VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALYTICKÉ ZPRACOVÁNÍ METADAT K LÁMÁNÍ HESEL

METADATA ANALYSIS OF CRACKING PASSWORD TASK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ŠIMON POKORNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. VLADIMÍR VESELÝ, Ph.D.
SUPERVISOR

BRNO 2019

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

21745

Student: **Pokorný Šimon**
Program: Informační technologie
Název: **Analytické zpracování metadat k lámání hesel**
Metadata Analysis of Cracking Password Task
Kategorie: Web
Zadání:

1. Seznamte se s nástroji Hashcat a John the Ripper pro obnovu hesel. Nastudujte si architekturu systémů BOINC a FITcrack.
2. Dle doporučení vedoucího proveďte měření a sběr metadat v rámci úlohy lámání hesel pro různé typy nástrojů (např. doba inicializace GPU, velikosti pracovních balíčků v závislosti na lámaném formátu, průměrné velikosti stavových prostorů).
3. Navrhněte úpravy současného systému FITcrack a analytické pohledy na dostupná metadata, které zohlední zjištění z bodu 2.
4. Navržené řešení implementujte jako rozšíření funkcionality systému FITcrack.
5. Výsledek otestujte a měření z experimentů analyzujte. Diskutujte zjištění v kontextu dopadů na provoz celého systému FITcrack.

Literatura:

- D. P. Anderson, "BOINC: a system for public-resource computing and storage," Fifth IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4-10. doi: 10.1109/GRID.2004.14.
- HRANICKÝ Radek, HOLKOVIČ Martin, MATOUŠEK Petr a RYŠAVÝ Ondřej. On Efficiency of Distributed Password Recovery. The Journal of Digital Forensics, Security and Law. 2016, roč. 11, č. 2, s. 79-96. ISSN 1558-7215.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Veselý Vladimír, Ing., Ph.D.**
Konzultant: Hranický Radek, Ing., UIFS FIT VUT
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Tato práce se zabývá návrhem a implementací analytického webového dashboardu pro aplikaci na vzdálenou správu systému Fitcrack. Tento systém slouží pro distribuovanou obnovu hesel. Administrace tohoto systému je jednostránková aplikace, která je rozdělena na serverovou a klientskou část. Tyto části spolu navzájem komunikují. Napříč systémem se vyskytuje mnoho informací, které jsou buď zaznamenávány do databáze pro pozdější analýzu, nebo zobrazovány v reálném čase uživateli. Ideálním zobrazením dat jsou grafické prvky nejrůznějších typů (šipka, graf, tabulka, čtvereček) a barev znázorňující, co a jakým způsobem se právě děje.

Abstract

This thesis deals with the design and implementation of the analytical web dashboard for remote administration of the Fitcrack system. This system is used for distributed password recovery. Web administration is a single-page application which is divided into a server and client part. These parts communicate with each other. There is a lot of information across the system that logged into a database for later analysis or displayed in real-time for the user. The best form to show data is a graphical element of various types (arrow, graph, table, square) and different colors showing what is happening.

Klíčová slova

Hashcat, John the Ripper, BOINC, lámání hesel, distribuované výpočty

Keywords

Hashcat, John the Ripper, BOINC, password cracking, distributed computation

Citace

POKORNÝ, Šimon. *Analytické zpracování metadat k lámání hesel.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Veselý, Ph.D.

Analytické zpracování metadat k lámání hesel

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Veselého, Ph.D. Uvedl jsem všechny literární parametry a publikace, ze kterých jsem čerpal.

. . . . . .
Šimon Pokorný
14. května 2019

Poděkování

Mé největší poděkování patří mé rodině, která mě vychovala, zabezpečila, dala všechnu lásku a vždy tu pro mě byla, nehledě na to, co se v mém životě přihodilo. Poděkovat chci také mé přítelkyni, která mě maximálně podporuje v činnostech, které mě naplňují a zároveň mi dodává obrovské množství lásky, pocitu užitečnosti a klidu.

Dále bych chtěl tímto poděkovat panu Ing. Vladimírovi Veselému Ph.D. za odborné vedení mé práce, obrovskou motivaci do zpracování tohoto tématu a především stylistickou korekturu. Jako vyjádření díků, bych mu rád věnoval tento recept na vytvoření domácího cheesecake, který mě doprovází v rámci celého studia na této škole.

Co budeme potřebovat: 1,5 balení BeBe sušenek, cca 100g másla, 1x žervé (ne pažitkové), 1x lučina (taktéž ne pažitková), 1x plnotučný tvaroh ve vaničce, cukr moučka a zdobící přísadu dle chuti (ovoce, kakao, zdobení). Sušenky vložíme do pytlíku a pomocí válečku (či obdobného nástroje) rozdrtíme na jemné drobení. Vzniklou substanci přesuneme do mísy, zalijeme předem rozehřátým máslíčkem a zamícháme. Při volbě nádoby, ve které cheesecake zhotovíte, berte na vědomí, že jej budete chtít také z nádoby vyjmout. Doporučuji tedy použít dortovou formu na jejíž dno dáme pečící papír - tuto radu při servírování oceníte. Dno vybrané nádoby vyplníme připravenou hmotou. Celou nádobu i s hmotou pak necháme vychladnout v lednici, aby máslíčko ztuhlo. V mezičase smícháme žervé s lučinou a tvarohem a za stálého míchání osladíme cca dvěma lžícemi cukru. Pokud je již sušenkový korpus řádně odležen, navršíme na něj umíchanou bílou náplň. Celé veledílo opět umístíme do lednice, kde jej necháme několik hodin odpočívat. Pokud chcete, můžete dort dozdobit dle vlastního uvážení.

Obsah

Kapitola 1

Úvod

Každý produkt nebo proces ovlivňuje svým chováním své prostředí. Toto prostředí obsahuje spousty dat, která je možné sledovat a na základě znalostí z těchto dat produkt či proces vylepšovat. V reálném čase však není možné sledovat všechny ovlivněné činitele a proto je potřeba je sjednocovat či agregovat. Ideálním způsobem zobrazení je nejčastěji grafická forma (graf, šipka ukazující tendenci, barva značící závažnost) či jednoduchá tabulka. Abychom mohli sledovat jak se přibližujeme našemu cíli, sloučíme několik takovýchto zobrazení v jedno komplexní a vznikne tak analytický dashboard [7].

Má práce se věnuje především systému Fitcrack a jeho administrační části. Nástroj Fitcrack slouží k obnově hesel z hešů. Na základě zadání v jednoduché webové administraci se pak generují jednotlivé úkoly. Tyto úkoly jsou pak distribuovány na klientské počítače v síti. Díky tomu je mnohonásobně zkrácen čas, než za jaký by to zvládl pouze jeden počítač. Přidáváním počítačů do celého systému je pak zvyšován výkon.

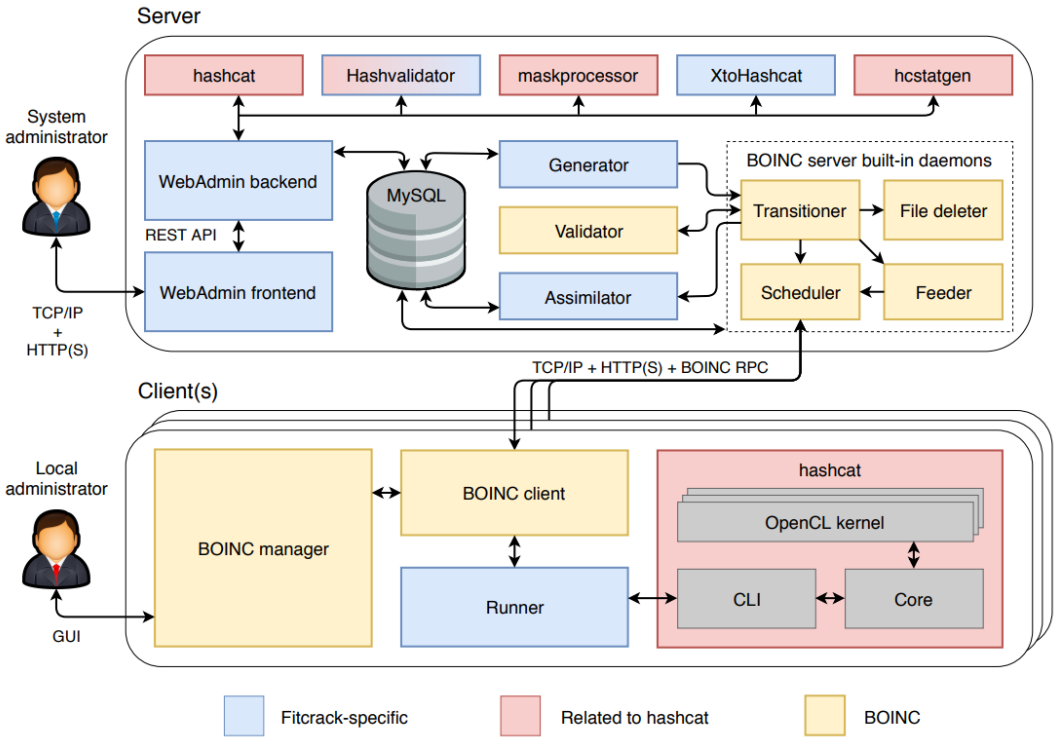
Administrační prostředí pro systém Fitcrack je jednoduché a intuitivní. Zobrazuje aktuální postup v lámání i připojené zařízení. V této práci se budu zabývat hlubší analýzou veškerých dat, která se v systému nacházejí, a různými pohledy na tato data, především pak jejich jednoduchému a inteligentnímu zobrazení. Výstupem zpracování těchto dat jsou různě komplexní grafy a tabulky, zobrazující jak aktuální data, tak i informace v rámci dlouhodobého pozorování.

Tento dokument je složen z osmi kapitol. Postupně se věnuji vysvětlení a seznámení v kapitole 2 – k čemu celý systém Fitcrack slouží, z jakých částí se skládá, a jak funguje. V kapitole 3 jsou popsány a vysvětleny technologie, se kterými jsem pracoval při tvorbě dashboardu a doplňování informací napříč celým systémem. V další kapitole 4 je návrh a vysvětlení nových informací přidaných v systému Fitcrack. Kapitola 5 se zabývá implementací veškerých navrhovaných změn společně s vysvětlením, jak a proč jsou takto dané změny zapracovány. Kapitola 6 se zabývá testováním změn, které byly v systému provedeny. Předposlední; kapitola 7, pojednává o dalších možných rozšíření, která je možné v budoucnu do systému zakomponovat. Závěrečná kapitola 8 obsahuje shrnutí a současně také uzavírá tuto práci.

Kapitola 2

Systém Fitcrack

Fitcrack je systém, který slouží především pro obnovu hesel, jejichž obrazy jsou kryptografické heše. Dále je systém schopný dekryptovat zašifrované soubory různých typů, jako je například zip, rar, pdf. Předchůdcem systému Fitcrack je aplikace Wrathion[9]. Narozdíl od této aplikace je Fitcrack rozdělen na serverovou a klientskou část, což umožňuje rozdělení úkolů na distribuovanou síť počítačů, které se můžou nacházet kdekoliv v Internetu. Každou část architektury klient-server je možné rozdělit na další funkční bloky. Každý z těchto funkčních bloků odvádí určitou část práce. Navzájem spolu komunikují vždy pomocí specifického rozhraní. Celá architektura je znázorněna na obrázku 2.1 .



Obrázek 2.1: Kompletní architektura systému Fitcrack [10]

Serverová část má především za úkol rozdělení práce a její distribuci na jednotlivé výpočetní uzly. Pro získání kryptografického heše ze zašifrovaného souboru slouží program

XtoHashcat[1]. Prostřednictvím modulu WebAdmin má správce k dispozici vzdálené ovládání celého systému Fitcrack. Pro generování nových pracovních úkolů pro klientskou část slouží modul Generator. Funkční blok Validator kontroluje sytaxi všech příchozích zpráv od uzlů. MySQL databázový server slouží pro uchování veškerých dat.

Úkolem jednotlivych uzlů je práce na výpočtu kryptografických hešů. Výsledky jsou pak zaslány na server, kde se zpracují a vyhodnotí. Stejně jako u serverové části, i klientská část je tvořena funkčními bloky, jež navzájem spolupracují. BOINC client slouží ke komunikace se serverovou částí, a zároveň je jedinou částí, která je nutná manuálně na klientovi nainstalovat. Součástí je pak BOINC Manager, jenž poskytuje grafické rozhraní pro nastavení klienta. Dalším modulem klientské části je Runner starající se o zpracování příchozích argumentů ze serverové části. Kromě toho Runner následné spuští i nástroj Hashcat, který slouží k samostatnému výpočtu kryptografických hešů.

2.1 Hashcat

Hashcat[2] je velmi výkonný program pro obnovu hesel, který využívá technologii OpenCL[3]. Tento nástroj v současné době podporuje obnovu z více než 200 typů kryptografických hešů. Rychlost podtrhuje především fakt, že tým sestavený z vývojářů Hashcatu obsadil první místo v soutěži Crack Me If you Can a to v letech 2010, 2012, 2014, 2015 a 2017[4].

2.2 BOINC

Systém Berkley Open Infrastructure for Network Computing (BOINC)[5] je volně šiřitelný framework, který se stará o distribuci dílčích úkolů na připojené klienty. Na těchto klientských stanicích pak probíhá pokus o nalezení hesla pomocí aplikace Hashcat.

Architektura BOINC je rozdělená na server a výpočetní uzly (klienty), které řeší přidělené úlohy. Navzájem mezi sebou uzly se servrem komunikují prostřednictvím zpráv kódovaných v jazyce XML. Tyto zprávy jsou pak přenášeny pomocí protokolu HTTP nebo šifrovaného HTTPS. Serverová část systému BOINC se stará o distribuci jednotlivých úloh mezi výpočetní uzly. Následně zpracovává výsledky, které dostane od uzlů zpět. Každý výpočetní uzel periodicky žádá server o přidělení úlohy ke zpracování. Jakmile klient dostane úlohu, začne jí zpracovávat a výsledek dané úlohy odešle zpět na server. Poté klient opět žádá server o novou úlohu [1].

2.3 Architektura systému Fitcrack

Zjednodušený model architektury Fitcrack je na obrázku 2.2. Model je rozdělen do několika částí, přičemž jsou z důvodu jednoduchosti vynechány některé komponenty systému BOINC.

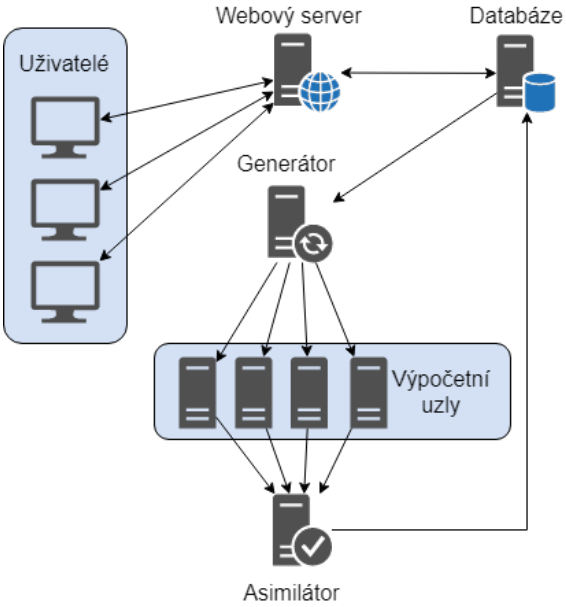
[1] http://www.fit.vutbr.cz/~izobal/prods.php.cs?id=514

[2] https://hashcat.net/hashcat/

[3] Open Computing Language - Průmyslový standard pro paralelní programování heterogenních počítačových systémů. https://digitalcommons.uri.edu/cgi/viewcontent.cgi?httpsredir=1&article=2031&context=theses

[4] https://contest-2017.korelogic.com/

[5] https://boinc.berkeley.edu/



Obrázek 2.2: Architektura systému Fitcrack [13]

2.3.1 Webový server

Na webovém serveru běží administrační část systému Fitcrack. V reálném čase může být k serveru připojeno více uživatelů najednou a mohou každý vykonávat různé akce. Po připojení uživatele na webový server se prostřednictvím prohlížeče stáhne webová aplikace, která komunikuje se serverem skrz REST API (podrobněji v kapitole 3.2). Prostřednictvím této aplikace může uživatel například monitorovat stav jednotlivých strojů, zadávat nové úlohy k řešení a především číst data z analytického dashboardu, který je cílem mé bakalářské práce.

2.3.2 Generátor

Generátor se stará o zadávání úloh výpočetním uzlům. V případě, že nastane čas, na který je naplánovaná úloha, rozdělí úlohu na menší části (pracovní jednotky), které následně předá jednotlivým uzlům. Generátor vytváří dva různé typy úkolů. Prvním typem je úkol, pomocí kterého měří výkon nově připojeného výpočetního uzlu, popřípadě uzlu, na kterém nastala chyba. Druhým typem úkolu je pak samotná část lámání heše. Generátor se tak snaží rozdělovat úkoly takovým způsobem, aby náročnost odpovídala naměřenému výkonu jednotlivých uzlů a odhadovaný čas vyřešení se tak nejvíce blížil výslednému času lámání.

2.3.3 Asimilátor

Úkolem asimilátoru je zpracování výsledků, které dostane od jednotlivých výpočetních uzlů. Na základě zpráv, které od uzlů dostane, upravuje asimilátor databázi. Pokud například přijde zpráva, že bylo nalezeno heslo, upraví databázi tak, že nastaví stav úkolu jako dokončený. Následně je pak všem uzlům, které se na této úloze podíleli, zasláno upozornění, že mohou výpočet na své straně zastavit.

2.3.4 Výpočetní uzel

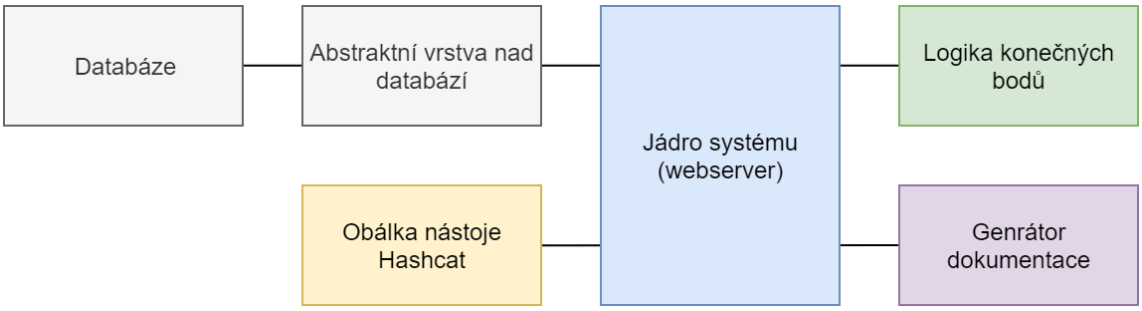
Jednotlivé výpočetní uzly jsou fyzičtí klienti připojení k systému Fitcrack. Na těchto klientech běží aplikace zvaná Runner, která zpracovává zprávy od Generátoru a řídí výpočet. Výpočet jako takový je zpracováván pomocí nástroje Hashcat 2.1.

2.4 Administrátorské rozhraní

V současné době je pro systém Fitcrack napsané webové rozhraní promocí moderních technik programování. Řešení je rozděleno na serverovou a uživatelskou část. Je tedy možné jednoduše testovat serverové rutiny pomocí automatizovaných testů. Prezentační vrstvu představuje jednostránková aplikace napsaná v jazyce JavaScript za využití frameworku Vue, viz kapitola 3.4. Díky tomu není nutné vždy zasílat celou stránku, ale pouze data, která si uživatel prostřednictvím klientské aplikace vyžádá.

2.4.1 Serverová část

Serverová část je rozdělena na několik modulů, které spolu navzájem komunikují. Diagram rozdělení modulů je znázorněn na obrázku 2.3.



Obrázek 2.3: Rozdělení serverové části na moduly[13]

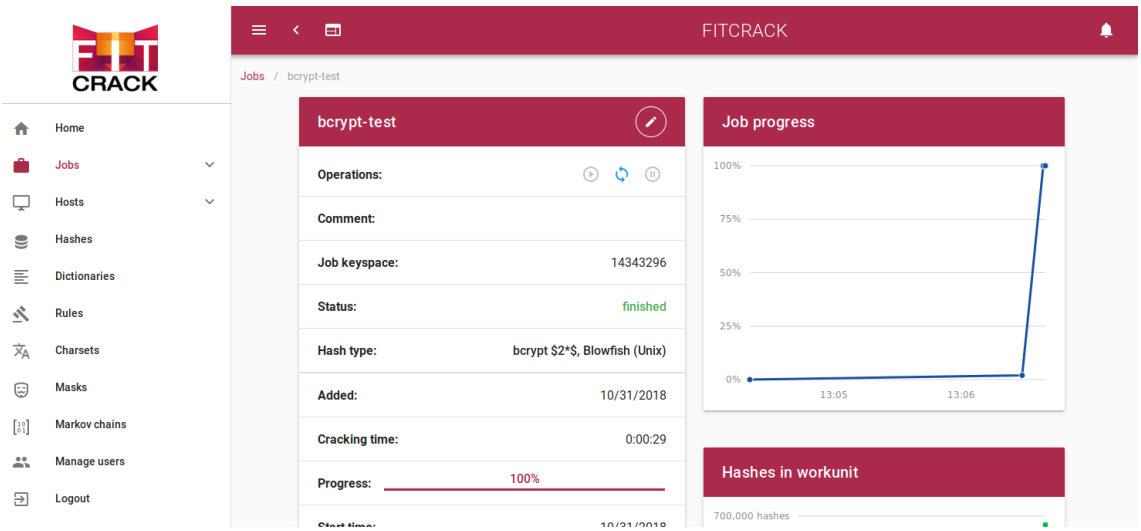
Jádro systému je základním stavebním kamenem, který plní funkci webového serveru. Zároveň v sobě má uložené veškerá nastavení jako například přístupové údaje do databáze. V případě chyby je zasílána serverem zpráva s kódem začínajícím číslicí 5 (viz 3.1.3). **Logika konečných bodů** implementuje obsluhu jednotlivých dotazů na konkrétní URI (viz 3.2.1). Dotazy nad různými objekty mohou být zaslány pomocí různých HTTP metod (viz 3.2.3). URI pro některé objetky také mohou obsahovat další parametry parametry. Chyby z tohoto modulu začínají číslem 4. **Obálka nástroje Hashcat** představuje třídu, která má za úkol získávat informace (podporované typy útoků, výpočet velikosti množiny hesel) přímo z tohoto nástroje. Modul **databáze** představuje relační databázový stroj (MySQL[6]), na který se pomocí modulu **abstraktní vrsty nad databází** jádro systému připojuje. Abstraktní vrstva mimo zjednodušení komunikace jádra systému s databází také zvyšuje ochranu před útoky typu SQL injection[7]. Po spuštění **generátoru dokumentace** jsou zpracovány všechny zdrojové kódy včetně komentářů a na jejich základě pak vygenerována interaktivní dokumentace. Ta obsahuje také příklady odpovědí na dotaz na dané konečné body.

[6] https://dev.mysql.com/
[7] https://www.w3schools.com/sql/sql_injection.asp

2.4.2 Klientská část

Klientská část je tvořena webovou aplikací napsaná pomocí frameworku Vue.js (viz 3.4). Jedná se jednostránkovou aplikaci, která pomocí HTTP dotazů komunikuje se serverovou částí. Tato komunikace probíhá pomocí asynchronních požadavků typu AJAX ve formátu JSON (viz 3.4.1). Toto využití technologií umožňuje, aby nemusela být stránka obnovována při každé akci a zároveň je tak snížena zátěž serverové části.

Aplikace je složená z postranního panelu, horní lišty a prostoru pro hlavní obsah. Postranní panel umožňuje uživateli přesouvat se v jednotlivých oblastech aplikace, jako je správa úkolů, výpočetních uzlů, slovníkových sad, uživatelů či Markovových řetězců[8]. Protože je uživatelské rozhraní responzivní[9], v případě malého rozlišení displeje skryje a umožní tak uživateli se lépe zaměřit na hlavní obsah. Horní lišta obsahuje tlačítko pro zobrazení menu a upozornění, které uživateli zpřístupní všechny notifikace. Hlavní obsah je pak genericky vytvořen pro každou stránku. Na obrázku 2.4 je zobrazen detail úkolu.



Obrázek 2.4: Zobrazení detailu úkolu

[8] http://cmp.felk.cvut.cz/~navara/stat/Markov_print.pdf
[9] http://spot.pcc.edu/~mbudiman/class-materials/usefultips/responsive-web-design.html

Kapitola 3

Použité technologie a současná administrace

V této kapitole se zabývám současným stavem administrace systému Fitcrack. Zaměřuji se na popis jednotlivých služeb, které jsou v tomto systému použity a pomocí kterých budu tento systém dále rozšiřovat.

3.1 Hypertext Transfer Protocol (HTTP)

HTTP je textový protokol určený primárně k přenosu dokumentu ve formě HTML nebo XML[1] (Extensible Markup Language). I přes to, že HTTP je textový protokol, je dnes přes něj možné díky rozšířenému MIME[2] (Multipurpose Internet Mail Extensions) poslat jakýkoli typ souboru.

3.1.1 Princip HTTP

Tento protokol funguje způsobem dotaz-odpověď. Uživatel typicky prostřednictvím internetového prohlížeče pošle serveru dotaz. Server jej zpracuje a zašle zpět klientovi odpověď s výsledkem dotazu. Výsledek obsahuje především informaci o tom, zda se podařilo dokument najít, jakého je typu. a poté i samotná data požadovaného dokumentu [4].

3.1.2 Zabezpečení HTTP

Protokol jako takový nezabezpečuje spojení. Proto bylo přidáno zabezpečení komunikace pomocí protokolu TLS[3]. Tento protokol funguje nad TCP na transportní vrstvě, což umožňuje otevřít kanál s šifrovanou komunikací. Pro kombinaci přenosu HTTP protokolu zabezpečeného pomocí TLS se vžilo pojmenování HTTPS [15].

3.1.3 Definice návratových kódů serveru

Díky stavovým kódům předává sever klientské aplikaci informaci o výsledku zpracování požadavku.

[1] https://www.w3.org/TR/xml/
[2] RFC2045 - https://tools.ietf.org/html/rfc2045
[3] Transport Layer Security (RFC 8446) - https://tools.ietf.org/html/rfc8446

O seznam stavových kódů se stará organizace IANA[4]. Každý kód je definován trojčíslím v desítkové soustavě. První cifra definuje kategorii odpovědi a zbylé dvě cifry již konkrétní typ zprávy.

Návratové kódy, které mají první cifru 1, předávají klientovi informaci, že zpráva byla přijata a pochopena. Kódy začínající cifrou 2 říkají, že byl dotaz přijat, pochopen a v pořádku vyhodnocen. Skupina začínající číslicí 3 sděluje klientovi, že je potřeba provést další akci, nejčastěji je o přesměrování. Kódy ze skupiny 4xx říkají, že došlo k chybě na straně klienta (stránka nenalezena, překročen limit žádostí). Do této skupiny patří také kód 418 I'm a teapot[5], která byla definována jako Aprílový žert. Poslední skupina začíná číslicí 5 a definuje tak kódy označující chybu na straně serveru. Nejčastěji se vyskytuje chyba indikující výpadek serveru či přetížení [5].

3.1.4 Metody HTTP požadavků

V rámci protokolu HTTP je definováno několik metod, které říkají, co se má s daným objektem provést.

- **GET** je nejčastějším typem požadavku. Celá žádost o dokument je definována v cestě URL. Tělo dotazu je pak prázdné.
- **HEAD** je velice podobný předchozí metodě. Server ale vrací pouze hlavičku odpovědi.
- **POST** podobně jako GET odesílá data na server s tím, že jsou zahrnutá v těle dotazu, nikoli v rámci URL.
- **PUT** se využívá pro nahrání souboru na server.
- **DELETE** metoda žádá o smazání objektu ze serveru definovaném v URI.
- **OPTIONS** je žádost klienta na server o seznam podporovaných metod.

3.1.5 HTTP/2

HTTP druhé generace je nástupce protokolu HTTP definovaného z roku 1991. HTTP/2 vychází z experimentálního protokolu SPDY[6] vyvíjeného společností Google. V květnu 2015 byl protokol přijat jako RFC 7540 [2]. Mezi hlavní vylepšení patří především navázání pouze jednoho TCP spojení, ve kterém je zasíláno více dat souběžně. Došlo tak k odstranění navazování nových spojení s každým novým dotazem. V rámci jednoho spojení je také možno určovat prioritu dat, které klient nutně potřebuje. V neposlední řadě lze komprimovat hlavičky dotazů.

3.2 Architektura REST

Tuto architekturu v roce 2000 navrhl a popsal ve své disertační práci Roy Fielding [6], který je zároveň spoluzakladatelem protokolu HTTP. Representational State Transfer (REST) je navržený tak, že části programu mohou fungovat na různých oddělených strojích. Uživatel pak přistupoval jednotným rozhraním k různým zdrojům. Pomocí HTTP požadavků tak

[4] Internet Assigned Numbers Authority - https://www.iana.org/
[5] https://tools.ietf.org/id/draft-nottingham-thanks-larry-00.html
[6] https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

může klient číst, upravovat, vkládat nebo mazat informace ze serveru. Přístup k jednotlivým zdrojům je pak možný prostřednictvím URI, které tak představuje jedinečný identifikátor.

3.2.1 URI

Uniform Resource Identifier (URI, nebo také jednotný identifikátor zdroje) je řetězec identifikující konkrétní zdroj informací. Dohromady je tvořeno schématem, hierarchickou částí, dodatečnými (query) parametry a fragmentem. Schéma říká, s jakým typem URI pracujeme [3]. Hierarchická část pak samotnou cestu ke zdroji. Dodatečné parametry slouží k určení konkrétních informací ze zdroje a fragment popisuje sekundární zdroj, například id elementu, kam se má prohlížeč přesunout.

3.2.2 Zásadní principy architektury REST

RESTful rozhraní je takové, které splňuje 6 základních zásad, díky kterým je toto rozhraní výkonné, škálovatelné, jednotné, upravitelné, přenositelné a spolehlivé.

Architektura client-server

Oddělením klientské aplikace od dat je jednou z nejdůležitějších zásad. Server tak nemusí generovat pro uživatele grafické prvky a může obsluhovat více klientů najednou. Díky zjednodušení komponent je snazší rozložit zátěž. Zároveň je zjednodušena přenositelnost uživatelského rozhraní na více platformách.

Bezstavová architektura

Bezstavová architektura z pohledu serveru definuje, že server nesmí ukládat žádné informace o stavu klienta. Každý požadavek z klientské strany tedy musí obsahovat veškeré potřebné informace pro jeho vyřízení. Veškeré stavy klienta si klientská aplikace musí uchovávat sama.

Uchovávání zdrojů v mezipaměti

Využívání uchování dat v mezipaměti (caching) umožňuje snížení zátěže serveru. Server tak musí označovat data, která jsou možná uložit v mezipaměti a v případě, že je lze uložit, nastavuje i dobu expirace. Klient tak nemusí vždy žádat o data nová a může je, pokud nevypršel čas expirace, načíst z mezipaměti.

Vrstevnatý systém

Odpověd na klientský dotaz nemusí vždy pocházet přímo ze serveru, na kterém běží serverová část. Odpovědět může také zprostředkovatelský server, který se využívá k zlepšení škálovatelnosti, popřípadě je tak možné uplatňovat další bezpečnostní pravidla (ochrana proti DDoS[7]).

Předání spustitelného kódu klientovi

Předání nejčastěji kompilovaného kódu či JavaScript kódu může server dočasně rozšířit nebo upravit některé klientské funkce. Jedná se o volitelné omezení.

[7] https://en.wikipedia.org/wiki/Denial-of-service_attack

Jednotné rozhraní

Jedním z nejdůležitějších omezení je právě jednotné rozhraní, které definuje především typy a formáty zpráv, které si klient a server posílají. Nejčastěji se využívá JSON, XML, HTML [12].

3.2.3 Základní metody pro přístup k datům

- **GET** při zavolaní na URI kolekce se vrátí pole s jednotlivými prvky často doplněné o doplňující informace, jako například počet prvků. V případě, že dané URI definuje jeden prvek, je vrácen konkrétní jeden záznam.
- **POST** vytvoří nový prvek kolekce, nastaví odeslané hodnoty a vrátí ID nového záznamu.
- **PUT** vymění nebo aktualizuje data konkrétního záznamu. V případě, že neexistuje, vytvoří nový.
- **DELETE** smaže záznam nebo kolekci definovanou URI.

3.3 Python

Jazyk Python[8] je skriptovací jazyk, který navržen v roce 1991 Guido van Rossum. Jedná se o iterpretovaný, víceparadigmatický jazyk. Python nabízí programování objektově orientované, strukturální a v omezené míře i funkcionální. Jazyk má dynamickou kontrolu datových typů. Dnes je vyvýjení jako open source projekt.

3.3.1 Flask

Knihovna Flask[9] v sobě implementuje malý webový framework, který obsahuje jen několik základních funkc. Další funkcionalita se doplňuje pomocí modulů, jako například SQLAlchemy pro připojení k databázovému stoji.

3.3.2 SQLAlchemy

Jedná se o open source knihovnu napsanou v jazyce Python, která implementuje objektově relační mapování (ORM) právě pro jazyk Python. SQLAlchemy[10] tak pro Flask tvoří abstraktní vrstvu pro připojení k databázi. Díky ORM může programátor přistupovat k jednotlivým tabulkám v databázi pomocí kolekcí tříd. Může tak jednoduše přidávat, upravovat, filtrovat a řadit jednotlivé zdroje. O samotnou komunikaci se pak stará tato knihovna.

3.3.3 Psutil

Rozšířená knihovna psutil[11] umožňuje sledovat mimo vytížení systémových zdrojů jako jsou procesor, paměť RAM, disk či síťová karta také například správu procesů, teplotních senzorů nebo třeba baterie. Jedná se o multiplatformní knihovnu podporující především systémy Linux, Windows, macOS či BSD systémy.

[8] https://www.python.org/
[9] http://flask.pocoo.org/
[10] https://www.sqlalchemy.org/
[11] https://pypi.org/project/psutil/

3.4 Vue.js

Vue.js[12] (zkráceně jen Vue) je framework napsaný v JavaScriptu pro vytváření uživatelských rozhraní. Nejčastěji se používá k vytváření jednostránkových webových aplikací (Single-page application). Vue narozdíl od konkurečních frameworků (React, Angular) se odlišuje především tím, že je velice malý a jednoduchý. Podobně jako Flask se Vue zpravidla rozšiřuje dalšími moduly.

Při vývoji aplikace v tomto framework se jednotlivé části aplikace dělí do menších logických celků, zvaných komponenty. Každá komponenta reprezentuje znovupoužitelnou část kódu. Jednotlivé komponenty jsou složeny z HTML šablony vyplněné JavaScript logikou a doplněné o kaskádové styly. V případě změny jednoho z JavaScriptových objektů na stránce dojde automaticky k překreslení části definované danou komponentou [14].

3.4.1 JSON

JavaScript Object Notation[13] (JSON) je způsob zápisu dat nezávislý na programovacím jazyku či platformě. Vstupem může být jakákoliv struktura dat, přičemž na výstupu bude snadno čitelný řetězec znaků. Nejčastěji se využívá pro přenos dat. Díky jeho jednoduchosti jej téměř každý programovací jazyk implementuje.

3.4.2 FlexBox

V rámci návrhu CSS3 [14] přibyla nová CSS vlastnost `Flex`. Tato funkce slouží k logickému, responzivnímu a flexibilnímu rozvržení stránky. Umožněno je tak nastavení jednotlivým elementům na stránce, aby si sami dopočítali šířku tak, aby vždy zaplnili požadované místo. Výhodou použití FlexBoxu je například možnost definování orientace řazení, popřípadě zalamování elementů (řazení elementů do řádků), specifikace poměrů šířky proti ostatním elemtům.

[12] https://vuejs.org/
[13] https://classes.engineering.wustl.edu/cse330/index.php/AJAX_and_JSON
[14] https://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

Kapitola 4

Analýza dat a navrh nových komponent

V této kapitole se věnuji analýze dat, která jsou uložena v databázi celého systému Fitcrack. Kapitola je rozdělena do několika částí. Nejdříve se zaměřím na principy a zásady tvorby dahboardů. V druhé části se zabývám návrhem zobrazení dat o jednotlivých úlohách. Třetí část je věnována informacím o výpočetních uzlech a v poslední části této kapitoly seskupuji další možná a zajímavá data k zobrazení, která ale již nejsou součástí původní databáze.

4.1 Tvorba dashboardu

Pro tvorbu dashboardu neexistuje žádná šablona, podle které by byl výsledek správný. Každý typ dashboardu je unikátní a přizpůsobený právě tomu, co uživatel potřebuje. Všechny typy mají svá společná pravidla, podle kterých se řídí, a díky nim nepůsobí na uživatele matoucím dojmem [16].

- Dashboard by se měl vejít na jednu obrazovku.
- Uživatel musí na první pohled vědět, co daná informace říká.
- Poskytovány jsou pouze relevantní informace pro naplnění cílů uživatele.
- Data jsou graficky kvalitně zpracována, aby uživatel mohl odhalit souvislosti [11].

Tvorbu jako takovou lze rozdělit do několika kroků. Na začátek si autor musí uvědomit, jaký je účel vytvářeného dashboardu. Na základě tohoto údaje si vybere datové zdroje, ze kterých chce čerpat a následně je reprezentuje vhodným typem média, která rozmístí na obrazovku.

4.1.1 Reprezentace informací

Textová reprezentace

Reprezentace dat pomocí textu se využívá nejčastěji k zobrazení pouze malého počtu hodnot. Informace obsahující pouze několik málo hodnot v grafické podobě zabírá hodně místa a v případě, že se nejedná o důležitou část dashboardu, přitahuje také moc pozornosti. Pokud má být takováto informace nějakým způsobem akcentována, vhodné použití je zvýraznění pomocí barvy či velikosti textu.

Grafická reprezentace

Oproti textové reprezentaci, která je vhodná pro méně dat, reprezentace grafická dokáže zobrazit velké množství informací bez ztráty přehlednosti. Dále je možné jednoduše znázornit souvislosti mezi různými typy ukazatelů. Nejčastějším médiem pro grafické zobrazení informací jsou grafy (sloupcový, výsečový, spojnicový), ikony, geografické mapy nebo časové osy.

4.2 Úkoly a pracovní jednotky

Zde se věnuji nejdříve databázovým tabulkám, které obsahují data týkající se úkolů zadaných uživatelem a pracovních jednotek, které jsou automaticky generovány. Následně je popsán návrh zobrazení nových komponent, které doplní současné zobrazení úvodní stránky nebo detailu úkolu či pracovní jednotky.

4.2.1 Důležité databázové tabulky

fc_job

Tabulka `en_job` je hlavní tabulkou držící základní informace o konkrétních úkolech. Najdeme zde údaje jako typ útoku, aktuální stav, čas začátku a konce výpočtu, nebo třeba nalezené heslo. Tato tabulka je základním stavebním kamenem pro výběr dat.

fc_workunit

Zde jsou k dispozici údaje o jednotlivých částech úkolu (pracovních jednotkách) definovaného v tabulce `en_job`. Nacházejí se zde pouze přidělené pracovní jednotky jednotlivým uzlům. Mezi důležité sloupce patří velikost pracovní jednotky (tj. počet hesel v pracovní jednotce), pokud je hodnota rovna nule, jedná se pouze o výkonnostní test, počet již ověřených hesel, odkaz na masku, čas přidání záznamu, host, který daný úkol řeší nebo třeba příznaky udávající, že nastala chyba a daný workunit byl vytvořen znovu.

fc_hash

V této tabulce se nacházejí informace o již nalezených heslech společně s heší. S touto tabulkou se kontrolují nové úkoly, aby nedošlo k lámání něčeho, co již bylo nalezeno. Mezi důležité sloupce patří především vstupní heš a nalezený výsledek.

4.2.2 Nové komponenty

Řešené a poslední vyřešené úkoly

Data vyjádřena tabulkou která bude umístěna na dashboardu s informací o aktuálně běžících úkolech obsahující název úkolu, čas spuštění úkolu, čas řešení úkolu, procento prozkoumaného prostoru a stav úkolu, definující, zda je úkol aktivní nebo pozastaven. V případě, že je úkol vyřešen, bude zde zobrazen výsledek. Zobrazuje se pouze aktuálně řešené úkoly doplněné o vyřešené úkoly za poslední den.

Počet pracovních jednotek

Stránka s detailem daného úkolu bude doplněna o počet vytvořených pracovních jednotek. Jednotky budou rozděleny na ty, které představují pouze měření výkonu (tedy velikost pracovní jednotky je nula) a ty, které představují reálnou práci.

Průměrná velikost pracovní jednotky

Po počtu pracovních jednotek bude přidána další statistická informace. Ta říká, jaká je průměrná velikost jedné pracovní jednotky v daném úkolu. Toto číslo se odvíjí od počtu výpočetních uzlů, které společně řeší daný úkol. Pokud se počet aktivních výpočetních jednotek mění (například se často odpojí od sítě), bude toto číslo nižší, než když jsou stále všechny aktivní [8]. Návrh zobrazení společně s předchozím bodem je na obrázku 4.1.

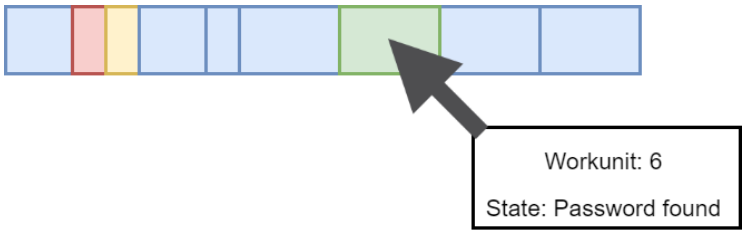
Workunits | Benchmark: 1 | Work: 10 | Avg size: 1,06M

Host	Progress	Cracking time	Generated	Start index	Keyspace	Retry	Finished	Log
DESKTOP-2D3ALBP (Radek Hranicky)	100	0:00:21	10/17/2018	0	0	No	Yes	
DESKTOP-2D3ALBP (Radek Hranicky)	100	0:00:16	10/17/2018	0	1	No	Yes	

Obrázek 4.1: Počet pracovních jednotek a průměrná velikost

Stav pracovních jednotek

Graficky bude každá pracovní jednotka znázorněna jedním čtverečkem na detailu řešení daného úkolu. Šířka čtverečku je dána poměrem velikostí pracovní jednotky, kterou představuje. Dále bude čtvereček vyplněn barvou na základě stavu představuje pro uživatele přehled, kolik bylo vytvořeno pracovních částí a v jakém stavu se nachází (přiděleno, řešeno, dokončeno úspěšně, dokončeno neúspěšně, heslo nalezeno). Podrobnější informace se zobrazí po najetí kurzorem nad daný čtvereček. Návrh je na obrázku 5.1.



Obrázek 4.2: Stav pracovních jednotek

Efektivita obnovování

V případě, že je daná úloha již dokončena, můžeme spočítat efektivitu řešení podle následujícího vzorečku [8].

$$E_{ff} = \frac{\sum_{x=1}^{N} t_x}{N * T_{fin}}$$

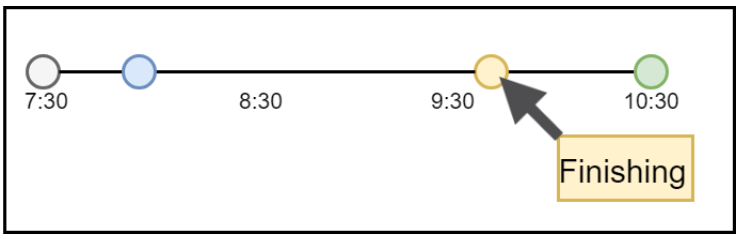
Kde je N počet uzlů, které se podílely na úkolu, t_x je čas, který byl stráven na řešení a T_{fin} je celkový čas řešení dané úlohy. Výsledná hodnota pak udává, kolik procent času bylo stráveno řešením úlohy proti celkovému času. Zbytek času pak tvoří měření výkonu či komunikace se serverem. Tato informace se bude nacházet na stránce detailu úkolu.

Zobrazení dat na časové ose

Každý detail řešené úlohy bude doplněn o časovou osu znázorňující milníky (změny stavu) v rámci řešení úkolu. Jednotlivé milníky definuje taulka 4.1. Návrh zobrazení je pak na obrázku 4.3.

Status	Name	Description
0	ready	Job is ready to be started.
1	finished	Job is finished, one or more hashes cracked.
2	exhausted	Job is finished, no password found.
3	malformed	Malformed due to incorrect input.
4	timeout	Job was stopped due due to exceeded time_end.
10	running	Computation is in progress.
11	validating	Validating hashes. (not used)
12	finishing	All keyspace assigned, some hosts still compute.

Tabulka 4.1: Stavové kódy úlohy [10]



Obrázek 4.3: Zobrazení změn stavů na časové ose

4.3 Výpočetní uzly

V této sekci se zabývám informacemi souvisejících s jednotlivými výpočetními uzly.

4.3.1 Důležité databázové tabulky

fc_host

Zde jsou uloženy informace o tom, která výpočetní jednotka pracuje na jakém úkolu. Tabulka slouží pro mapování hostů na jednotlivé úkoly. Dále je tabulka doplněna informací

o posledním naměřeném výkonu hosta, času kdy byl přidán záznam a stavové informaci - práce na výkonnostním měření, práce na přiděleném úkolu, práce na úkolu dokončena, chyba při zpracování úkolu.

fc_host_activity

Předchozí databázová tabulka udává, které výpočetní jednotky aktuálně pracují na kterém úkolu. Zde je uloženo které jednotky jsou přiřazeny ke konkrétním úkolům. Jinak řečeno, na kterých úkolech se bude jednotka podílet.

fc_host_staus

Tabulka slouží pro informaci o tom, kdy byl host naposledy připojen. V rámci administrace je tedy pak zobrazen jako online nebo offline.

4.3.2 Nové komponenty

Základní informace o uzlech

Na hlavním informačním panelu bude k dispozici informace o celkovém počtu výpočetních uzlů, kolik z nich je aktivních, kolik pracuje na nějakém úkolu a počet neaktivních uzlů.

Aktivní úkol výpočetních jednotek

V tomto případě se jedná pouze o doplnění informace do přehledu všech hostů. Pro lepší přehlednost bude v tabulce doplněn údaj o tom, na kterém úkolu právě výpočetní uzel pracuje.

4.4 Další pohledy mimo data v DB

V celém systému Fitcrack je spousta dalších infomací, které se neukládají do současné databáze. Níže je uvedeno několik dalších potenciálních komponent, které nečerpají data pouze ze stávající databáze nebo naopak potřebují vytvořit nebo upravit současné tabulky. V případě, že je to nutné, je u každé komponenty popsáno, jakým způsobem dojde k úpravě současného řešení a co informace přináší.

4.4.1 Informace o výpočetních uzlech

Aktuální vytížení uzlu

Jedná se o komponentu, která bude zobrazovat u jednotlivých uzlů informaci o vytížení jejich zdrojů (tj. procesor, paměť RAM, síťovou kartu, pevný disk a grafickou kartu). Pro tuto komponentu bude vytvořena v databázi nová tabulka, která uchovává posledních několik záznamů, aby bylo možné sledovat data v krátké historii[1]. Data bude odesílat na server sonda, která v dané periodě zjistí informace o stavu klienta. Mimo procentuální vytížení bude v tabulce také uchovávána teplota, která je zaznamenaná na procesoru a grafické kartě. Nová tabulka je znázorněna tabulkou 4.2.

[1] V případě dlouhodobého sledování zdrojů může dojít k rychlému nárůstu dat

Název	Typ	Popis
id	bigint	Primární klíč, identifikátor
boinc_host_id	int	Odkaz na hosta
time	timestamp	Čas příchodu informace
cpu	tinyint	Vytížení procesoru v procentech
ram	tinyint	Vytížení paměti RAM v procentech
net	tinyint	Vytížení síťové karty v procentech
hdd	tinyint	Vytížení pevného disku v procentech
gpu	tinint	Vytížení grafické karty v procentech
cpu_temp	decimal	Teplota na procesoru v °C
gpu_temp	decimal	Teplota na gravické kartě v °C

Tabulka 4.2: Tabulka držící informace o zdrojích

4.4.2 Informace o serveru

Mimo samotné úkoly a výpočetní jednotky je také důležité monitorovat informace o serveru, který přiděluje úkoly a uzly řídí. Z těchto informací je možné zjisti, jaká má server slabá místa, která mohou výrazně zpomalovat řešení úloh.

Sledování zdrojů serveru

Další z informací na dashboardu bude nový údaj o zdrojích serveru. K dispozici bude informace o aktuálním vytížení procesoru, síťové karty, paměti RAM nebo pevného disku. Každý rámeček bude podbarven podle nastavených limitů. Pokud se hodnota bude blížit kritické mezi, bude mít buňka barvu červenou, v opačném případě podbarvena nebude. Tyto informace sleduje serverová část administračního rozhraní za pomocí knihovny psutil 3.3.3. Návrh rozložení je na obrázku 4.4.

System usage

Part	Utilization	Temperature
CPU	30%	66°C
GPU	80%	80°C
RAM	26%	
HDD	5%	
NET	16%	

Obrázek 4.4: Návrh rozložení vytížení serveru

Záznam vysokého vytížení zdrojů serveru

Uživatel systému nemá pod kontrolou analytický dashboard neustále, a proto je důležité hlídat vysoké vytížení automaticky a v případě, že dojde k přetížení zdrojů, informace zaznamenat. Později je pak uživatel schopný dohledat na základě času, při které aktivitě byl daný zdroj vytížen. Tyto informace mohou sloužit také pro další optimalizaci serverové části

systému Fitcrack. Data jsou ukládána stejné tabulky (4.2), jako informace o výpočetních uzlech.

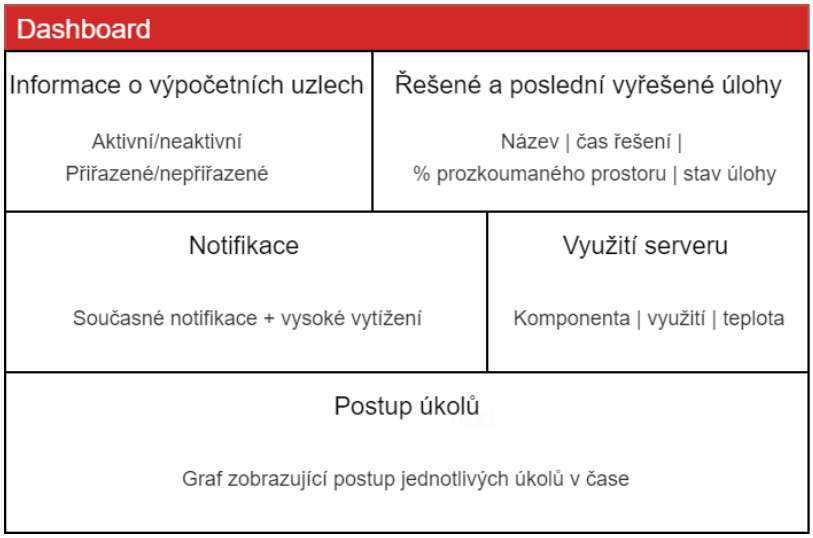
Délka obsluhy dotazu na pracovní jednotku

Společně s obecným vytížením souvisí také informace, která říká, jak dlouho trvalo odpovědět na požadavek o novou pracovní jednotku. Pracovní jednotky se přenáší pomocí *BOINC scheduling server protocol*[2], který funguje nad protokolem HTTP(S). Výsledky je tedy možné sledovat při správné konfiguraci[3] přímo z logu webového serveru. Tento údaj může opět pomoci při optimalizaci obsluhy výpočetních jednotek. Každá jednotka totiž může obsahovat desítky, až stovky tisíc hesel. Délka trvání obsluhy bude uložena v rozšířené tabulce `fc_workunit` o položku `op_time` typu `int`. Uložen zde bude čas obsluhy v sekundách.

4.5 Dashboard

Nový dashboard bude přepracován a doplněn o nové informace, které jsou popsané výše. Současné údaje budou buď upraveny nebo pouze odsunuty níže. Dashboard bude ukazovat vždy aktuální informace, nebude tedy nutné stránku aktualizovat. Návrh rozložení je na obrázku 4.5. Mezi nové komponenty patří zejména přepracované informace o uzlech, řešené a poslední vyřešené úlohy a využití serveru. Postup úkolu, již se současného řešení, bude zahrnut tak, aby byl stejně jako zbytek vždy vidět.

Ostatní komponenty, mezi které se řadí aktivita hostů, nebo manipulace se serverem, ze současného řešení budou až pod tímto panelem.



Obrázek 4.5: Návrh rozložení dashboardu

[2] https://boinc.berkeley.edu/trac/wiki/RpcProtocol
[3] http://www.ducea.com/2008/02/06/apache-logs-how-long-does-it-take-to-serve-a-request/

Kapitola 5

Implementace

Veškeré mnou dělané změny byly tvořeny nad zdrojovými kódy z oficiálního účtu GitHubu aplikce Fitcrack[1] ke dni 8. února 2019.

Administrační část systému Fitcrack je rozdělena do čtyř základních částí, které mohou běžet každá na jiném zařízení. V rámci této práce bylo zasahováno především do vrstvy klientské aplikace společně s REST API. Několik změn bylo také provedeno na úrovni databáze.

Část datová

Datová část je reprezentována relační databází MySQL[2] a uchovává veškerá data v rámci celé aplikace Fitcrack. Databáze je složena z tabulek, které využívá přímo služba BOINC. Doplněna je o tabulky sloužící pro administrační část celého systému. Tyto tabulky mají prefix `fc`.

Obsluhující část BOINC

Tato součást systému Fitcrack zprostředkovává komunikaci s výpočetními uzly, kterým přiděluje jednotlivé pracovní jednotky. Výsledky těchto pracovních jednotek pak zpracovává a na jejich základě pak upravuje jednotlivé databázové tabulky.

REST API

Aplikační rozhraní REST API napsané v jazyce Python tvoří komunikační vrstvu mezi klientskou aplikací a databází. Celé aplikační rozhraní se složeno z endpointů, které představují přístupové body pro jednotlivé typy dotazů. Každý z těchto endpointů má za úkol správu jednotlivé části celého systému.

Klientská aplikace

Uživatelské rozhraní je tvořeno jednostránkovou aplikací (SPA - single page application) napsanou v jazyce JavaScript za použití frameworku zvaného Vue.js.

V následujících sekcích je popsáno, jakým způsobem jsou navrhované změny implementovány, jak ovlivňují celý systém a jaký je výsledek.

[1] https://github.com/nesfit/fitcrack
[2] https://dev.mysql.com/doc/

5.1 Změny v úkolu

5.1.1 Souhrnné informace o pracovních jednotkách

Data v souhrnných informacích o pracovních jednotkách jsou reprezentována pomocí textu v hlavičce seznamu pracovních jednotek na detailu každého úkolu viz obrázek 5.1. Uživateli jsou zobrazeny následující údaje:

- Počet pracovních jednotek, které nejsou pro měření výkonu
- Počet pracovních jednotek, které slouží pouze pro měření výkonu
- Průměrnou velikost pracovní jednotky

Po otevření stránky s detailem jednotlivého úkolu jsou pomocí funkce `loadData` načteny veškeré informace související s daným úkolem. Jednou z těchto informací je seznam všech pracovních jednotek. Tento soubor informací je uložen v rámci aplikace a periodicky se aktualizuje. Po načtení těchto informací dochází ke zpracování, které vypočítá údaje zobrazené právě v hlavičce pracovních jednotek. Analýza je rozdělená na dvě části - balíčky pro měření výkonu a balíčky, které představují skutečnou práci na řešení úkolu.

Počet pracovních jednotek řešení úkolu

Následující ukázka slouží k zjištění počtu pracovních jednotek představující práci na úkolu.

```
1 const validWorkunits = this.data.workunits.filter(workunit => workunit.hc_keyspace !== 0);
2 this.workunitTitle.valid = validWorkunits.length;
```

Výpis 5.1: Výpočet jednotek představující skutečnou práci

Počet pracovních jednotek, které představují práci na řešení úkolu je zjištěn za použití Javascript funkce `.filter()`, která vybere pouze data na základě specifikované funkce. V tomto případě byla definována anonymní funkce s návratovou hodnotou typu `boolean`, na jejíž základě vyhodnotí, zda prvek vyhovuje podmínkám. Podmínkou zde je, že velikost pracovní jednotky není nulová, tedy že hodnota `hc_keyspace` není rovna nule. Část kódu, která se o tento výpočet stará je zobrazena ve výpisu 5.1

Počet pracovních jednotek pro měření výkonu

Po zjištění počtu pracovních jednotek, které se podílejí na plnění úkolu, se zjišťuje počet pracovních jednotek, které slouží pouze pro měření výkonu. K tomu slouží jednoduchý výpočet. Od celkového počtu pracovních jednotek je odečten počet jednotek zjištěn ve výpisu 5.1.

Průměrná velikost pracovní jednotky

Ve výpisu 5.2 je popsáno získávání průměrné velikosti pracovní jednotky.

```
1 if (validWorkunits.length > 0)
2 this.workunitTitle.avgKeyspace = validWorkunits
3     .map(workunit => workunit.hc_keyspace)
4     .reduce((a, b) => a + b) / validWorkunits.length;
```

Výpis 5.2: Výpočet jednotek představující skutečnou práci

V případě, že existuje alespoň jedna pracovní jednotka, která řeší práci na úkolu, je počítána průměrná velikost pracovních jednotek. V rámci výpočtu je využito funkce `.map()`, která z každého objektu pole `validWorkunits` vrátí velikost pracovního balíčku (`hc_keyspace`). Funkce `.reduce()` všechny vrácené hodnoty sečte a vrátí výsledný součet. Pro výpočet průměru je ještě nutné výsledek předešlé operace vydělit počtem pracovních jednotek.

Workunits | Work: 20 | Benchmark: 3 | Avg keyspace: 8,145,062.5

Host	Progress	Cracking time	Generated	Start index	Keyspace	Retry	Finished	Log
SimaCZ-Dell (Admin)	100	0:00:30	21.04.2019 13:19	0	0	No	Yes	
SimaCZ-Dell (Admin)	100	0:00:50	21.04.2019 13:20	0	8276880	No	Yes	
SimaCZ-Dell	100	0:00:50	21.04.2019	8276880	8276880	No	Yes	

Obrázek 5.1: Souhrnné informace o pracovních jednotkách

5.1.2 Efektivita výpočtu

Efektivita představuje procento času stráveného na řešení úkolu proti celkovému času řešení úlohy.

```
// Computing of efficiency
if(this.data.hosts.length === 0){
    this.efficiency = "No hosts";
}
else if (validWorkunits.length === 0) {
    this.efficiency = "No workunits yet";
} else {
    const efficiency = validWorkunits.map(workunit => workunit.cracking_time)
        .reduce((a, b) => a + b) / (this.data.hosts.length * this.data.cracking_time);
    this.efficiency = (efficiency * 100).toFixed(2) + ' %';
}
```

Výpis 5.3: Výpočet efektivity řešení

Efektivita výpočtu je doplněná informace v základním přehledu údajů o úkolu (viz obrázek 5.2). Informace o efektivitě řešení problému je zobrazena už ve chvíli, kdy je vytvořen alespoň jeden pracovní balíček. Uživatel má tak vždy informaci o tom, jak efektivně výpočet probíhá.

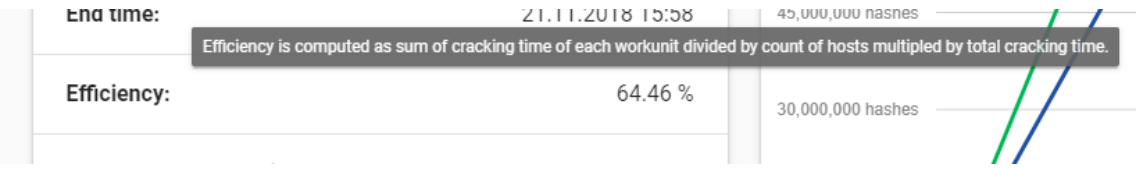
Ve výpisu 5.3 je nejdříve provedena kontrola, zda jsou k úkolu přiřazeny výpočetní uzly (`hosts`) a zároveň, zda existuje alespoň jedna pracovní jednotka, která řeší úkol. Pokud jsou přiřazeni hosté a existuje alespoň jedna validní pracovní jednotka, sečte se čas všech pracovních jednotek a vydělí se počtem přidělených uzlů vynásobeným celkovým časem řešení (`Cracking time`). Výsledná hodnota je potom vynásobena číslem 100 pro získání hodnoty v procentech.

Informace o délce řešení je uložena v sekundách. Přepočet na minuty je řešen až při zobrazování informace pomocí knihovny `moment`[3].

[3]Knihovna pro formátování času a data https://momentjs.com/

Aby uživatel nemusel hledat způsob výpočtu v dokumentech, byla k informaci o samotné efektivitě přidána také popisek, jakým způsobem je efektivita počítána. Uživatel si jej tak může zobrazit po najetí myši.

Uživateli je, mimo výslednou hodnotu efektivity, také zobrazena informace o způsobu výpočtu. Po najetí myši se uživateli zobrazí informační text, který v sobě obsahuje



Obrázek 5.2: Informace o efektivitě zobrazená v tabulce

5.1.3 Grafické zobrazení pracovních balíčků

Znázornění balíčků v grafické podobě je zobrazeno pod hlavičkou tabulky pracovních jednotek (viz obrázek 5.3). Každý rámeček představuje jeden pracovní balíček. Zobrazeny jsou pouze pracovní balíčky, které značí výpočet v rámci úkolu. Balíčky, které byly vygenerovány pouze pro účely měření výkonu zobrazeny nejsou. Rámečky jsou definovány šířkou a barvou. Šířka rámečku značí velikost pracovního balíčku vzhledem k celkové velikost všech pracovních balíčků a barva jeho stav. Přehled barev a stavů je vypsán v tabulce 5.1, také je možné zjistit stav a přesnou velikost pracovního balíčku, pokud uživatel najede myší nad daný rámeček.

```
<div class="workunit-parent">
    <div v-for="workunit in workunitsGraphical"
         v-bind:style="{
            'flex-grow': workunit.keyspace, 'background-color': workunit.color,
        }"
         class="workunit-child">
        <v-tooltip top>
            <div slot="activator"> </div>
            <span>{{workunit.text}}</span>
        </v-tooltip>
    </div>
</div>
```

Výpis 5.4: Grafické zobrazení pracovních balíčků

Ve výpisu 5.4 je popsána tvorba grafického zobrazení balíčku. Pomocí `<div v-for="..."` jsou mapována data. Tato data jsou připravena po načtení informací pracovních balíčků. K tomuto slouží funkce, která prochází jednotlivé pracovní balíčky a připravuje list objektů obsahující vždy několik vlastností - velikost, barvu a text při najetí myši.

Pomocí `v-bind:style` jsou jednotlivým objektům (rámečkům) přiřazeny styly podle informací, které přišli v připraveném objektu.

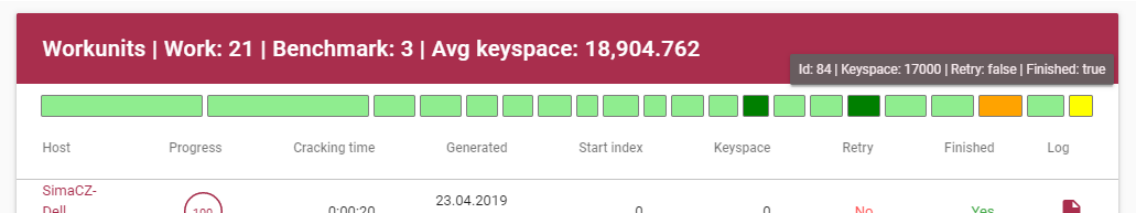
Balíček se může nacházet ve čtyřech možných stavech. Tyto stavy jsou závislé na dvou parametrech:

- zda bylo zpracování dokončeno;
- zda při zpracování došlo k chybě a zpracování balíčku musí být opakováno.

Příklad grafického zobrazení pracovních jednotek je vidět na obrázku 5.3.

CSS hodnota	barva	Popis
green		Balíček byl opakován a dokončen
ligthgreen		Balíček nebyl opeakován a byl dokončen
orange		Balíček byl opakován, ale nebyl dokončen
yellow		Balíček nebyl opakován ani dokončen

Tabulka 5.1: Definice barev pracovních balíčků



Obrázek 5.3: Grafické zobrazení pracovních balíčků

5.1.4 Historie změn stavu úkolu

Do nově vytvořené databázové tabulky `fc_job_status` jsou ukládány změny stavů. Tabulka obsahuje následující sloupce:

- jedinečný identifikátor záznamu;
- identifikátor úkolu, ke kterému změna stavu náleží;
- identifikátor typu stavu (viz tabulka 4.1);
- čas ve kterém ke změně došlo.

n

Tato tabulka je plněna daty automaticky pomocí dvou databázových spouštěčů (triggerů). Triggery jsou spuštěny po vložení nového záznamu do tabulky `fc_job` a před aktualizací záznamu v této tabulce. Implementace spouštěčů je definováva v algoritmu 5.5 a 5.6.

```
CREATE TRIGGER job_status_changes_new AFTER INSERT ON ‘fc_job‘ FOR EACH ROW
BEGIN
       INSERT INTO fc_job_status (job_id, status, time)
       VALUES (NEW.id, NEW.status, NOW());
END
```

Výpis 5.5: Přidání nového záznamu při vložení do fc_jobs

```
CREATE TRIGGER job_status_changes_edit BEFORE UPDATE ON ‘fc_job‘ FOR EACH ROW
BEGIN
    IF NEW.status <> OLD.status THEN
        INSERT INTO fc_job_status (job_id, status, time)
        VALUES (NEW.id, NEW.status, NOW());
    END IF;
END
```

Výpis 5.6: Přidání nového záznamu při změně stavu

Aby mohla být data předána z databáze do administrátorské aplikace, bylo potřeba upravit REST API. V souboru `models.py` jsem doplnil následují kód ve výspisu 5.7. Tento soubor obsahuje modely prostřednictvím kterých jsou definovány databázové tabulky pro modul `Flask-SQLAlchemy`. Každá tabulka je reprezentována jednou třídou, která musí obsahovat vlastnost `__tablename__` a vlastnosti definující typy jednotlivých sloupců. Možné je také použít dekorátor `@hybrid_property`, díky kterému můžeme vytvořit virtuální vlastnost objektu. Tato virtuální vlastnost fakticky není databázovým sloupcem, ale vytváří se na základě již definovaných sloupců tabulky. Příkladem může být volání funkce pro získání názvu stavového kódu `package_status_text_to_code_dict` na základě číselné hodnoty uložené v tabulce.

O zpracování jednotlivých požadavků se stará aplikační vrstva s frameworkem `Flask` s rozšířením `Flask-RESTPlus`[4]. Balík `endpoints` se stará o zpracování jednotlivých požadavků na konkrétní zdroje. Byl tedy vytvořen nový endpoint `status` reprezentován třemi soubory. Tento endpoint se stará o zpracování požadavků na data o změnách stavu a zaslání výsledků zpět.

- `__init__.py` - vytvoří ze složky package
- `responseModels.py` - definuje tvar objektu, který bude zaslán
- `status.py` - definuje zpracování dotazů REST API

```
class FcJobStatus(Base):
    __tablename__ = 'fc_job_status'

    id = Column(BigInteger, primary_key=True)
    job_id = Column(ForeignKey('fc_job.id'), nullable=False, index=True)
    status = Column(SmallInteger, nullable=False, server_default=text("'0'"))
    time = Column(DateTime, nullable=False, server_default=text("CURRENT_TIMESTAMP"))

    @hybrid_property
    def status_text(self):
        return package_status_text_to_code_dict.get(int(self.status))

    @hybrid_property
    def status_tooltip(self):
        return package_status_text_info_to_code_dict.get(int(self.status))

    @hybrid_property
    def status_type(self):
        if int(self.status) == 0 or int(self.status) == 10 or int(self.status) == 12:
            return 'info'
        if int(self.status) == 1:
            return 'success'
        if int(self.status) == 0 or int(self.status) == 4:
            return 'warning'
        if int(self.status) == 1 or int(self.status) == 2 or int(self.status) == 3:
            return 'error'

    job = relationship('FcJob')
```

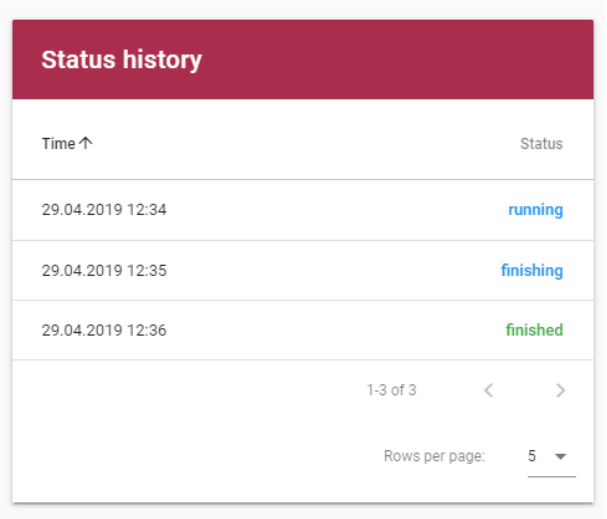
Výpis 5.7: Databázový model v REST API pro tabulku `fc_status`

[4] https://flask-restplus.readthedocs.io/en/stable/

V rámci klientské aplikace je vytvořena nová tabulka na stránce detailu úkolu. Po načtení dat z REST API jsou informace uloženy do stavu stránky. Změna stavu stránky automaticky vyvolá překreslení a vygenerování tabulky jako takové. V rámci tabulky jsou zobrazeny údaje o času, ve kterém došlo ke změně stavu a statusu, do kterého úkol přešel. Stavy, do kterých může úkol přejít se řadí do tří kategorií rozlišených třemi barvami.

