04 19:14:40.932	b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929	
21.	../test-results/crate-968a5fcae558f904fb2ebdc747bae1dadc567cfb-result.json	2024-02-25 19:20:00.017	2024-03-04 19:15:09.067	968a5fcae558f904fb2ebdc747bae1dadc567cfb	

Obrázek 6.3: Obsah databáze systému PerfEval pro projekt crate se simulovanými záznamy

Výstup programu s nasimulovanými výsledky měření ukazuje, že výsledky není možné doměřit. V konfiguračním souboru je maximální počet měření, který lze provést shora omezen číslem 100. Pokud tedy PerfEval odhadl, že by bylo zapotřebí více než 100 měření, tak selže (exit kód 1). PerfEval tedy není schopen v tomto případě sám o sobě rozhodnout jestli se nová verze nezhoršila.



```
old version: 968a5fcae558f904fb2ebdc747bae1dadc567cfb
new version: b838a4ac7dd37f85a3b5ea2ae2c1b3040fb08929
```

Name	NewAverage	OldAverage	Change [%]	Lower CI bound	Upper CI bound	Comparison result	Comparison verdict
iccS.measure_Long_parseLong_invalid	1 us/op	4 us/op	+180.20	+1.432	+3.315	different distribution	OK
iccS.measure_StringUti~ParseLong_invalid	0.000 us/op	0.000 us/op	+80.88	+0.005	+0.006	different distribution	OK
iceeccL.measureConsumeL~neBatchIterator	209 ms/op	217 ms/op	+3.60	+4.376	+10.608	different distribution	OK
iceeaG.measureGroupingO~NumericDocValues	349 ms/op	479 ms/op	+37.14	+98.003	+161.276	different distribution	OK
icejC.benchmarkRuntimeMethod	3 us/op	4 us/op	+26.69	-3.984	+4.931	impossible to measure (857 samples needed)	NOT OK
iceedmS.measurePagingIteratorWithRepeat	16 ms/op	16 ms/op	+1.73	+0.205	+0.345	different distribution	OK
iccS.measure_Long_parseLong_valid	0.000 us/op	0.000 us/op	+79.10	+0.015	+0.015	different distribution	OK
iceeaG.measureGroupBySumLong	592 ms/op	578 ms/op	-2.34	-17.002	-10.369	different distribution	OK
ict.StringTypeTest.booleanConversion	1922379545 ops/s	1054958427 ops/s	+82.22	-882518623.400	-851369990.862	different distribution	OK
iceeaG.measureGroupByMinString	603 ms/op	751 ms/op	+24.57	+50.939	+267.224	different distribution	OK
icoaI.benchmarkIntervalAvgAggregation	271 us/op	269 us/op	-0.84	-9.239	+5.364	same distribution	OK
icejC.benchmarkMemoryMXBeanMethod	37 us/op	47 us/op	+25.67	-37.385	+64.362	impossible to measure (896 samples needed)	NOT OK
iceeccO.measureLoadAndC~neBatchIterator	6.0 s/op	6.0 s/op	+2.58	+0.115	+0.203	different distribution	OK
icoaH.benchmarkHLLPlusPlusMurmur128	314 us/op	319 us/op	+1.44	-1.777	+11.140	same distribution	OK
icdjR.measureConsumeBlockNestedLoopJoin	102 ms/op	148 ms/op	+45.13	+25.235	+68.859	different distribution	OK
icdjR.measureConsumeHashInnerJoin	2 ms/op	3 ms/op	+4.68	+0.049	+0.176	different distribution	OK
iceedmS.measurePagingIt~orWithoutRepeat	10 ms/op	11 ms/op	+10.14	+0.864	+1.103	different distribution	OK
icoaH.benchmarkHLLAggregationOffHeap	479 us/op	509 us/op	+6.19	+3.826	+53.760	different distribution	OK
iceT.currentTimeMillisNextGen	0.00 us/op	0.00 us/op	+232.61	+0.073	+0.076	different distribution	OK
icdjR.measureConsumeHa~thHashCollisions	0.0 ms/op	0.0 ms/op	+15.90	+0.046	+0.079	different distribution	OK
icoaH.benchmarkHLLAggregationOnHeap	429 us/op	452 us/op	+5.28	+21.541	+23.769	different distribution	OK
iceeaG.measureGroupingOnNumericDocValues	316 ms/op	323 ms/op	+2.21	+2.993	+10.970	different distribution	OK
icoaH.benchmarkHLLPlusPlusMurmur64	294 us/op	297 us/op	+0.89	-13.576	+22.705	same distribution	OK
icdjR.measureConsumeNestedLoopLeftJoin	107 ms/op	115 ms/op	+7.06	+0.970	+15.172	different distribution	OK
icoaI.benchmarkIntervalSumAggregation	613 us/op	644 us/op	+5.02	+19.855	+42.294	different distribution	OK
iceedmS.measurePagingIteratorWithSort	474 ms/op	573 ms/op	+20.97	+82.276	+115.069	different distribution	OK
iceT.currentTimeMillisOriginal	0.0000 us/op	0.0000 us/op	+6.29	+0.001	+0.001	different distribution	OK
iceeaG.measureGroupingOnLongArray	196 ms/op	346 ms/op	+76.22	+139.139	+159.956	different distribution	OK
iccS.measure_StringUti~ryParseLong_valid	0.00 us/op	0.00 us/op	+84.98	+0.015	+0.018	different distribution	OK
icdjR.measureConsumeNestedLoopJoin	81 ms/op	86 ms/op	+6.42	+4.067	+6.284	different distribution	OK
iceedmS.measureListSort	468 ms/op	495 ms/op	+5.82	+23.059	+32.018	different distribution	OK
iceeaA.measureAggregateCollector	155 us/op	158 us/op	+2.31	+0.592	+6.794	different distribution	OK

Obrázek 6.4: Výsledek porovnání po simulování měření

Závěr

Seznam použité literatury

Akinshin, A. (2024). dotnet/BenchmarkDotNet. URL https://github.com/dotnet/BenchmarkDotNet. original-date: 2013-08-18T06:17:48Z.

Heisler, B. (2024). criterion - Rust. URL https://docs.rs/criterion/latest/criterion/index.html.

Motl, A., Schmidtmer, N., Fussenegger, M., Traar, G., Utz, S. a Reisegger, F. (2024). crate. URL https://github.com/crate/crate. original-date: 2013-04-10T09:17:16Z.

Shipilëv, A. (2024). openjdk/jmh. URL https://github.com/openjdk/jmh. original-date: 2019-05-15T06:47:19Z.

Stefan, P., Horký, V., Bulej, L. a Tůma, P. (2017). Unit testing performance in java projects: Are we there yet? In *Proc. 8th ACM/SPEC Intl. Conf. on Performance Engineering (ICPE)*, pages 401–412. ISBN 978-1-4503-4404-3. doi: 10.1145/3030207.3030226.

Turčicová, M. (2021). Dvouvýběrové testy. URL https://www.karlin.mff.cuni.cz/~turcic/Dvouvyberove_testy.pdf.

Šámal, R. (2023). NMAI059 Pravděpodobnost a statistika 1. URL https://iuuk.mff.cuni.cz/~samal/vyuka/2223/PSt1/skripta.pdf.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha