Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Nástroj na analýzu úloh časem řízeného plánovače úloh cron

Petr Klejch

Vedoucí práce: Ing. Radomír Polách

11. května 2015

Poděkování

Děkuji Ing. Radomíru Poláchovi za poskytnuté rady s bakalářskou prací, rodině a přátelům za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Klejch, Petr. *Nástroj na analýzu úloh časem řízeného plánovače úloh cron.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce se zabývá analýzou a optimalizací naplánovaných úloh časového plánovače úloh cron. Cílem této práce je vytvořit nástroj v jazyce Python, který má za úkol analyzovat a najít optimální rozložení úloh v čase.

Klíčová slova cron, crontab, python, analýza běhu úloh, optimalizace, genetické algoritmy

Abstract

This bachelor's thesis describes analysis and optimalization of scheduled tasks, scheduled by time scheduler cron. Goal of this thesis is to create tool in Python, which will analyze those tasks and find optimal distribution in time.

Keywords cron, crontab, python, analysis of tasks, optimalization, genetic algorithms

Obsah

Seznam obrázků

Úvod

Servery často spouštějí různé programy a skripty periodicky a automaticky, například import zboží do databáze elektronického obchodu, promazávání nepotřebných a záloha důležitých dat, atd. Tuto automatizovanou činnost zajišťují plánovače úloh. Plánovač úloh pro unixové operační systémy se nazývá cron, jeho úkolem je zajistit, že se daná úloha spustí v určitý čas.

Cílem této bakalářské práce je vytvořit nástroj, který bude analyzovat běh naplánovaných úloh časovým plánovačem cron a pokusí se o nalezení optimálního rozložení úloh v čase.

Analýza úloh se skládá z vizualizace běhu naplánovaných úloh a sestavení grafu zátěže. Tato analýza využívá metadata, která jsou pomocí komentářů přidávána k naplánovaným úlohám.

Optimalizace je realizována pomocí genetického algoritmu, který hledá nejrovnoměrnější rozložení zátěže a tím eliminuje zátěžové špičky.

Výstupem optimalizace je nový konfigurační soubor pro plánovač cron, který lépe rozloží úlohy v čase.

Kapitola 1

Analýza

1.1 Plánovač úloh cron

Cron je softwarový démon (program, který běží na pozadí a není v přímém kontaktu s uživatelem), který má za úkol spouštět programy nebo skripty v určitý čas. Tomuto typu softwaru se říká plánovač úloh, alternativa pro operační systémy Windows se nazývá Task Scheduler (Plánovač úloh).

Mezi různými unixovými systémy existuje více implementací tohoto plánovače, které se liší umístěním a formátem konfiguračních souborů a možnostmi spouštění úloh. Dokonce i mezi různými distribucemi operačního systému GNU/Linux existují rozdíly, následující kapitola se bude týkat plánovače cron z distribuce Debian a něj odvozených (např. Ubuntu).

1.1.1 Princip plánovače

Plánovač cron při startu načte všechny konfigurační soubory crontab. Poté se cron probudí každou minutu a zkontroluje jestli neexistují nějaké úlohy, které by se měli spustit.

Zároveň kontroluje jestli se změnil čas modifikace adresářů s konfiguračními soubory crontab a čas modifikace hlavního konfiguračního souboru crontab. Pokud se změnil čas modifikace adresáře, plánovač poté prohlédne všechny soubory a znovu načte ty, které byly změněny. Z tohoto důvodu se podle [1] nemusí plánovač restartovat pokaždé, když se změní konfigurační soubor.

Plánovač cron nedědí z důvodů konzistence žádné proměnné prostředí. Při spouštění úloh však používá tyto proměnné: PATH, HOME, SHELL, LOGNAME a MAILTO.

PATH Tato proměnná má stejný význam jako stejnojmenná proměnná prostředí PATH v unixových systémech, obsahuje adresáře oddělené dvojtečkou, ve kterých se hledají skripty a programy ke spuštění. Plánovač nastavuje tuto proměnnou na „/usr/bin:/bin“, uživatel ji může upravit a přidat tak vlastní složky.

HOME Tato proměnná je nastavena podle domovského adresáře uživatele, který program spouští. Tento adresář slouží zároveň jako výchozí pracovní adresář.

LOGNAME Tato proměnná je nastavena podle uživatele, který program spouští. Tuto proměnnou uživatel nemůže změnit.

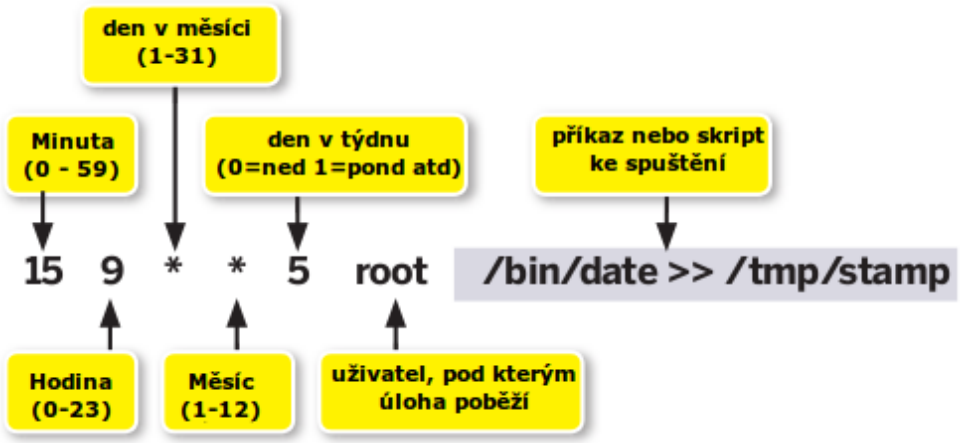
SHELL Tato proměnná je nastavena plánovačem na hodnotu „/bin/sh“, uživatel ji může změnit a ovlivní tím jakým shellem budou interpretovány skripty.

MAILTO Touto proměnnou lze určit, kterému uživateli se budou zasílat standardní a chybové výstupy úloh. Pokud tato proměnná neexistuje, plánovač cron zasílá standardní i chybový výstup naplánované úlohy pomocí emailu uživateli, který daný program či skript spouští. Pokud proměnná existuje, ale je prázdná, výstup nebude poslán nikomu.

1.1.2 Konfigurační soubory crontab

1.1.2.1 Formát konfiguračního souboru

Plánovač úloh cron se nastavuje pomocí konfiguračních souborů crontab. Název crontab vznikl složením dvou slov: cron a table (tabulka). Formát konfiguračního souboru je jednoduchý, každý řádek odpovídá jednomu záznamu a každý záznam má pevně danou strukturu, právě proto formát konfiguračního souboru připomíná tabulku.



Obrázek 1.1: Příklad záznamu v konfiguračním souboru crontab, převzato z [2]

Konfigurační soubor podporuje rovněž komentáře, komentář je uvozen znakem „#“. Komentář může být na samostatném řádku nebo může být uveden za záznamem v konfiguračním souboru.

Jak lze vidět na obrázku 1.1, každý záznam se skládá ze sedmi polí (záznam v uživatelském konfiguračním souboru crontab obsahuje jen šest polí). Prvních pět polí určuje, kdy se daná úloha spustí, šesté pole určuje, pod kterým uživatelem bude úloha spuštěna (uživatel se však neuvádí u uživatelských konfiguračních souborů) a poslední pole určuje program, který se spustí.

Z hlediska plánování je nejdůležitějších prvních pět polí, které určují, kdy bude daná úloha spuštěna. Konkrétně v tomto pořadí určují: minutu, hodinu, den v měsíci, měsíc a den v týdnu. Pro každé toto pole lze zadat:

- Konkrétní hodnotu, např.: 5.
- Rozsah oddělený pomlčkou, např.: 1-4.
- Výčet hodnot oddělených čárkou, např.: 1,3,7.
- Hvězdičku „*“, která má význam vždy.
- Hvězdičku či rozsah hodnot následované lomítkem a hodnotou, což bude mít za význam každých n minut z daného intervalu. Např.: */5 či 10-50/5.

Dále lze kombinovat některé druhy zápisů, např. lze kombinovat rozsahy a výčty, dva rozsahy oddělené čárkou, atd. Některé implementace plánovače cron umožňují používat zástupná makra jako jsou např.: @daily, @monthly, atd.

Příklady záznamů v konfiguračním souboru crontab:

```
#Tento záznam spustí každou minutu skript backup.sh
* * * * * root backup.sh

#Tento záznam spustí každý den o půlnoci skript hello.sh
#zápis 0 0 * * * je stejný jako makro @daily
0 0 * * * root hello.sh

#Tento záznam spustí každé pondělí ve 2:37 program date
37 2 * * 1 root date

#Tento záznam spustí každých 5 minut, každý den během dopoledne
# v únoru a červnu program who, pod uživatelem klejcpet
*/5 0-11 * 2,6 * klejcpet who
```

Jak lze z ukázek vidět, crontab má velmi široké vyjadřovací schopnosti při zachování jednoduchého formátu. Tento jednoduchý přístup však naráží na své limity, pokud má uživatel specifické nároky na běh.

Např. uživatel nemůže nastavit, aby byla úloha spuštěna každých 30 sekund, největší frekvence, která lze nastavit, je jednou za minutu. Tento problém lze však vyřešit pomocí více záznamů a programu sleep. Dále podle [3], uživatel nemůže konkrétně specifikovat den v měsíci a den v týdnu (např.: pátek 13.),

jelikož pokud jsou uvedeny konkrétní hodnoty v polích den v měsíci a den v týdnu, cron mezi těmito hodnotami použije logickou spojku NEBO, což bude mít za následek, že úloha poběží každý pátek a každý 13. den v měsíci.

Rovněž základní implementace cronu neumožňuje uživateli spouštět programy např. každý poslední den v měsíci (toto je vyřešeno v některých pokročilejších implementacích pomocí direktivy „L“). Mezi další omezení patří nemožnost nakonfigurovat běh např. mezi 23. a 3. hodinou v noci pomocí jednoho záznamu v konfiguračním souboru.

```
#pokud chce uživatel, aby běžela úloha každých 30 s, musí použít
#program sleep
* * * * * root kazdych_30.sh
* * * * * root sleep 30; kazdych_30.sh

#toto nemá očekávaný efekt, úloha poběží každý pátek a každý
#13. den v měsíci o půlnoci
0 0 13 * 5 root patek_trinacteho.sh

#uživatel chce aby úloha běžela mezi 23. a 3. hodinou v noci
#toto nebude fungovat
0 23-3 * * * root nocni_skript.sh

#tento problém lze však vyřešit rozdělením úlohy do dvou záznamů
0 23 * * * root nocni_skript.sh
0 0-3 * * * root nocni_skript.sh
```

Tyto omezení jsou však v praxi zanedbatelná, protože se týkají jen velmi okrajových situací.

1.1.2.2 Druhy konfiguračních souborů

Plánovač cron rozlišuje několik druhů konfiguračních souborů.

Systémový crontab Systémový crontab je umístěn v /etc/crontab, tento konfigurační soubor obsahuje pole, ve kterém se specifikuje uživatel, pod kterým úloha poběží. Podle [1], musí být tento konfigurační soubor vlastněn uživatelem root a nesmí mít právo zápisu pro skupinu nebo ostatní.

Tento soubor slouží jako hlavní konfigurační soubor pro celý systém a nemůže být upravován běžným uživatelem. Do tohoto konfiguračního souboru se většinou ukládají záznamy týkající se celého systému.

Adresář /etc/cron.d/ V tomto adresáři můžeme nalézt crontab soubory, které využívají např. nainstalované balíčky, pokud jim nevyhovují přednastavené adresáře /etc/cron.daily, atd. (viz dále) a potřebují větší kontrolu nad spouštěním.

Konfigurační soubory v tomto adresáři musí být, stejně jako v předchozím případě, ve vlastnictví uživatele root a nesmějí mít práva zápisu pro skupinu a ostatní. Stejně jako v předchozím /etc/crontab souboru, i v tomto souboru se uvádí uživatel, pod kterým úloha poběží.

Podle [1], by systémový administrátor neměl využívat tuto složku, ale hlavní konfigurační soubor /etc/crontab.

Uživatelské crontab soubory Uživatelské crontab soubory se liší umístěním v různých operačních systémech. Umístění je podle [4] následující:

Mac OS X /usr/lib/cron/tabs/
FreeBSD/OpenBSD/NetBSD /var/cron/tabs/
CentOS/Red Hat/RHEL/Fedora/Scientific Linux /var/spool/cron/
Debian / Ubuntu Linux /var/spool/cron/crontabs/
HP-UX Unix /var/spool/cron/crontabs/
IBM AIX Unix /var/spool/cron/

V těchto adresářích jsou umístěné konfigurační soubory, jejichž název odpovídá uživatelskému jménu, z tohoto důvodu se nemusí v těchto konfiguračních souborech uvádět uživatel, pod kterým úloha poběží. Ačkoliv je uživatel vlastníkem těchto konfiguračních souborů, nemůže je běžný uživatel měnit přímo, ale jen s pomocí programu crontab.

Toto chování je podle [1] vynuceno tím, že složku obsahující uživatelské konfigurační soubory může upravovat jen superuživatel nebo skupina crontab. Program crontab má poté nastaven setgid bit, který mu umožní, aby byl spouštěn s právy skupiny crontab.

Program crontab umožňuje vytvářet, upravovat, vypisovat a mazat uživatelské crontab soubory. Při upravovaní konfiguračního souboru, program crontab otevře konfigurační soubor v editoru, který je nastaven v proměnné prostředí EDITOR popř. VISUAL. Pokud není ani jedna proměnná nastavena, program crontab otevře konfigurační soubor programem /usr/bin/editor, což je symbolický link na editor nastavený operačním systémem.

Program crontab dále kontroluje přítomnost souborů /etc/cron.allow a /etc/cron.deny. Tyto soubory obsahují na každém řádku jedno uživatelské jméno. Pokud existuje soubor /etc/cron.allow, uživatel musí být uveden v tomto souboru, aby mohl použít program crontab. Pokud existuje soubor /etc/cron.deny uživatel nesmí být uveden v tomto souboru, aby mohl použít program crontab.

1.1.3 Další rozšíření

1.1.3.1 Složky /etc/cron.hourly/, /etc/cron.daily/, atd.

Pokud uživatel nechce nastavovat konfigurační soubory crontab a spokojí se základním nastavením, může využít složek /etc/cron.hourly/, /etc/cron.daily/,

/etc/cron.weekly/ a /etc/cron.monthly/. Tyto složky tu existují jako alternativa ke konfiguračním souborům crontab.

Uživatel do těchto složek pouze nakopíruje skript či program a ten se bude spouštět z danou periodou. Přednastavené periody spouštění jsou: jednou za hodinu, denně, týdně a měsíčně. Program či skripty musí mít nastaveno právo na spuštění a musí být vlastněny uživatelem root. Plánovač cron či program anacron (viz dále) mají ve svých konfiguračních souborech příslušné záznamy, které pomocí programu run-parts spustí všechny programy a skripty v obsažené v těchto složkách.

1.1.3.2 Anacron

Jedna z nevýhod plánovače cron je předpoklad, že počítač běží neustále. Tento předpoklad je sice splněn u serverů, ale osobní počítače ho nesplňují. Pokud by si např. uživatel naplánoval každý týden v neděli o půlnoci zálohu důležitých dat na svém osobním počítači, ale neměl by v tuto dobu zapnutý počítač, plánovač cron by zálohu neprovedl.

Tento problém se snaží vyřešit program anacron. Princip je podobný jako u plánovače cron s tím rozdílem, že se u úloh nespecifikuje přesný čas spuštění úlohy, ale perioda, ve které se bude úloha spouštět.

Program anacron však není, na rozdíl od plánovače cron, softwarový démon, což znamená, že nečeká na pozadí na nějakou událost, ale pouze spustí programy, které má a poté skončí.

Program anacron je podle [5] spouštěn při různých událostech (start počítače, probuzení, připojení přenosného počítače do elektrické sítě), ale jinak je jeho spuštění přenecháno na plánovači cron. Konfigurační soubor programu anacron se jmenuje anacrontab a je umístěn v /etc/anacrontab.

Každá řádka konfiguračního souboru anacrontab obsahuje záznam, který obsahuje 4 pole: periodu spuštění ve dnech, prodlevu v minutách, unikátní identifikátor úlohy a příkaz ke spuštění, jak lze vidět na obrázku 1.2. Jelikož se perioda spuštění nastavuje v celých dnech, nejmenší perioda spouštění úlohy je jeden den. Jelikož by se mohlo stát, pokud by počítač neběžel dlouhou dobu, že by se po spuštění programu anacron spouštělo velmi úloh najednou, existuje tu možnost prodlevy, která alespoň částečně rozloží běh úloh v čase.

Anacron při spuštění pro každý záznam v konfiguračním souboru zkontroluje složku /var/spool/anacron/, ve které jsou soubory obsahující časová razítka posledního spuštění dané úlohy.

Tyto soubory se jmenují stejně jako unikátní identifikátor v konfiguračním souboru. Anacron zkontroluje příslušné časové razítko a ověří zdali byla úloha spuštěna v posledních n dnech. Pokud ne, anacron nejprve počká tolik minut, kolik je specifikováno v konfiguračním souboru u dané úlohy a poté ji spustí. Nakonec aktualizuje příslušné časové razítko.

Obrázek 1.2: Příklad záznamu v konfiguračním souboru anacrontab, převzato z [6]

1.1.3.3 Spolupráce programů cron a anacron

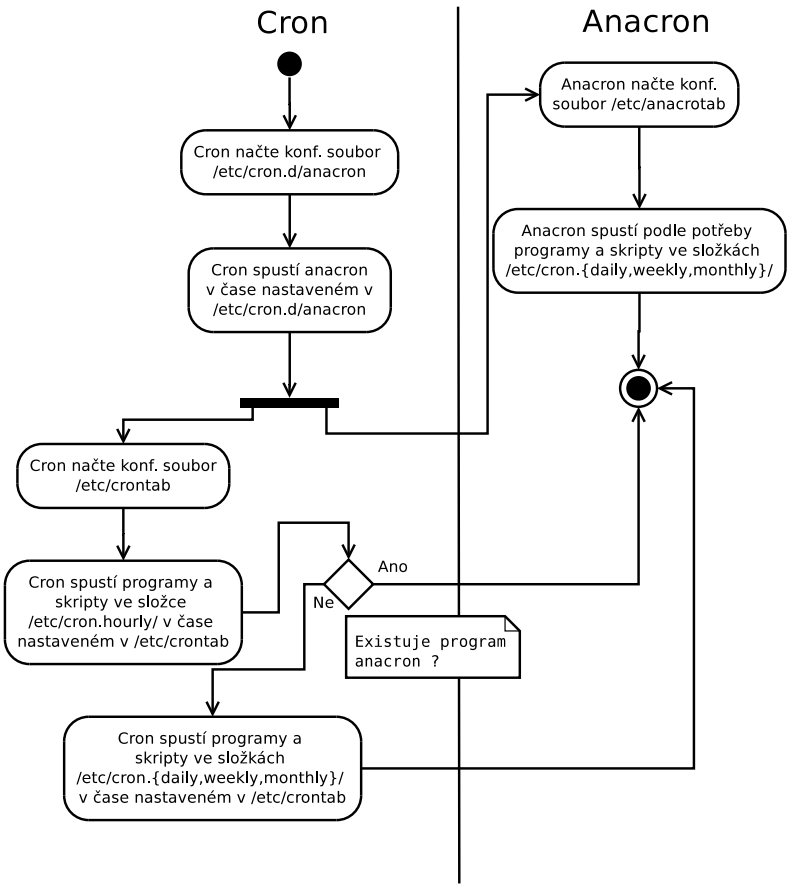
Jelikož se o složky /etc/cron.daily/, /etc/cron.weekly/ a /etc/cron.monthly/ starají oba programy cron i anacron, musí být zajištěno, že se dané úlohy spustí jen jednou v daném intervalu.

Jak lze vidět na obrázku 1.3, plánovač cron má v jednom ze svých konfiguračních souborů příslušný záznam, který jednou denně spouští program anacron. Program anacron poté načte svůj konfigurační soubor a podle potřeby spustí obsah složek /etc/cron.daily, /etc/cron.weekly a /etc/cron.monthly. Zabránění duplicity spuštění je poté docíleno pomocí následujících řádku v konfiguračním souboru /etc/crontab:

```
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts
    --report /etc/cron.daily )
47 6 * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts
    --report /etc/cron.weekly )
52 6 1 * * root test -x /usr/sbin/anacron || ( cd / && run-parts
    --report /etc/cron.monthly )
```

Jak si lze všimnout, plánovač cron testuje přítomnost spustitelného souboru /usr/sbin/anacron, pokud tento soubor není nalezen, cron spustí programy a skripty ze složek /etc/cron.daily/, /etc/cron.weekly/ a /etc/cron.monthly/.

Starost o složku /etc/cron.hourly/ je však nechána zcela na plánovači cron, protože anacron neumožňuje rozlišovat periodu menší než jeden den.



Obrázek 1.3: Diagram spolupráce programů cron a crontab

1.1.3.4 Další implementace plánovače cron

V příkladech výše byla zmiňována implementace cronu, která běží na distribucích Debian, Ubuntu, atd. Tato implementace se jmenuje **vixie-cron**. Je to implementace založená na SysV cronu a podporuje například použití vlastních proměnných prostředí v uživatelských crontabech [7].

Mezi další implementace cronu patří například:

cronie je podle [7] odnož vixie-cronu, kterou používá distribuce Fedora. Tato implementace umí vše, co vixie-cron a navíc má v sobě vlastní implementaci anacronu.

dcron je podle [7] implementace, která se snaží být jednoduchá, elegantní a bez nepotřebných funkcí. Např. nepodporuje žádné proměnné prostředí uvnitř crontabů a všechny programy jsou spouštěny pomocí shellu /bin/sh.

fcron se podle [7] snaží nahradit vixie-cron a anacron. Implementuje několik dalších funkcí, které dále rozšiřují množnosti spouštění a plánování úloh.

1.2 Genetické algoritmy

1.2.1 Princip genetických algoritmů

Genetické algoritmy vynalezl v 60. letech John Holland na Michiganské univerzitě. Jeho původním záměrem nebylo navrhnou algoritmus k řešení konkrétního problému, ale formálně studovat fenomén adaptace, tak jak se vyskytuje v přírodě [8].

V roce 1975 vydal knihu s názvem „Adaptation in natural and artificial systems“, ve které popsal jak aplikovat principy přirozeného výběru na optimalizační úlohy a sestavil první genetický algoritmus [9].

Genetické algoritmy pracují na principu evoluce, což znamená, že do příštích generací se dostávají nejsilnější jedinci. Abstrakci tohoto biologického principu vyžívají genetické algoritmy, které postupně vyvíjí množinu řešení, kde na konci zůstanou ta řešení, která jsou nejlepší. Jelikož špatná řešení jsou zavrhována a dobrá řešení jsou vybírána mezi rodiče, kteří zplodí lepší potomky, řešení postupně konvergují ke globálnímu optimu.

1.2.2 Základní pojmy a průběh algoritmu

Genetické algoritmy přebírají terminologii z biologie, používají se termíny jako např. alely, geny, chromozomy, fenotyp, genotyp, jedinec, fitness, generace a populace. Kvůli zjednodušení se některé pojmy vypouští a překrývají.

V následujících kapitolách se budou používat tyto termíny:

alely tvoří množinu hodnot, které může nabývat gen.
gen je jeden článek chromozomu.
chromozom reprezentuje zakódované řešení problému.
jedinec je jedno řešení problému. Je definován svých chromozomem.
populace je soubor všech jedinců v dané generaci.
fitness určuje míru adaptace na prostředí a šanci na předání genetické informace dalším generacím.

Při vytváření genetického algoritmu je nutné zvolit způsob jak zakódovat řešení problému do formy, která bude vhodná pro zpracování genetickým algoritmem. Mezi běžnými formami kódování jsou například řetězce, vektory z $\mathbb{R}^n$ či stromy.

Ačkoliv lze narazit na různé genetické algoritmy, jejich standardní průběh je podle [10] následující:

1. Vynuluj hodnotu počítadla generací t=0.

2. Náhodně vygeneruj počáteční populaci P(0).
3. Vypočítej ohodnocení (fitness) každého individua v počáteční populaci P(0).
4. Vyber dvojice individuí z populace P(t) a vytvoř jejich potomky P'(t).
5. Ohodnoť nově vytvořená individua v P'(t).
6. Vytvoř novou populaci P(t + 1) z původní populace P(t) a množiny potomků P'(t).
7. Zvětši hodnotu počítadla generací o jedničku (t:= t + 1).
8. Vypočítej ohodnocení (fitness) každého individua v populaci P(t).
9. Pokud t je rovno maximálnímu počtu generací nebo je splněno jiné ukončovací kritérium, vrať jako výsledek populaci P(t); jinak pokračuj krokem číslo 4.

1.2.3 Hodnotící (fitness) funkce

Hodnotící (fitness) funkce je funkce, které určuje míru kvality řešení. Tato funkce je aplikována na každého jedince v populaci a určí tak jedince, kteří mají šanci na předání genetických informací do dalších generací.

1.2.4 Výběr jedinců

Výběr jedinců je operace, která vybere vhodné jedince z populace, kteří se budou účastnit křížení. Tato operace musí zohledňovat fitness daného jedince, to znamená, že jedinci s vyšší fitness mají větší šanci se zkřížit. Existuje několik druhů výběrových metod, mezi základní patří:

Ruletová selekce je jedna ze základních výběrových metod. Velikost fitness jedince představuje výseč na ruletě. Kulička, která je posléze vhozená do rulety má větší pravděpodobnost, že skončí ve výseči jedince s vyšší fitness. Tato metoda je implementována tak, že velikost fitness určuje velikost intervalu v poměru k ostatním uvnitř intervalu $< 0; 1 >$, následně se vygeneruje náhodné číslo z intervalu $< 0; 1 >$. Vybrán bude ten jedinec, do jehož intervalu spadá náhodně vygenerované číslo.

Turnajová selekce je metoda, ve které se vybere n jedinců z populace a vybere se ten s nejvyšší hodnotou fitness. Tuto metodu opakujeme tolikrát dokud nenaplníme budoucí populaci. Pomocí vstupního parametru n (tj. kolik jedinců se účastní turnaje) můžeme ovlivnit selekční tlak.

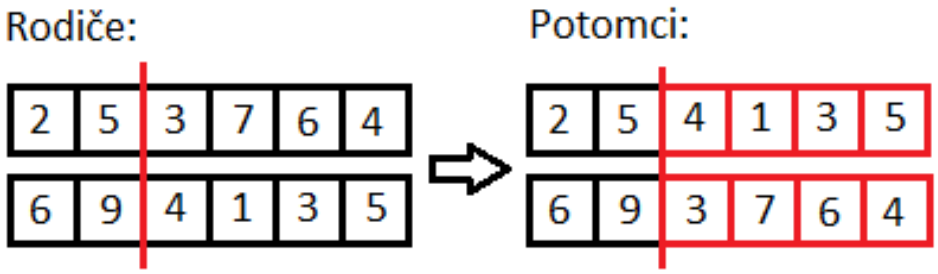
Elitismus není přímo výběrová metoda, ale pouze rozšíření ostatních. Z populace se vybere několik jedinců s nejvyšší fitness a ti se převedou do nové populace. Tím je zajištěno, že tito jedinci nemohou vymřít.

1.2.5 Operátory křížení a mutace

1.2.5.1 Operátor křížení

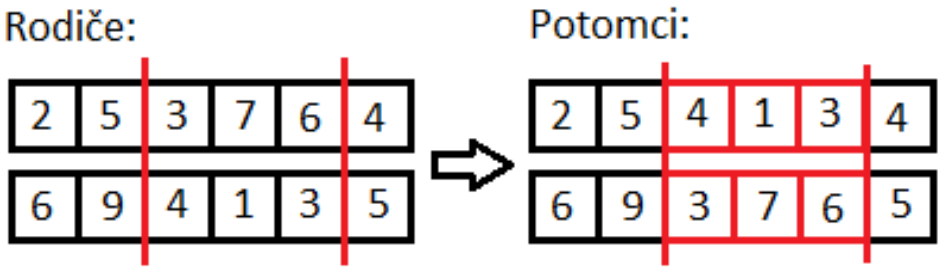
Operátor křížení je funkce, která na vstupu obdrží dva rodiče, ty následně zkříží a výstupem jsou dva potomci. Existuje mnoho metod křížení, mezi základní patří:

Jednobodové křížení je nejjednodušší metoda křížení, tato metoda náhodně zvolí místo v chromozomu. Od tohoto místa si pak oba rodiče vymění geny, jak lze vidět na obrázku 1.4.



Obrázek 1.4: Jednobodové křížení

Dvoubodové křížení je rozšíření jednobodového křížení. V této metodě se náhodně zvolí dva body. Geny mezi těmito body budou vyměněny mezi rodiči, jak lze vidět na obrázku 1.5.



Obrázek 1.5: Dvoubodové křížení

Uniformní křížení je metoda, která každý gen s určitou pravděpodobností vymění vzájemně mezi rodiči.

Podle [8] neexistuje žádná ideální univerzální metoda křížení, vždy záleží na konkrétním problému. Jako vhodné metody křížení se často volí dvoubodové či uniformní křížení s pravděpodobností 0.7-0.8.

1.2.5.2 Operátor mutace

Po aplikování operátorů křížení následuje aplikace operátoru mutace. Mutace náhodně změní jeden či více genů na jiný. Mutace zanáší do chromozomu nové genetické informace a zabraňuje tak uvíznutí v lokálním optimu. Existuje několik metod mutace, např.:

Flip Bit je metoda mutace, která se používá u chromozomů, které jsou vyjádřeny jako řetězec binárních hodnot (tj. posloupnost jedniček a nul). Tato metoda náhodně změní jeden nebo více bitů na opačnou hodnotu.

Uniformní mutace je metoda, která se zvolenou pravděpodobností změní každý gen na hodnotu, která je vybrána z definovaného intervalu. Jak už název napovídá, všechny hodnoty z tohoto intervalu mají stejnou pravděpodobnost, že budou vybrány.

Oba operátory křížení a mutace se neaplikují vždy, ale pouze s určitou pravděpodobností. Neexistuje žádná univerzální pravděpodobnost, která by byla optimální pro všechny genetické algoritmy. Lze například vytvořit genetické algoritmy, jejichž výstupem budou parametry pro „hlavní“ genetický algoritmus (tzv. meta-optimalizace). Jak je uvedeno v [8], pravděpodobnost křížení se zpravidla pohybuje v rozmezí 0.75-0.95 a pravděpodobnost mutace v rozmezí 0.005-0.01.

Zatímco pravděpodobnost křížení se volí poměrně velká, aby docházelo ke zvyšování fitness budoucích jedinců, pravděpodobnost mutace se volí velmi malá, protože při velké pravděpodobnosti mutace by se změnil genetický algoritmus na náhodný vyhledávací algoritmus. Nulová mutace by však mohla zapříčinit uvíznutí v lokálním optimu.

Kapitola 2

Návrh nástroje

2.1 Současná řešení

V současné době neexistuje ucelený nástroj, který by prováděl predikci běhu naplánovaných úloh, vizualizaci běhu úloh, detekci zátěžových špiček a hledání optimálního rozložení úloh. Existují však nástroje, které řeší některé dílčí úlohy.

2.1.1 Nástroje řešící predikci běhu

Webový nástroj **CRON tester** [11] napsaný v jazyce PHP umí z jednoho řádku konfiguračního souboru predikovat až 100 následujících iterací zadané úlohy. Tento jednoduchý nástroj slouží hlavně ke kontrole, zda zadaný záznam v konfiguračním souboru odpovídá požadovanému výsledku. Při testování vlastní implementace predikce však bylo zjištěno, že tento nástroj obsahuje několik chyb v predikci.

Program **Schedule-Cron-Events**, který je napsán v Perlu, umí rovněž predikovat běh následující iterace. Navíc umí vypsat čas běhu již spuštěných úloh z minulosti.

2.1.2 Nástroje řešící vizualizaci

Mezi nástroje vizualizující běh cron úloh patří např. cronviz [12], který je napsán v jazyce Ruby. Tento nástroj umí ze zadaného konfiguračního souboru crontab vygenerovat graf, který poslouží uživateli k lepší představě, kdy běží jaká úloha.

2.2 Výběr programovacího jazyka

Mezi programovací jazyky vhodné pro implementaci byly zvažovány jazyky C++ a Python. C++ jako zástupce kompilovaných jazyků a Python jako zá-

stupce interpretovaných jazyků.

Kompilovaný jazyk je takový jazyk, který musí být před spuštěním překladačem přeložen ze zdrojového kódu do strojového kódu. Interpretovaný jazyk Python je překladačem přeložen pouze do tzv. mezikódu a tento mezikód je dále interpretován.

Pro implementaci jsem zvolil skriptovací jazyk Python a to hlavně kvůli standardnímu modulu datetime, jelikož velkou část nástroje bude tvořit manipulace s časem. Modul datetime poskytuje snadnou a efektivní práci s časem. Mezi další výhody jazyka Python patří podle [13] vysoká rychlost vývoje, stejný zdrojový kód zapsaný v C++ je 5-10x delší než v jazyce Python.

Nevýhoda jazyku Python oproti C++ je např. nižší rychlost a vyšší paměťová náročnost, tyto faktory však nejsou při implementaci nástroje, který zajišťuje statickou analýzu, tolik důležité.

2.3 Zvolení metadat cron úloh pro analýzu

Jako metadata pro analýzu a optimalizaci naplánovaných úloh se používá tří parametrů: duration, shift a weight.

duration Tento parametr určuje jak úloha dlouho běží. Je to jediný parametr, který je vypočítán automaticky externím nástrojem cronwrapper. Hodnotou parametru je celé číslo, odpovídající délce běhu v sekundách. Pokud nebude tato hodnota uvedena použije se výchozí hodnota z konfiguračního souboru nástroje.

shift Tento parametr je využit při optimalizaci úloh a určuje maximální hodnoty posunu dané úlohy. Hodnotou jsou dvě celá čísla oddělena dvojtečkou, určující hranice intervalu posunu v minutách. Pokud je uvedeno jen jedno celé číslo, znamená to, že maximální hodnota posunu vzad i vpřed je stejná. Uživatel musí tuto hodnotu nastavit manuálně, pokud ji neuvede, použije se výchozí hodnota z konfiguračního souboru nástroje.

weight Tento parametr je rovněž využit při optimalizaci úloh a při vykreslování zátěžového grafu. Hodnotou je celé číslo určující náročnost úlohy, kde vyšší číslo znamená vyšší náročnost. Uživatel musí opět nastavit tuto hodnotu manuálně, pokud ji neuvede použije se výchozí hodnota z konfiguračního souboru nástroje.

Všechny tyto parametry jsou uvedeny jako komentář za naplánovanou úlohou v konfiguračním crontab souboru. Nastavení parametru se provádí znakem „=“, na pořadí parametrů nezáleží, stejně tak na přítomnosti bílých znaků. Viz následující ukázka.

```
#parametr shift v následujícím záznamu má uvedenu pouze jednu
#hodnotu, tato hodnota bude převedena na ekvivalentní zápis
```

```
#shift=-10:10
0 0 * * 0 klejcpet bckp.sh # duration=60,shift = 10, weight = 10
* * * * * klejcpet date #shift=-5:10,duration=1,weight=1
```

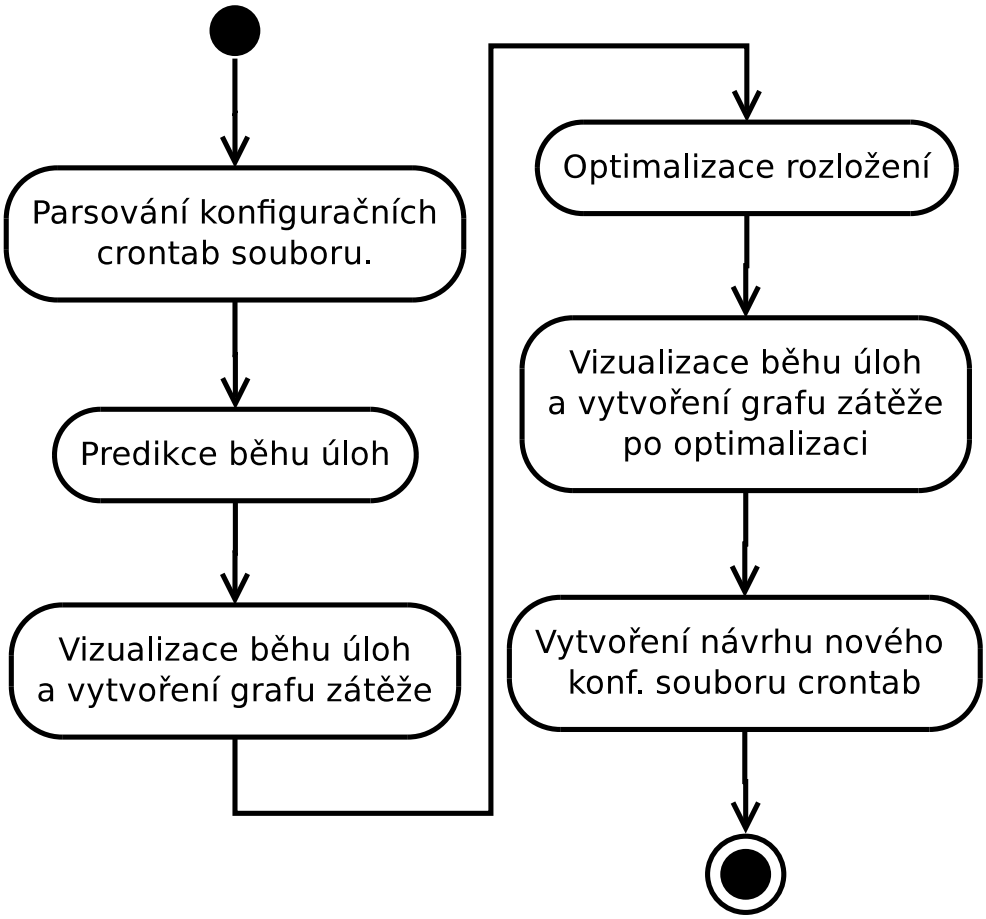
2.4 Diagram analýzy

Nástroj se skládá z několika částí každý řešící jednu část analýzy. Jak je patrné z obrázku 2.1 nejprve se parsují konfigurační soubory crontab, pro analýzu jsou zejména důležité informace, kdy úloha poběží a také metadata o úloze. Dále se na základě těchto vyparsovaných dat provede predikce běhu na dobu specifikovanou v konfiguračním souboru nástroje.

V další části se provádí vizualizace běhu a sestavení zátěžového grafu. Graf s běhy úloh slouží pro lepší představu, kdy úlohy běží, jak často a zdali nedochází k překryvům (úloha ještě nedoběhla, ale už se začíná spouštět další). Zátěžový graf poté slouží k vizualizaci zátěžových špiček.

V následující části se provádí optimalizace běhu pomocí genetického algoritmu. Následně se znovu vizualizuje toto nové, optimální rozložení, znovu se sestaví zátěžový graf a všechny grafy jsou poté dostupné ve výstupní zprávě k porovnání.

Na závěr proběhne návrh nových a optimalizovaných konfiguračních souborů crontab.



Obrázek 2.1: Diagram analýzy úloh

Kapitola 3

Implementace

3.1 Analýza běhu úloh

Analýza naplánovaných úloh probíhá ve dvou částech: nejprve se z konfiguračních crontab souborů pomocí predikce vytvoří zdrojová data a ty se následně vizualizují.

Pro vizualizaci dal byl zvažován program Gnuplot a Python modul matplotlib [14]. Pro vizualizaci byl nakonec zvolen modul matplotlib, jelikož poskytuje přímou integraci s nástrojem na rozdíl od programu Gnuplot, který by musel být volán externě.

3.1.1 Predikce běhu

Predikce běhu probíhá následujícím způsobem. Nejprve se musí převést hodnoty, které jsou v konfiguračním souboru na místech označujících minuty, hodiny, dne v měsíci, měsíce a dne v týdnu převést na seznam celých čísel. Viz následující ukázka.

„*“ u pole s minutami se převede na seznam [0,1,2,...,58,59]
rozsah 5-9 u pole s hodinami se převede na seznam [5,6,7,8,9]
výčty dnů v měsíci 2,6 se převede na seznam [2,6]
zápis každých 5 minut „*/5“ se převede na seznam [0,10,15,...,50,55]

Tyto seznamy poté slouží jako vstup pro generátory.

Generátor je funkce, která se chová jako iterátor. Iterátor je objekt, který implementuje metodu next(), která vrací následující prvek. Generátor vypadá jako běžná funkce, ale místo klíčového slova return používá klíčové slovo yield, jeho hlavní výhodou je, že si pamatuje svůj vnitřní stav.

Chování generátoru je podle [15] následující: při prvním zavolání metody next() se provede tělo generátoru až do klíčového slova yield, poté generátor vrátí hodnotu, uspí se a čeká na další zavolání metody next(). Po příštím zavolání metody next() se opět provede část kódu až do klíčového slova yield.

Toto chování pokračuje dokud se nevyčerpá tělo generátoru nebo dokud se nenarazí na klíčové slovo return.

Generátory jsou použity pro získání následující minuty, hodiny, atp. Pseudokód funkce, pro získání dalšího času dané úlohy vypadá takto:

Pseudokód 3.1: Funkce pro získání dalšího času úlohy

```
try:
    posuň minutu
except:
    resetuj generátor minut
    posuň minutu
    try:
        posuň hodinu
    except:
        resetuj generátor hodin
        posuň hodinu
        try:
            posuň den
        except:
            resetuj generátor dní
            posuň den
            try:
                posuň měsíc
            except:
                resetuj generátor měsíců
                posuň měsíc
                posuň rok
```

Jak je patrné z ukázkového pseudokódu 3.1, nejprve se zkusí posunout minuta, pokud je však vyčerpán generátor minut, resetuje se, posune se minuta a zároveň se zkusí posunout hodina. Generátor hodin může být také vyčerpán a tak vyvolá posun dnů, změna minuty tedy může vyvolat postupnou iteraci všech generátorů, např. přidání jedné minuty k datu 23:59 31.12.2014.

Jelikož je nejmenší frekvence opakování naplánovaných úloh jeden rok, nemusí se roky nijak kontrolovat a postačí jejich prostá inkrementace o 1.

Predikce příštího běhu úlohy se ukázala jako poměrně netriviální problém. A to zejména kvůli velkému množství kombinací, kterými se dá vyjádřit běh úlohy

Mezi největší problémy patřilo implementování znaku „*“ a zápisů typu „*/5“ u položky s dny, jelikož se počet dnů musí pokaždé přepočítávat v závislosti na aktuálním měsíci a roku.

Dalším problémem v implementaci bylo ošetření všech kombinací zápisu týkající se dnů a dnů v týdnu. Tento problém se rozpadl na 4 podproblémy:

1. Pokud je na místě dnů a dnů v týdnu znak „*“, je posunut den a na den v týdnu se vůbec nepohlíží.

2. Pokud je na místě dnů nějaká hodnota jiná než „*“, např. rozsah, výčet, atd., je den posunut po vytvořeném seznamu dnů a na den v týdnu se nepohlíží.

3. Pokud je v položce dnů znak „*“ a na místě s dny v týdnu nějaká hodnota jiná než znak „*“, je v nekonečném cyklu nejprve posunut den. Na tento den poté proveden dotaz, zdali je to nějaký ze zadaných dnů v týdnu. Pokud ano, cyklus končí. Pokud ne, je den opět posunut. A poté znovu proveden dotaz. Při každém posunu dnů však může nastat situace, že je to poslední den ze seznamu dnů, tím se musí vyvolat posun měsíce. Posun měsíce však může vyvolat i posun roku.

4. Posledním případem je situace, kdy dny i dny v týdnu nejsou znaky „*“, ale jiné hodnoty (rozsahy, výčty, konkrétní dny, atd.). Jak již bylo uvedeno v 1.1.2.1, pokud není na místě dnů a dnů v týdnu znak „*“, je mezi těmito položkami použit operátor NEBO. Tento poslední případ je rozšíření předchozí situace. Nejprve je vytvořen pomocný seznam všech dnů v daném měsíci. Poté se provede posun po tomto pomocném seznamu všech dní a tento pomocný den je porovnáván na shodu s definovaným seznamem dnů v týdnu. Pokud není nalezena shoda, je tento den dále porovnáván s definovaným seznamem dní. Pokud opět není nalezena shoda, posune se pomocný den a celý proces je zopakován. Stejně jako v předchozím případě, posun pomocného dnu může vyvolat další posuny.

Posunování dalších minut, hodin a měsíců bylo triviální, protože tyto položky mají vždy fixní počet.

V konfiguračním souboru nástroje se poté specifikuje na kolik dní se analýza predikuje. Tento počet dní se přičte k aktuálnímu datu a tím se získá cílové datum predikce. Pseudokód predikce vypadá takto:

Pseudokód 3.2: Vytváření intervalů úloh

```
while datum_ulohy <= cilove_datum:
    start=ziskej_dalsi_cas()
    konec=start+delka behu
    uloz_interval_behu(start,konec)
```

Jak si lze všimnout z pseudokódu 3.2, funkce pro získání dalšího času vrací startovní čas dané úlohy, k tomuto času se přičte délka běhu v sekundách (parametr duration v konfiguračním crontab souboru). Tímto se získá interval běhu úlohy, ten je uložen pro pozdější vykreslení.

Výstupem této části nástroje je seznam objektů CCron. Každý objekt CCron obsahuje informace o dané úloze, např. vyparsované záznamy z konfiguračního souboru, především zápisy, kdy bude úloha spuštěna a kdo úlohu

spouští, název spouštěného programu, délku úlohy, náročnost úlohy, maximální hodnoty posunu, atd. Dále obsahuje seznam intervalů, které určují přesný čas, kdy bude úloha běžet.

3.1.2 Grafy běhu úloh a zátěže

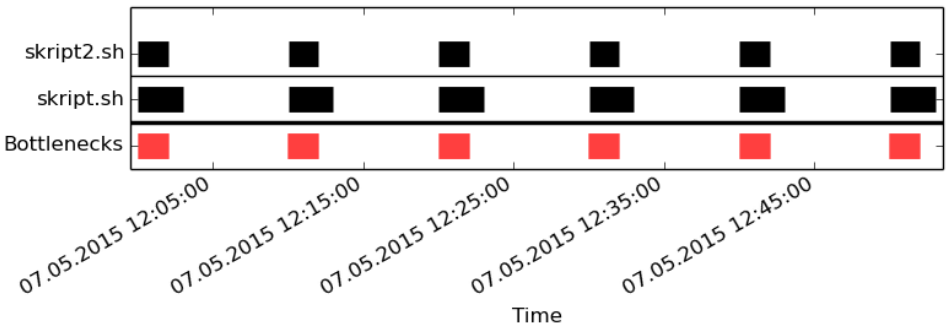
3.1.2.1 Graf běhu úloh

Výstup předchozí části nástroje vytvořil intervaly, kdy úloha běží. Tyto intervaly jsou použity pro vykreslení běhu úlohy, jak lze vidět v pseudokódu 3.3. Vzhled výsledného grafu lze vidět na obrázku 3.1. Na ose x je vynesen čas běhu a na ose y jsou názvy jednotlivých úloh.

Pokud je v určitém čase překročen práh (threshold) zátěže, vykreslí se rovněž čára znázorňující intervaly, která představují úzká hrdla (bottlnecks).

Pseudokód 3.3: Funkce pro vykreslení intervalů úloh

```
for cron in cronList:
        for interval in cron.cronJobList:
                vykresli(interval)
```



Obrázek 3.1: Graf běhu úloh

3.1.2.2 Zátěžový graf

Zátěžový graf vizualizuje sumu zátěže jednotlivých úloh v určitém čase, jak lze vidět na obrázku 3.3. Na ose x je vynesen čas a na ose y je vidět suma zátěže. Pokud zátěž překročí v určitém čase práh (threshold) zátěže, je poté zvýrazněna červenou barvou.

Pro sestavení zátěžového grafu je nutné sečíst zátěže překrývajících se úloh v určitém čase. Toho je docíleno pomocí datové struktury Interval Tree.

Interval Tree je vylepšený vyvážený binární vyhledávací strom, který umožňuje vkládat a mazat intervaly a rovněž klást dotazy jako jsou: které intervaly

se překrývají se zadaným bodem či zadaným intervalem. Sestavení intervalového stromu zabere $O(n \log n)$ času a zabere $O(n)$ paměti. Složitost dotazu je poté $O(\log n + m)$, kde m je počet nalezených překrývajících se intervalů.

Pro implementaci intervalového stromu byl použit Python modul intervaltree 2.0.4 [16], který je implementován nad AVL-stromem a podporuje další operace jako např.: průniky, sjednocení a rozdíly mezi stromy, atd.

Data pro zátěžový graf jsou vytvořena následujícím způsobem:

1. Zjistí se počáteční čas první úlohy, která byla predikována a koncový čas poslední úlohy.

2. Tyto časy se odečtou a získá se délka celé predikce. Ta by měla zhruba odpovídat počtu dnům, které byly nastaveny jako délka predikce v konfiguračním souboru. Pokud ale např. všechny analyzované úlohy běží jedenkrát za týden ve stejný čas a uživatel zvolil délku predikce 3 dny. Délka predikce bude odpovídat jen menšímu časovému intervalu, ve kterém běží tyto úlohy. Za podmínky, že úlohy poběží v definovaném 3 denním intervalu.

3. Interval délky predikce se rovnoměrně rozloží na n testovacích intervalů. Počet testovacích intervalů je vypočten jako $\frac{delka\_predikce}{delka\_testovaciho\_intervalu}$.

4. Tyto testovací intervaly jsou poté použity pro konstrukci intervalového stromu, každý testovací interval má na začátku sumu váhy 0.

5. Pro každý naplánovaný interval, ve kterém běží daná úloha, je proveden dotaz, který zjišťuje, které testovací intervaly protíná.

6. Výstupem tohoto dotazu je 0 až n testovacích intervalů. Do každého výsledného testovacího intervalu je poté přičtena hodnota zátěže úlohy, která je protíná.

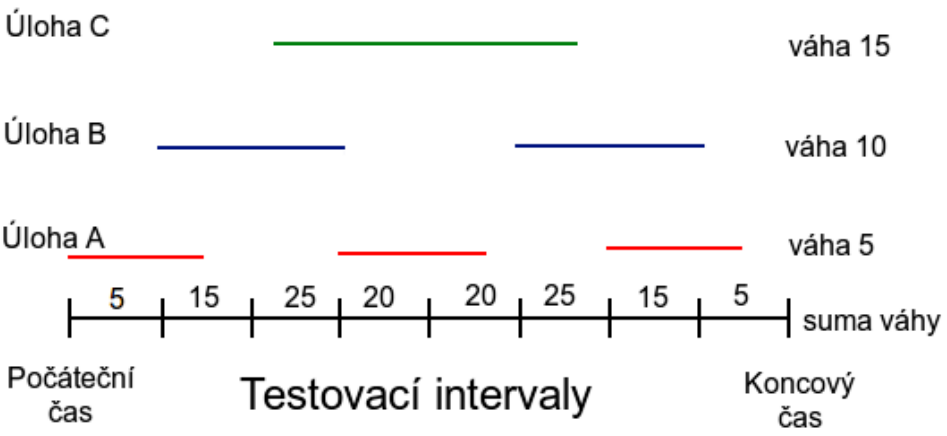
7. Na konci tohoto procesu je v každém testovacím intervalu uložena váha všech protínaných intervalů úloh.

Výstup tohoto algoritmu ilustruje obrázek 3.2.

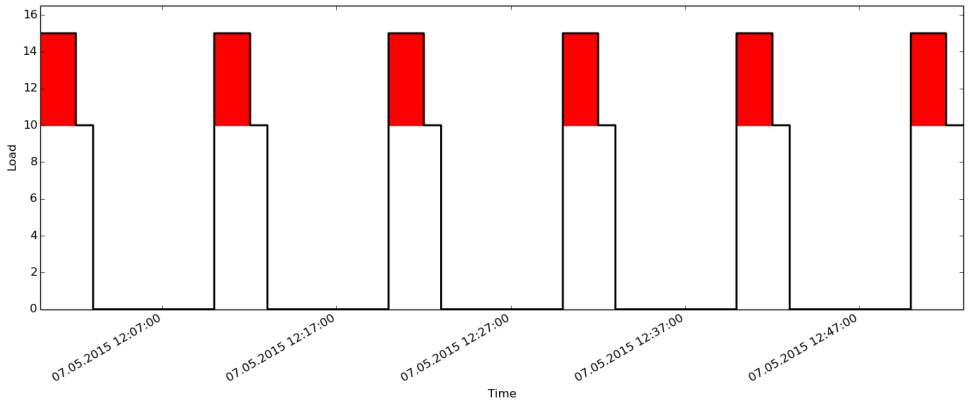
Počáteční a koncové body testovacích intervalů a jejich příslušné váhy se poté uloží do seznamu. V tomto seznamu se dále hledají intervaly, které překročily zátěžový práh. Intervaly, které překročí zátěžový práh jsou poté vykresleny do grafu běhu úloh 3.1 a označují úzká hrdla (bottlenecks).

3.2 Optimalizace běhu úloh

Úkolem optimalizace v tomto nástroji je najít takové rozložení úloh, jejichž běhy tvoří, co nejrovnoměrnější zátěž. K vyřešení tohoto problému byl použit genetický algoritmus.



Obrázek 3.2: Testovací intervaly

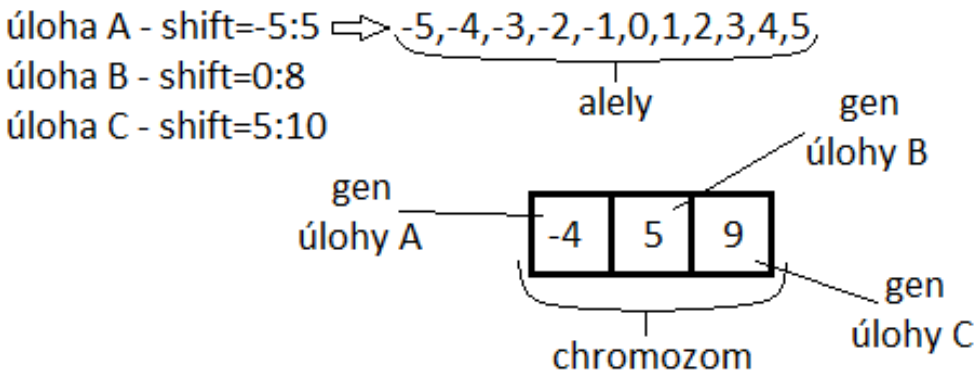


Obrázek 3.3: Zátěžový graf

Výhodou genetických algoritmů je snadnost implementace, možnost aplikace na širokou škálu problémů a získání přijatelně dobrého výsledku v přijatelném čase.

Pro implementaci genetických algoritmů byl použit framework DEAP [17]. Tento framework poskytuje pohodlné a rychlé implementování evolučních a genetických algoritmů.

Pro implementaci optimalizace byl pro zakódování řešení zvolen vektor, jehož délka odpovídá počtu analyzovaných úloh. Jak lze vidět na obrázku 3.4, v implementaci optimalizace úloh alela odpovídá intervalu celých čísel, jehož hranice tvoří hodnoty maximálního možného posunu úlohy. Gen představuje posun úlohy o n minut. Chromozom je poté reprezentován jako vektor těchto posunů, kde hodnota na i-té pozici představuje posun i-té úlohy.



Obrázek 3.4: Reprezentace chromozomu, genu a alel

3.2.1 Hodnotící (fitness) funkce

Fitness funkce, která byla použita v optimalizaci úloh, dostane na vstup vektor posunů jednotlivých úloh. Následně posune všechny cron úlohy o daný počet minut. Nakonec spočítá směrodatnou odchylku zátěže v testovacích intervalech. Testovací intervaly se vytváří a vyplňují stejným způsobem jako při vytváření zátěžového grafu 3.3.

Směrodatná odchylka se spočítá jako: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$. Kde n je počet počet testovacích intervalů, x_i je hodnota zátěže v i-tém testovacím intervalu a $\bar{x}$ je průměrná hodnota zátěže.

Genetický algoritmus se poté pokouší o minimalizaci směrodatné odchylky a tím pádem o nejrovnoměrnější rozložení. V ideálním případě by ve všech testovacích intervalech byla stejná zátěž, tím pádem nulová směrodatná odchylka.

3.2.1.1 Optimalizace hodnotící funkce

Jelikož je tato hodnotící funkce volána velmi často, při velké časové náročnosti hodnotící funkce může optimalizace trvat velmi dlouhou dobu. Z tohoto důvodu byl běh hodnotící funkce několikrát optimalizován.

Nejprve byl vylepšen algoritmus, který vyplňoval testovací intervaly hodnotami zátěže. Z původního naivního $O(n * k)$ algoritmu, kde n byl počet testovacích intervalů a k celkový počet intervalů úloh, byl algoritmus vylepšen pomocí struktury Interval Tree na složitost $O(k(\log n + m))$. Kde n a k mají stejný význam jako v předchozím případě a m je počet nalezených testovacích intervalů.

Dále bylo optimalizováno ohodnocování populace. Ohodnocení populace bylo původně realizováno pomocí funkce map. Funkce map, je funkce, která na vstupu obdrží dva parametry: funkci a iterátor. Následně na každý prvek iterátoru aplikuje funkci zadanou v parametru.

Optimalizace bylo dosaženo pomocí Python modulu multiprocessing, který poskytuje paralelní verzi funkce map. Její paralelní verze rozdělí iterátor na části a ty zpracovává současně v několika procesech. Touto optimalizaci se dosáhlo několikanásobného zrychlení výpočtu.

Původní implementace na osobním počítači Lenovo E430 s následujícími parametry: počet úloh - 150, celkový počet intervalů úloh - 1963, počet generací - 100, velikost populace - 100, délka predikce - 1 den, trvala přibližně 20 minut.

Vylepšena implementace na stejném počítači se stejnými parametry trvala okolo 6 minut. Bylo tak dosaženo více jak trojnásobného zrychlení.

3.2.2 Výběr jedinců

Jelikož standardní ruletová selekce nelze použít pro minimalizační problémy nebo pro jedince jejichž fitness může být menší nebo rovno než nula, byla pro implementaci optimalizace zvolená turnajová selekce. Velikost turnaje lze nastavit v konfiguračním souboru.

3.2.3 Operátory křížení a mutace

3.2.3.1 Operátor křížení

Pro implementaci optimalizace byla zvolena možnost použití jednobodového, dvoubodového i uniformního křížení. Metoda křížení se volí pomocí příslušné direktivy v konfiguračním souboru. Pro uniformní křížení je v konfiguračním souboru dodatečná direktiva, která určuje pravděpodobnost výměny každého genu.

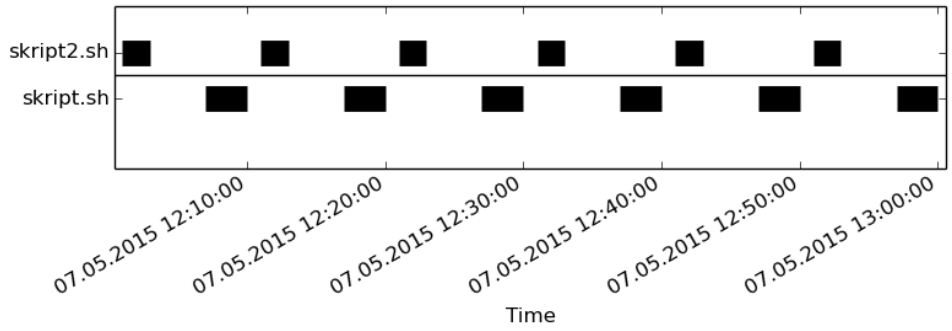
3.2.3.2 Operátor mutace

Pro optimalizaci rozložení úloh byla zvolena uniformní metoda mutace. Interval, ze kterého se náhodně vybírá hodnota je určen hodnotou minimálního a maximálního posunu dané úlohy. Pravděpodobnost změny každého genu je $1/n$, kde n je velikost chromozomu.

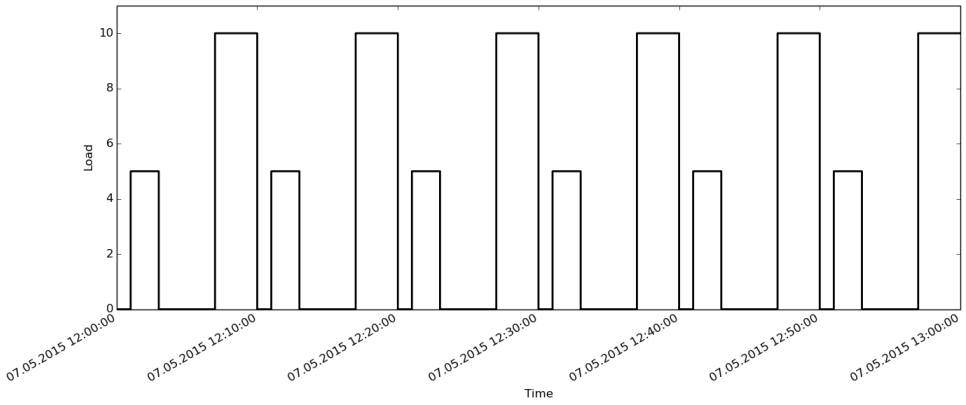
3.3 Výsledek optimalizace

Výstupem genetického algoritmu je jedinec s nejlepším fitness. To je takový jedinec, jehož chromozom obsahuje geny (posuny úloh), které dosahují nejrovnoměrnějšího zatížení (má nejmenší směrodatnou odchylku zatížení).

Tyto posuny z chromozomu jedince se následně aplikují na cron úlohy. Z takto posunutých cron úloh jsou poté sestaveny grafy běhu a zátěže stejně jako v sekcích 3.1.2.1 a 3.1.2.2. Optimalizovanou variantu grafu běhu úloh 3.1 lze vidět na obrázku 3.5, optimalizovanou variantu grafu zátěže 3.3 lze vidět na obrázku 3.6.



Obrázek 3.5: Graf běhu úloh po optimalizaci



Obrázek 3.6: Zátěžový graf po optimalizaci

Jak lze vidět na obrázku 3.6, optimalizace pomohla rozložit zátěž a snížit tak maximální hodnotu zátěže.

Následně se z chromozomu nejlepšího jedince sestaví optimalizovaný konfigurační crontab soubor. Sestavení nového konfiguračního souboru se ukázalo jako problémové místo.

Výsledek není vždy úplně v souladu s výstupem optimalizace, např. při vytváření nového konfiguračního souboru se předpokládá délka měsíce vždy 31 dní. Kvůli tomuto zjednodušení se může úloha posunout např. z 30. na 31. června. Tímto dojde k situaci, kdy je úloha naplánovaná na den, který nikdy nenastane. Úloha se tedy nikdy nespustí. Výstupní crontab soubor by měl být vždy ručně zkontrolován kvůli těmto chybám.

Pokud si však v praxi uživatel přeje spouštět úlohu jednou měsíčně, nastavuje její běh obvykle na první den v měsíci. Jelikož se tedy v praxi nenastavují obvykle úlohy ke konci měsíce, tento problém se bude týkat jen výjimečných případů.

Dále muselo být vyřešeno, jak bude naloženo s kombinací dnů a dnů v týdnu při posunu optimalizované úlohy na další dny. Postup posunu je následující:

1. Pokud je na místech dnů a dnů v týdnu znak „*“ neposunou se nikam. Je ale zkontrolováno, zdali by posun nezapříčinil případný posun měsíce.

2. Pokud je na místě dnů výčet, rozsah, atd. a na místě dnů v týdnu znak „*“ posunou se jen dny. Rovněž je kontrolováno, zdali posun nezapříčiní posun měsíce.

3. Pokud je na místě dnů znak „*“ a na místě dnů v týdnu výčet, rozsah, atd. jsou posunuty dny v týdnu. V tomto případě se měsíc nikdy neposune.

4. Pokud je na místech dnů a dnů v týdnu výčet, rozsah, atd. jsou posunuty jak dny, tak dny v týdnu. Zde je rovněž kontrolován případný posun měsíce.

Návrh nového konfiguračního crontab souboru lze poté najít ve výstupní zprávě.

Výstupní zpráva je vygenerovaný HTML dokument, který obsahuje grafy běhu úloh a zátěže před a po optimalizaci. Výstupní zpráva také obsahuje odkaz na logovací soubor, kde lze najít podrobnosti o analýze.

Součástí výstupní analýzy je porovnání několika měřených hodnot před a po optimalizaci. Mezi tyto parametry patří: nejmenší a největší zátěž v testovacích intervalech a směrodatná odchylka zátěže. Tyto hodnoty slouží k porovnání kvality optimalizace.

Kapitola 4

Testování

4.1 Testování prediktoru

Predikce běhů je stěžejní část, která generuje data, se kterými se pracuje v dalších částech nástroje, byl proto kladen důraz na správnost výsledků. Jelikož chyba, která vznikne v části s predikcí, by se velmi špatně hledala později, byl vytvořen testovací skript, který testuje data vůči webovému nástroji Cron-tester.

Ačkoliv nástroj Cron-tester ve výjimečných případech nevracel správné výsledky, bylo díky tomuto nástroji odhaleno a opraveno několik chyb.

4.2 Testování optimalizace

Jako testovací data posloužila reálná data dodaná vedoucím práce. Dodaná byla pouze část dat, která se týká časů, kdy úlohy poběží. K analýze jsou však potřebná další data, zejména délka běhu úlohy, náročnost úlohy a hodnoty maximálního posunu. Tyto dodatečná data byla vygenerována pomocí následujícího BASH skriptu.

```
while read line
do
        LEN=$((RANDOM%50 + 1))
        WEIGHT=$((RANDOM%50 + 1))
        SHIFT=$((RANDOM%50 + 1))
        echo "$line sleep $LEN # duration=$LEN, \
        weight=$WEIGHT, shift=0:$SHIFT" >> vystupni_data
done < vstupni_data
```

Tento skript doplnil náhodně generovaná metadata do dat dodaných vedoucím práce.

Data dodaná vedoucím práce obsahují 145 úloh, jejichž celkový počet běhů je přibližně 5000 za jeden den. Jedná se tedy o úlohy, které jsou spouštěny velmi často, průměrně se každá úloha spustí 44x za den.

Jako hlavní ukazatel kvality optimalizace byla zvolena směrodatná odchylka zátěže, maximální hodnota zátěže a minimální hodnota zátěže.

Cílem testů bylo najít vhodné výchozí parametry pro genetický algoritmus a ověřit vliv různých parametrů na kvalitu řešení.

4.2.1 Podmínky pro testování

Testování optimalizace probíhalo na osobním počítači Lenovo E430 s procesorem Intel Core i3 3110M a 4GB operační paměti.

Pro konzistentní podmínky testování byla predikce vždy prováděna mezi předem určenými daty: 4.5.2015 12:00 - 5.5.2015 12:00. Výchozí parametry pro genetický algoritmus byly následující:

délka testovacího intervalu 1 minuta
velikost populace 10 jedinců
počet generací 10
typ křížení uniformní s pravděpodobností 0.7 na záměnu každého genu
pravděpodobnost křížení 0.75
pravděpodobnost mutace 0.75
velikost turnaje 3

4.2.2 Typy testů

Mezi testované parametry byly zařazeny následující parametry:

- metoda křížení
- velikost turnaje při turnajové selekci
- pravděpodobnost křížení
- pravděpodobnost mutace
- velikost populace
- počet generací

Všechny testy, s výjimkou testů testující velikost populace a počet generací, byly prováděny 3x. Testy testující velikost populace a počet generací byly prováděny jen jednou z důvodů časové náročnosti. Z naměřených hodnot byl proveden aritmetický průměr.

4.2.3 Výsledky testů

Výstupy testů a jejich vliv na kvalitu optimalizace lze vidět na grafech v příloze C.

Při testování bylo zjištěno, že velký vliv na kvalitu optimalizace má parametr ovlivňující velikost populace. S malou populací se vygeneruje málo jedinců a tak se nedostatečně pokryje prostor všech řešení. Malé populaci, nepomůže ani velký počet generací, protože řešení brzy konverguje k lokálnímu optimu mezi vygenerovanými jedinci a následným křížením se řešení již nezlepší.

Při nastavení velké populace, ale malého počtu generací, nastane situace, kde pravděpodobně existují velmi dobrá řešení v populaci, ale díky malému počtu generací, genetický algoritmus nestihne zkřížit dobrá řešení a tak nedojde ke konvergenci ke globálnímu optimu.

Nastavení velké populace a malého počtu generací je však výhodnější než opačný případ, tj. nastavení malé populace a velkého počtu generací. Tato varianta je však paměťově náročnější.

Jelikož prvky vektoru jsou vybírány z intervalů, jejichž hranice tvoří hodnoty maximálních posunů úloh, velikost populace by měla být přímo úměrná velikostem těchto intervalů. Tzn. že se zvětšujícím se parametrem shift u úloh, by se měla zvětšovat i populace.

Některé výsledky testů nejsou v souladu s předpoklady, to může být způsobeno malým počtem opakování testů nebo povahou úlohy.

Z výstupních grafů lze identifikovat nejlepší parametry pro optimalizaci dat dodaných vedoucím práce s vygenerovanými náhodnými metadaty. Mezi vhodné parametry patří:

typ křížení uniformní s pravděpodobností 0.65 na změnu každého genu
velikost turnaje 5
pravděpodobnost křížení 0.95
pravděpodobnost mutace 0.06
velikost populace 100
počet generací 100

Při použití těchto parametrů pro optimalizaci testovacích dat dodaných vedoucím práce byla snížena směrodatná odchylka zátěže z původních 80.1 na 20.9. Došlo to tak téměř ke čtyřnásobnému zmenšení směrodatné odchylky.

Maximální zátěž se zmenšila z 355 na 131, klesla tedy přibližně 2.7x. Porovnání zátěžových grafů před a po optimalizaci, lze vidět v příloze D.

Závěr

Cílem této bakalářské práce bylo seznámit se s principem fungování plánovače cron a formátem konfiguračních crontab souborů. Dále na základě získaných znalostí vytvořit nástroj, který bude analyzovat běh naplánovaných úloh časovým plánovačem cron a pokusí se o nalezení optimálního rozložení úloh v čase.

Analýza byla realizována pomocí prediktoru, který predikuje další běhy naplánovaných úloh v budoucnu. Analýza se dále skládá z vizualizace úloh a sestavení zátěžového grafu, který slouží k vizuální detekci zátěžových špiček.

Optimalizace bylo dosaženo pomocí genetického algoritmu, který hledá optimální rozložení úloh. Z tohoto optimálního rozložení je posléze sestaven nový konfigurační soubor crontab.

Při testování nástroje nad daty dodanými vedoucím práce se povedlo, při vhodném nastavení parametrů pro genetický algoritmus, snížit směrodatnou odchylku zátěže téměř 4x. A také snížit maximální hodnotu zátěže přibližně 2.7x.

Cíle této bakalářské práce byly splněny. Při implementaci nástroje jsem prohloubil své znalosti v programování a lépe si osvojil principy operačního systému GNU/Linux.

Výhled do budoucna

Při implementaci tohoto nástroje nebylo implementováno několik dodatečných rozšíření. Tato rozšíření by mohla být implementována v budoucnu.

Např. nástroj neanalyzuje ani neoptimalizuje programy a skripty uložené ve složkách /etc/cron.daily/, atd. Dále nástroj neanalyzuje ani neoptimalizuje úlohy naplánované pomocí programu anacron.

Mezi další plánovaná rozšíření by mohla patřit optimalizace úlohy, ve které dochází k překryvům toho typu, kdy naplánovaná úloha ještě nedoběhne a již se spouští další iterace. Optimalizace tohoto překryvu by mohla spočívat v rozdělení úlohy na dvě. Tyto dvě samostatné úlohy by již šlo optimalizovat implementovaným genetickým algoritmem.

Mezi další vhodná rozšíření by mohlo patřit umožnění využívat zástupná makra @daily, @weekly, atd. Nebo možnost využívat tříznakových anglických názvů měsíců a dní.

Literatura

[1] Vixie, P.: *cron(8) Linux User's Manual*.

[2] TuxRadar [online]: *The anatomy of a crontab entry*. © 2009 Future Publishing Limited, [vid. 2015-04-30]. Dostupné z: `http://www.tuxradar.com/content/automate-linux-cron-and-anacron`

[3] pantz.org: *Cron and Crontab usage and examples [online]*. [cit. 2015-04-12]. Dostupné z: `http://www.pantz.org/software/cron/croninfo.html`

[4] NixCraft: *Linux / UNIX Crontab File Location [online]*. 2008, [cit. 2015-04-30]. Dostupné z: `http://www.cyberciti.biz/faq/where-is-the-crontab-file/`

[5] Community Help Wiki: *CronHowto [online]*. [cit. 2015-04-12]. Dostupné z: `https://help.ubuntu.com/community/CronHowto`

[6] TuxRadar [online]: *Anatomy of anacrontab*. © 2009 Future Publishing Limited, [vid. 2015-04-30]. Dostupné z: `http://www.tuxradar.com/content/automate-linux-cron-and-anacron`

[7] Gentoo Wiki: *Cron*. 2015, [cit. 2015-04-30]. Dostupné z: `https://wiki.gentoo.org/wiki/Cron`

[8] Mitchell, M.: *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN 0262631857.

[9] Sivanandam, S. N.; Deepa, S. N.: *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, první vydání, 2007, ISBN 354073189X, 9783540731894.

[10] Hynek, J.: *Genetické algoritmy a genetické programování*. Grada, 2008, ISBN 9788024726953.

[11] Schlitt, T.: *CRON Tester*. 2012. Dostupné z: `http://cron.schlitt.info/`

[12] Mack, D.: *cronviz*. GitHub, 2012. Dostupné z: `https://github.com/federatedmedia/cronviz`

[13] Python.org [online]: *Comparing Python to Other Languages*. ©2001-2015, [cit. 2015-04-30]. Dostupné z: `https://www.python.org/doc/essays/comparisons/`

[14] Hunter, J. D.: Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, ročník 9, č. 3, 2007: s. 90–95.

[15] Švec, J.: *Létající cirkus (10)*. Root.cz, 2002, [cit. 2015-04-30]. Dostupné z: `http://www.root.cz/clanky/letajici-cirkus-10/`

[16] Halbert, C.-L.: *intervaltree*. GitHub, 2015. Dostupné z: `https://github.com/chaimleib/intervaltree`

[17] Fortin, F.-A.; De Rainville, F.-M.; Gardner, M.-A.; aj.: DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, ročník 13, jul 2012: s. 2171–2175.

Příloha **A**

Seznam použitých zkratek

GNU GNU's not Unix

HTML HyperText Markup Language

BASH Bourne again shell

Příloha B

Uživatelská příručka

B.1 Instalace nástroje

Pro snadnou instalaci byl vytvořen Debian balíček, tento balíček obsahuje také manuálovou stránku. Instalace balíčku se provede následujícím příkazem v terminálu:

```
$ sudo dpkg -i cronalyzer_0.9-1_all.deb
```

Balíček lze také nainstalovat z grafického prostředí prostým spuštěním.

B.2 Konfigurační soubor nástroje

Vytvořený nástroj pro analýzu úloh se nastavuje pomocí konfiguračního souboru. Konfigurační soubor se uvádí jako argument skriptu. Pokud je nástroj nainstalován pomocí instalačního balíčku, výchozí konfigurační soubor se nachází v /etc/cronalyzer/config.conf. Pokud uživatel nainstaloval nástroj pomocí instalačního balíčku a chce využít výchozího konfiguračního souboru, stačí spustit nástroj jen jeho jménem v terminálu:

```
$ cronalyzer
```

B.2.1 Význam direktiv konfiguračního souboru

UserCrontabDirs označuje složku, ve které se budou hledat uživatelské crontab soubory. Lze zadat více složek, které je nutné oddělit znakem „:“.

OtherCrontabDirs označuje složku, ve které se budou hledat neuživatelské crontab soubory. Opět lze zadat více složek, oddělených znakem „:“.

IncludeMainCrontab určuje, zdali bude do analýzy zařazen soubor /etc/crontab.

IncludeCrondFolder určuje, zdali bude do analýzy zařazena složka /etc/cron.d/.

OutputDir určuje složku, kam se uloží výsledná analýza. Ta obsahuje grafy běhu a zátěže před a po optimalizaci a výstupní zprávu.

LogLevel určuje úroveň logování. Možnosti nastavení jsou debug nebo info. Možnost debug je podrobnější a zobrazuje např. vypočtené parametry.

RemoveLogOnStart tato direktiva určuje zda bude log soubor smazán při každém spuštění nástroje.

GraphName určuje název grafu běhu úloh.

WeightGraphName určuje název zátěžového grafu.

OnlyAnalysis určuje zdali bude provedena pouze analýza.

IncludeRareCronJobs určuje zda se do analýzy budou uvažovat i úlohy, které běží v méně než šesti měsících ročně.

PredictionDuration určuje délku predikce ve dnech.

ShowInteractiveImages určuje zda se obrázky zobrazí také v interaktivní formě.

Threshold určuje práh zátěže.

IgnoreShortCronJobs určuje zdali se budou ignorovat krátké úlohy.

IgnoreShortCronJobsLimit určuje délku úlohy, kterou musí úloha mít, aby byla zahrnuta do analýzy. Tato direktiva závisí na nastavení direktivy IgnoreShortCronJobs.

TimeFormatOnXAxis určuje časový formát na ose x.

LoadGraphApproximation určuje zdali bude proveden náhodný výběr z testovacích intervalů. Tato direktiva může zpřehlednit graf zátěže pokud obsahuje mnoho testovacích intervalů.

LoadGraphSampleSize velikost náhodného výběru do aproximace.

ParallelComputation určuje zdali se bude provádět výpočet genetického algoritmu paralelně.

IntervalLength určuje délku testovacího intervalu v minutách.

SizeOfPopulation určuje velikost populace genetického algoritmu.

NumberOfGenerations určuje počet generací genetického algoritmu.

MateOperator určuje operátor křížení. Povolené hodnoty jsou OnePoint, TwoPoint, Uniform.

UniformOperatorProbability určuje pravděpodobnost změny každého genu při křížení. Tato direktiva je závislá na direktivě MateOperator a bude mít efekt pouze v případě, že bude vybrána metoda křížení jako Uniform.

TournamentSize určuje velikost turnaje.

MutationProbability určuje pravděpodobnost mutace.

CrossoverProbability určuje pravděpodobnost křížení.

DefaultWeight určuje výchozí náročnost úlohy, pokud nebyla specifikována.

DefaultMaxShift určuje výchozí maximální posun úlohy, pokud nebyl specifikován.

DefaultMinShift určuje výchozí minimální posun úlohy, pokud nebyl specifikován.

DefaultDuration určuje výchozí délku úlohy, pokud nebyla specifikována.

B.3 Nástroj cronwrapper

Součástí balíčku je také nástroj cronwrapper. Tento nástroj měří délku běhu úlohy, která je zadána v argumentu. Délka běhu úlohy se poté zapíše do souboru. Nástroj se nastavuje pomocí konfiguračního souboru, který je umístěn v /etc/cronalyzer/cronwrapper.conf. Konfigurační soubor tohoto nástroje obsahuje jedinou direktivu FileWithDurations, jejíž hodnota obsahuje cestu k souboru, do kterého se budou zapisovat délky běhu úloh. Ukázka použití nástroje:

```
$ cronwrapper sleep 10
```

Nástroj cronwrapper zapisuje do výstupního souboru na každé řádce běh v celých sekundách a název spuštěného programu. Výše uvedená ukázka by vygenerovala následující řádku:

```
10 sleep 10
```

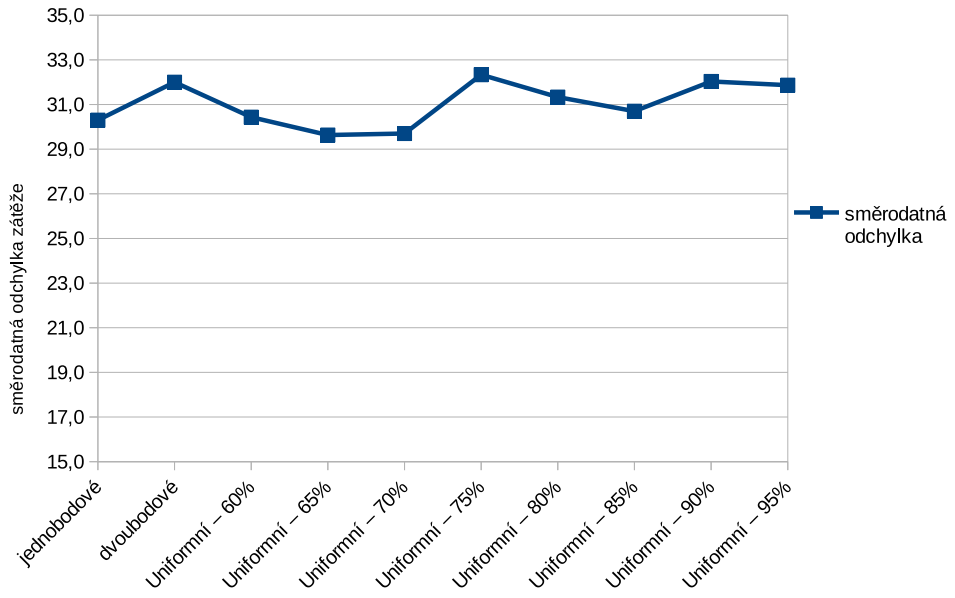
Nástroj cronwrapper se dá použít i v konfiguračním souboru crontab, např:

```
0 0 * * 0 klejcpet cronwrapper backup.sh
```

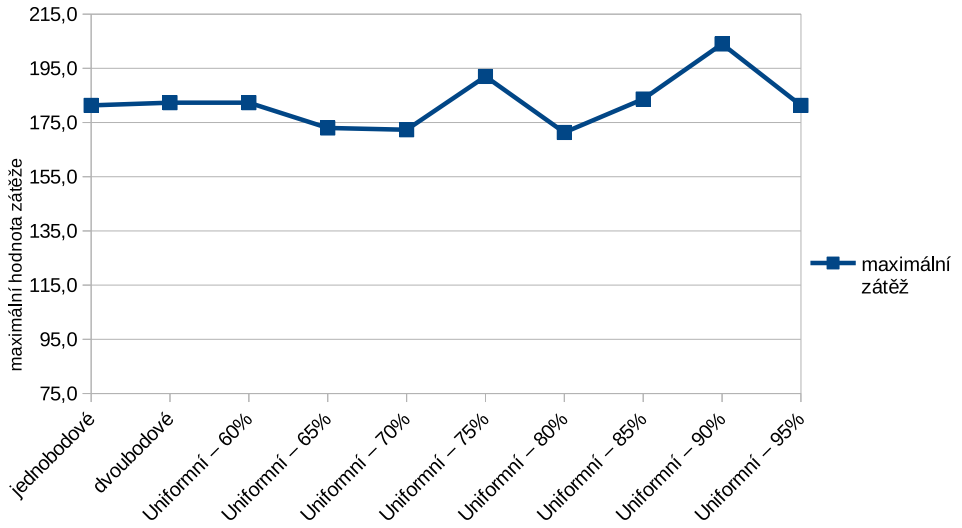
Výše uvedená ukázka by změřila čas běhu skriptu backup.sh a uložila by ho do souboru.

Příloha C

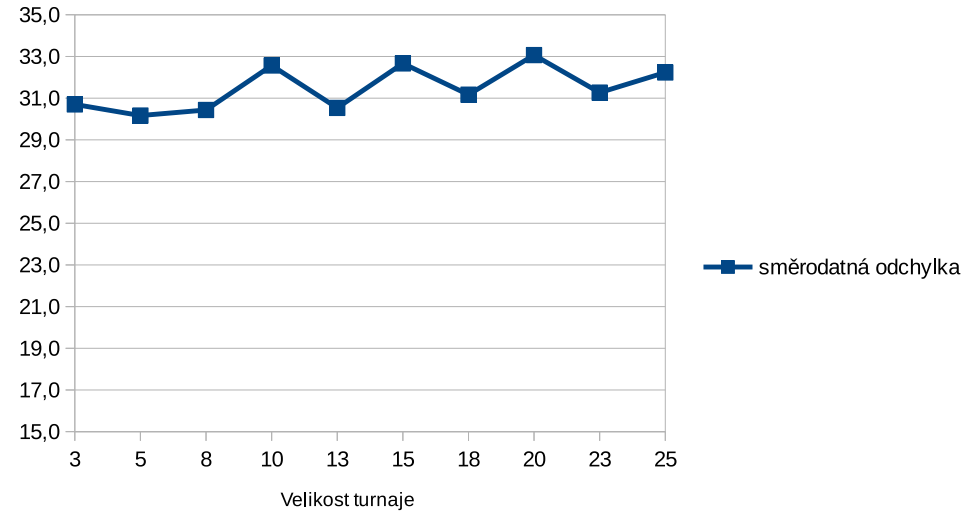
Výstupní grafy z testování



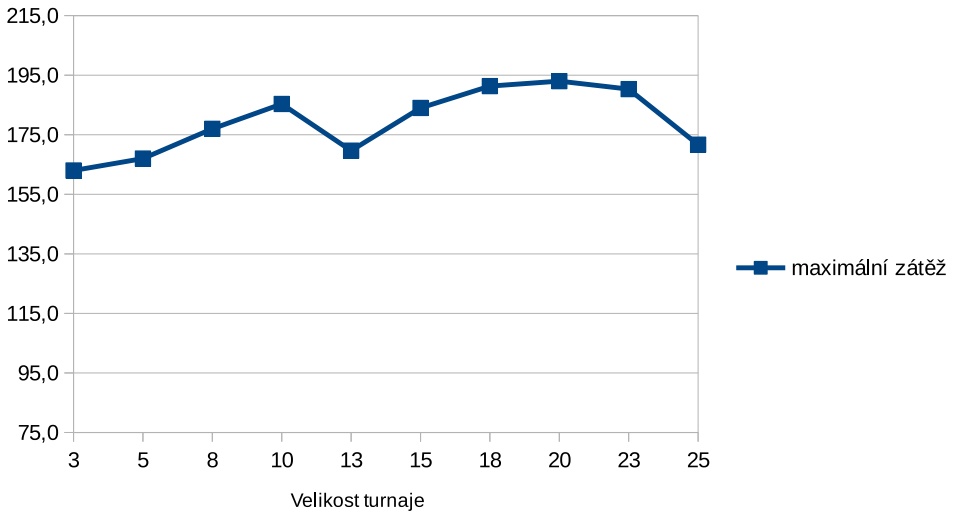
Obrázek C.1: Graf závislosti směrodatné odchylky zátěže na metodě křížení



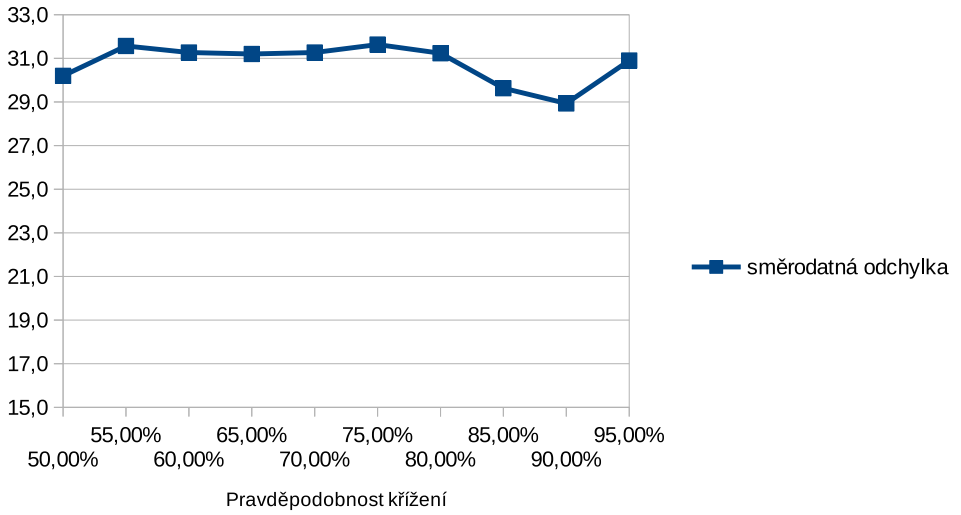
Obrázek C.2: Graf závislosti maximální zátěže na metodě křížení



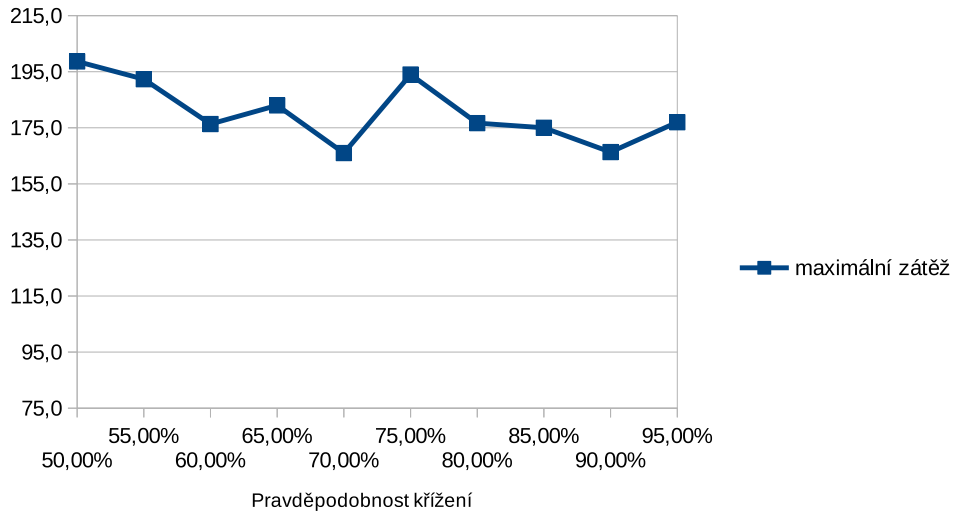
Obrázek C.3: Graf závislosti směrodatné odchylky zátěže na velikosti turnaje



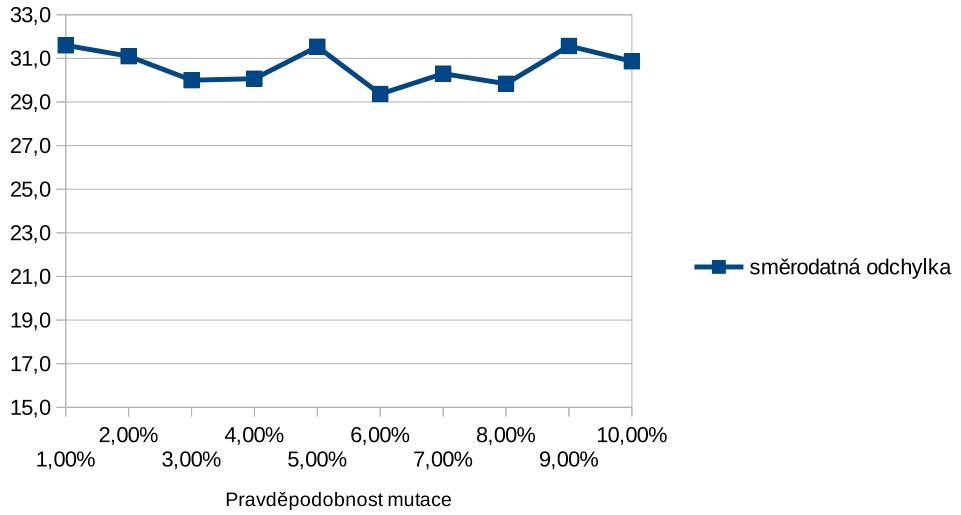
Obrázek C.4: Graf závislosti maximální zátěže na velikosti turnaje



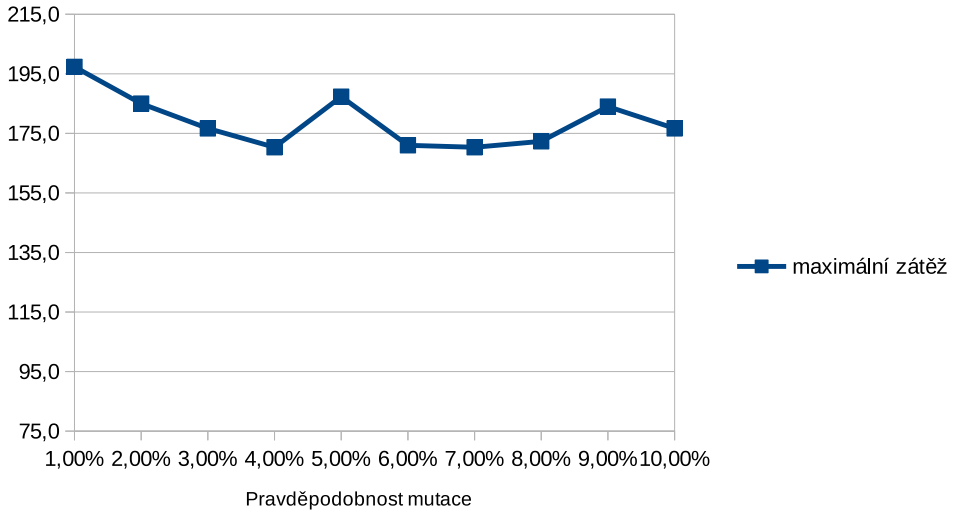
Obrázek C.5: Graf závislosti směrodatné odchylky zátěže na pravděpodobnosti křížení



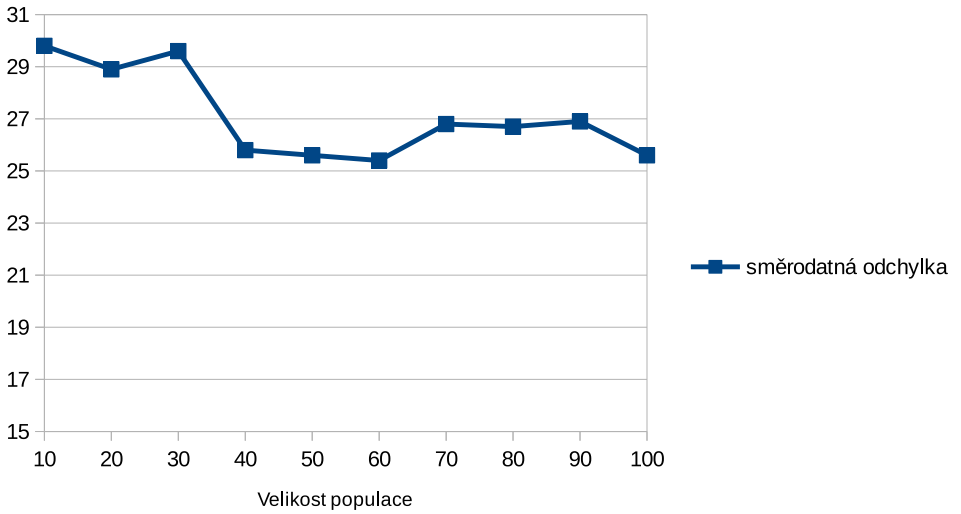
Obrázek C.6: Graf závislosti maximální zátěže na pravděpodobnosti křížení



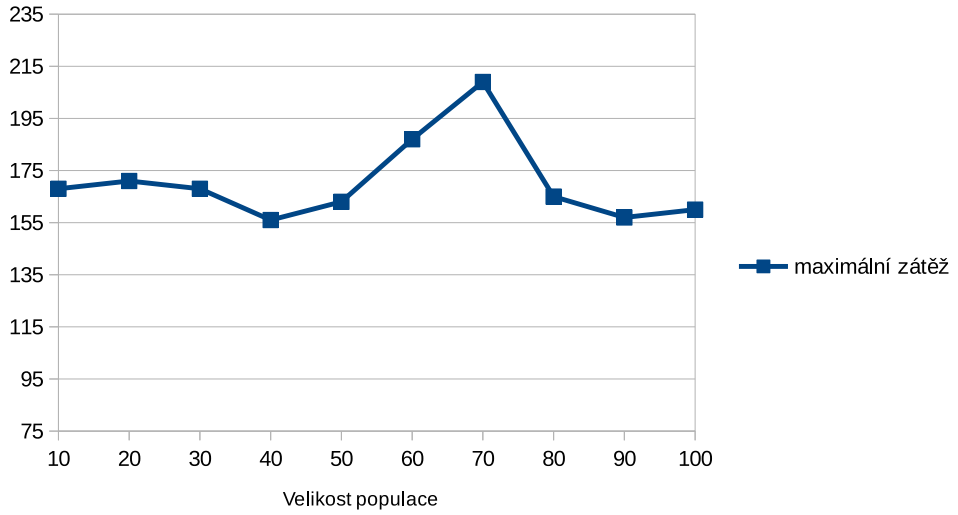
Obrázek C.7: Graf závislosti směrodatné odchylky zátěže na pravděpodobnosti mutace



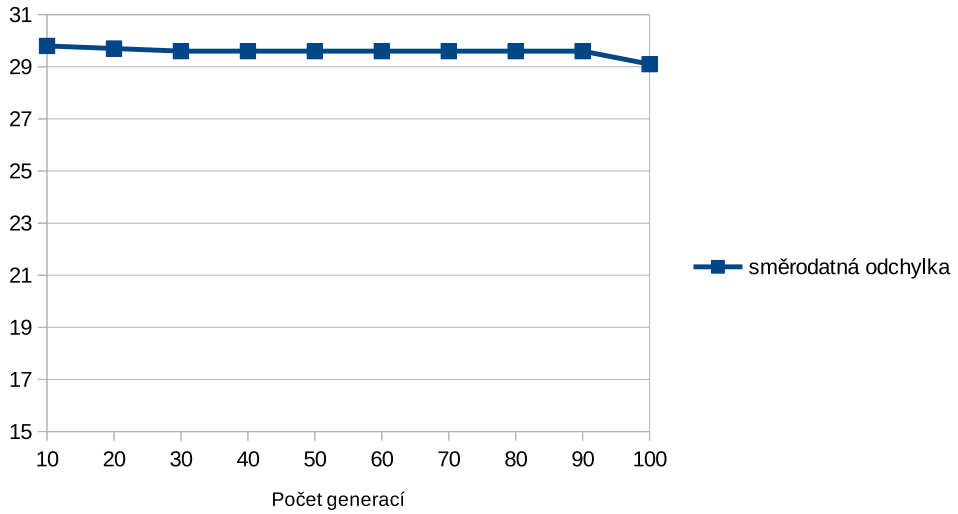
Obrázek C.8: Graf závislosti maximální zátěže na pravděpodobnosti křížení



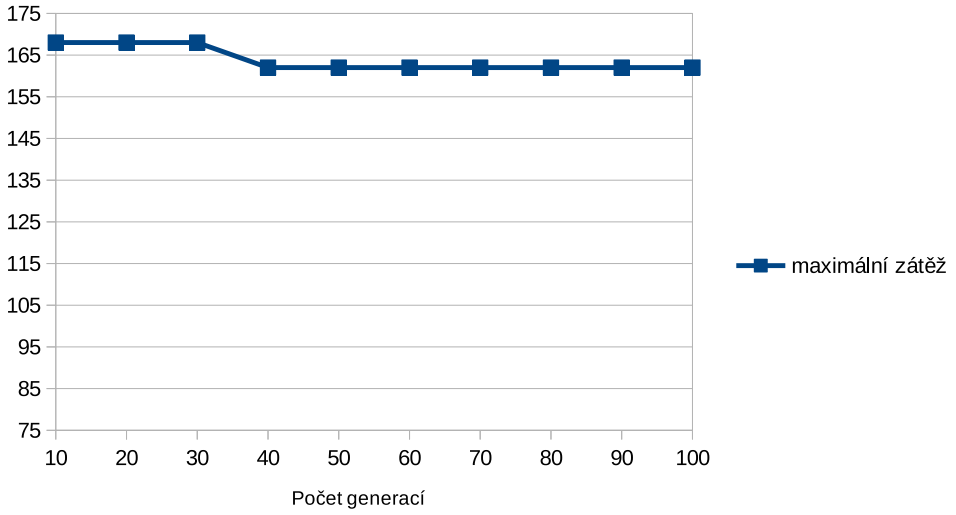
Obrázek C.9: Graf závislosti směrodatné odchylky zátěže na velikosti populace



Obrázek C.10: Graf závislosti maximální zátěže na velikosti populace



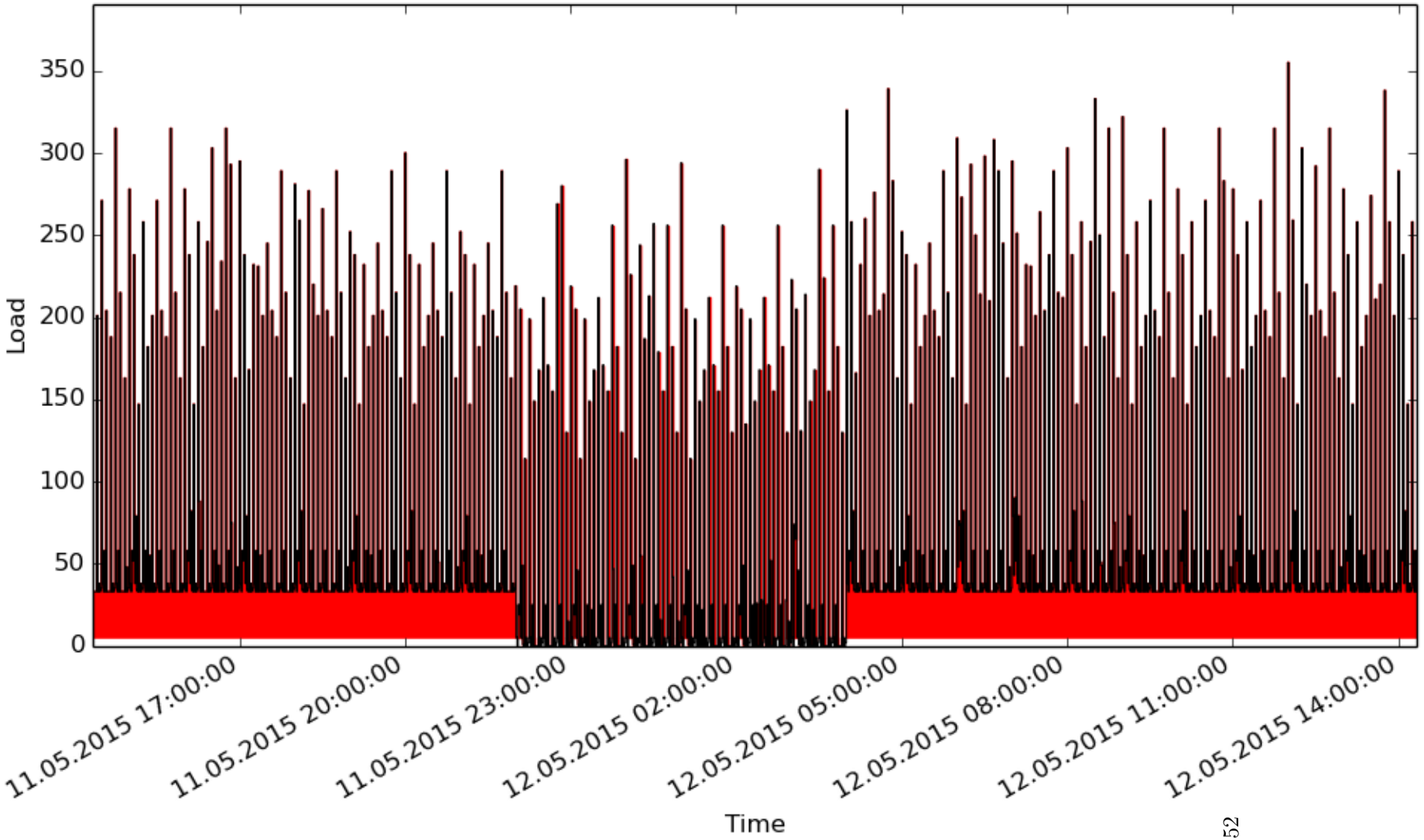
Obrázek C.11: Graf závislosti směrodatné odchylky zátěže na počtu generací



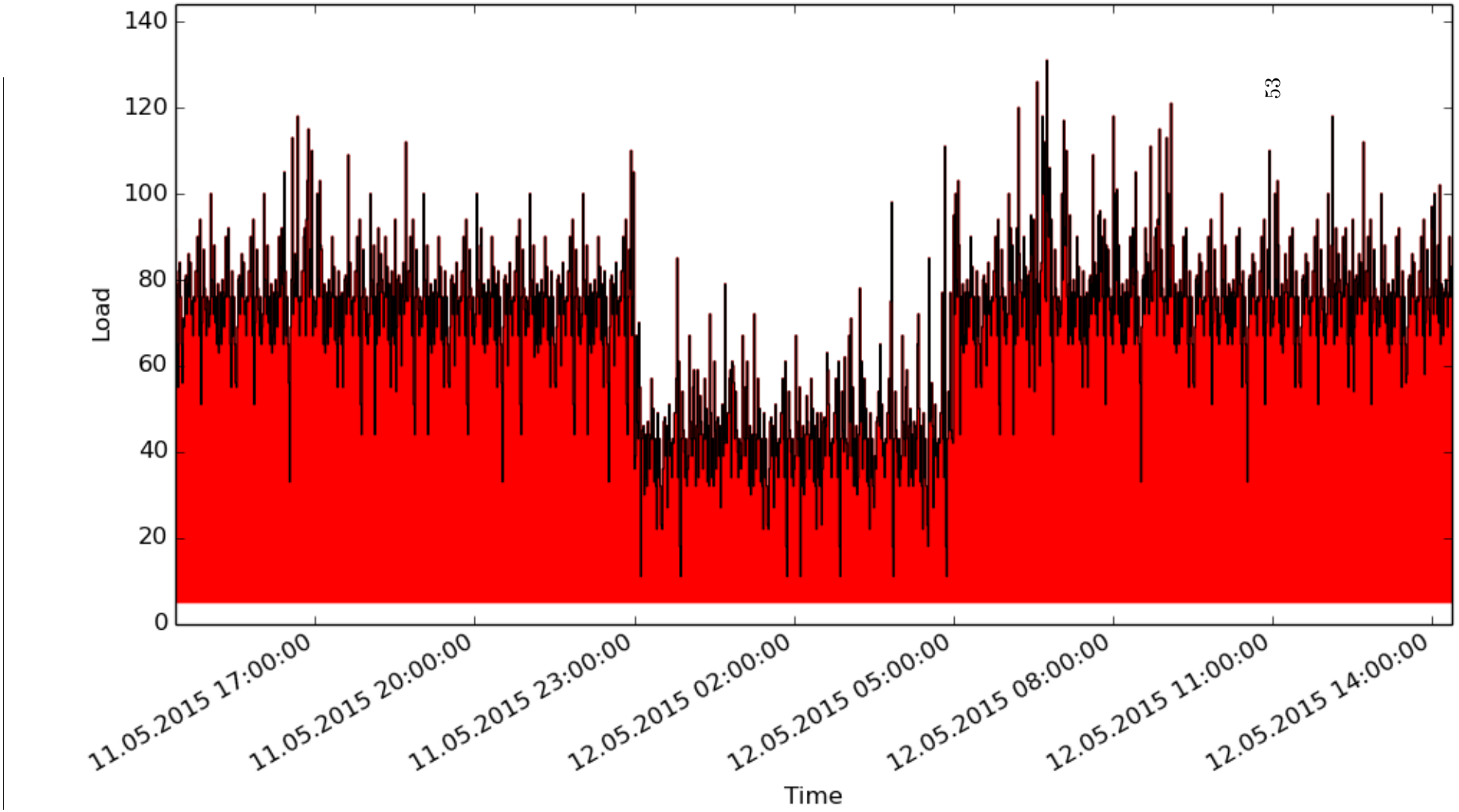
Obrázek C.12: Graf závislosti maximální zátěže na počtu generací

Příloha D

Porovnání grafů zátěže před a po optimalizaci



Obrázek D.1: Zátěžový graf před optimalizací



Obrázek D.2: Zátěžový graf po optimalizaci

Příloha E

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `cronalyzer` ..... zdrojové kódy nástroje na analýzu úloh
  - `cronwrapper` ..... zdrojové kódy nástroje na měření délky běhu úloh
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `sample-data` ..... ukázkové konfigurační soubory crontab
- `debian-package` ..... debian balíček pro snadnou instalaci nástroje