- zelená - dokončeno úspěšně;
- červená - dokončeno neúspěšně;
- modrá - nedokončeno (všechny ostatní stavy).

Pro zobrazení změn stavů mělo být původně využito zobrazení na časové ose. Aby mohla být data zobrazena na časové ose se zachováním poměrů mezi událostmi, byla by časová osa prostorově neúsporná a v případě dlouhého trvání řešení úlohy (déle než hodinu) by se body na časové ose nacházely pouze na začátku a na konci u všech úkolů stejně. Stejně tak, pokud by bylo na časové ose zobrazeno více záznamů, stávala by se časová osa nepřehlednou. Pokud by měla být časová osa implementována nehledě na poměry mezi událostmi, ztrácí tato osa smysl. Proto bylo zvoleno řešení zobrazení dat, pomocí tabulky. Uživatel si tak může jednoduše data seřadit dle potřeby a stránkovat. Data budou vždy přehledná.



Obrázek 5.4: Tabulka změn stavů

5.2 Změny v rámci výpočetních uzlů

Na stránce přehledu všech výpočetních jednotek registrovaných s tímto systémem je nově údaj o počtu aktivních úkolů[5]. Abychom mohli tuto informaci zobrazit, byl upraven v rámci REST API soubor `fitcrack/responseModels.py`, ve kterém byl přidán model pro základní údaje o balíčku - identifikátor, název balíčku a stav, ve kterém se úkol nachází. Definice tohoto návratového modelu je ve výpisu 5.8. Dále byl upraven návratový model `boincHostDetail_model`, který je využívaný pro odeslání přehledu informací pro zobrazení všech výpočetních uzlů.

[5] Aktivní úkol má hodnotu stavu `10`

```
package_nano_model = api.model('Package nano', {
    'id': fields.Integer(readOnly=True, required=False, description='id ukolu'),
    'name': fields.String(required=True, description='nazev ukolu'),
    'status': fields.Integer(readOnly=True, required=False, description='stav ukolu'),
})
```

Výpis 5.8: Předloha modelu pro základní údaje o úklolu

5.3 Sekce server

Menu webové aplikace bylo doplněno o položku `Server`. V rámci této stránky jsou všechny podrobné informace o serverové části celého nástroje Fitcrack. Uživatel se zde může dozvědět, které součásti celého nástroje jsou v provozu a které jsou vyřazeny mimo provoz. Dále je zde poskytnuta uživateli informace o tom, jak moc jsou jednotlivé části serveru vytíženy. Stránka obsahuje grafy zobrazující vytížení procesorové jednotky, paměti RAM, hodnoty zápisu a čtení pevného disku, nebo aktuální vytížení síťové karty.

5.3.1 Aktuálně běžící služby

Zobrazení aktuálně spuštěných služeb již bylo vytvořeno, ale bylo pouze přesunuto na úrovni prezentační vrstvy pod položku v menu `Server`. Zobrazeno je zde uživateli, které služby jsou na serverové části spuštěny a které v provozu nejsou (viz obrázek 5.5). Požadavek na stav o službách je standardně zaslán pomocí jednostránkové aplikace na REST API. Toto API si pak stáhne stav běžících procesů v rámci nástroje BOINC ve formátu XML, které je následně dekódováno a zasláno zpět prezentační vrstvě. Ta už se pouze stará o zobrazení ve formátu *název služby - stav služby*, tedy zda je služba spuštěna, či nikoliv.

5.3.2 Grafy zobrazující vytížení počítače

Data pro tyto grafy jsou doplňována do databáze automaticky za pomocí scriptu, který provede měření a pošle jej na REST API (viz 5.5.1). Data, která grafy zobrazují, se automaticky stahují z REST API každých 15 sekund. Každý z grafů se věnuje jedné oblasti měření.

- První graf ukazuje uživateli aktuální vytížení procesorové jednotky a procentuální zaplnění paměti RAM.
- Druhý graf zobrazuje uživateli aktuální zápis a čtení pevného disku. Jednotky zobrazené uživateli jsou v kilobitech za sekundu.
- Na posledním grafu má operátor k dispozici informaci o aktuální přenosové rychlosti. Přenosová jednotka je měřena v kilobitech za sekundu.

V rámci REST API byla vytvořen nová modelová odpověď obsahující všechny potřebné údaje. Data jsou načítána pomocí endpointu `serverInfo/getUsageData`, který přijíma parametry `from_date` (od kdy mají být data zobrazena) a `to_date` (do kdy mají být data zobrazena). Hodnoty těchto parametrů může uživatel upravovat pod grafy.

Server	
Download server:	running
Upload server:	running
Scheduler:	running
feeder:	running
transitioner:	running
file_deleter:	running
work_generator (fitcrack):	running
bitwise_validator (fitcrack):	running
assimilator (fitcrack):	running
trickler:	not running

Obrázek 5.5: Aktuální spuštěné služby

5.4 Hlavní stránka webové aplikace

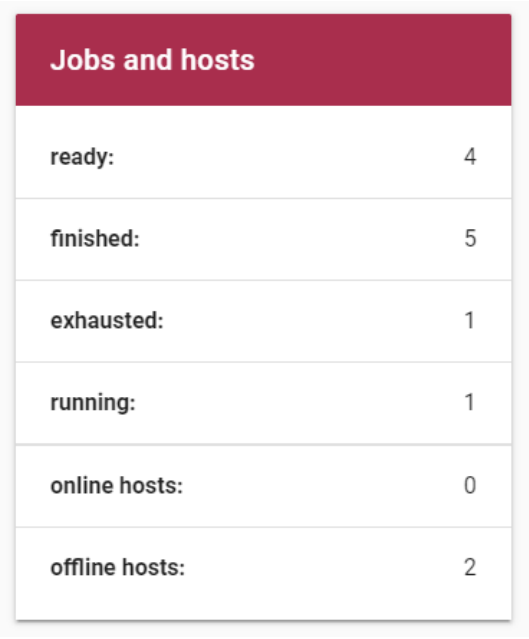
Hlavní stránka webové aplikace, webový dashboard, je upraven tak, aby zobrazoval nejdůležitější informace v rámci všech okruhů aplikace Fitcrack. V rámci celého zobrazovacího panelu má uživatel k dispozici následující informace rozdělené do několika podkapitol. Každá z následujících podkapitol představuje jednu tabulku na hlavním panelu zobrazující určitý typ informací uživateli. Data pro celý analytický dashboard jsou automaticky aktualizována v periodě 15 sekund.

5.4.1 Přehled úkolů a hostů

Tato karta má za úkol zobrazit uživateli agregovaný pohled na úkoly a výpočetní jednotky. Uživatel se zde dozví, kolik úkolů je v jakém stavu - připraveno, dokončeno úspěšně/neúspěšně, právě se počítá. Každá z těchto oblastí je doplněna o počet úkolů v daném stavu. Dále jsou v rámci karty zobrazeny ještě 2 položky, které uživateli říkají, kolik hostů je právě připojeno, tedy připraveno k provádění výpočtu a kolik je ve stavu `offline`, tedy nepřipojeno, nepřipraveno k výpočtu úkolu.

Tabulka vznikla pouze spojením již existujících tabulek (Hosts a Jobs). Ke spojení došlo z důvodu úspory místa na celém hlavním panelu.

Obrázek 5.6: Graf vytížení procesoru a zaplnění paměti RAM, graf aktuální přenosové rychlosti sítě a graf aktuální rychlosti zápisu a čtení



Obrázek 5.7: Informace o úkolech a výpočetních uzlech

5.4.2 Přehled posledních úkolů

Další karta uživateli zobrazí posledních 6 úkolů a stav, ve kterém jsou (viz obrázek 5.9). Pro výběr této sady dat bylo v rámci API enpointu přidána metoda `lastJobs`. Tato metoda

vrací úkoly, které nemají nastavený příznak `delete` a jsou seřazeny na základě poslední změny stavu.

```
1        @api.marshal_with(package_nano_list_model)
2        def get(self):
3                """
4                Vrati posledni ukoly, ve kterych byla zmena stavu
5                """
6
7                states = FcJobStatus.query.order_by(FcJobStatus.time.desc()).all()
8
9                ids = []
10               jobs_to_return = []
11
12               for jobStatus in states:
13                       if not ids.__contains__(jobStatus.job_id):
14                               ids.append(jobStatus.job_id)
15
16               for jobId in ids:
17                       job = FcJob.query.filter(FcJob.id == jobId).one()
18                       if job.deleted == 0:
19                               jobs_to_return.append(job)
20
21               return {'items': jobs_to_return}
```

Výpis 5.9: Zjištění posledních změn stavů úkolů

Definice metody je ve výpisu 5.9. Nejdříve jsou vybrána data z tabulky `fc_job_status` a seřazena od podle času od nejnovějšího. Následně je vybráno šest úkolů s poslední nejnovější změnou stavu. V případě, že některý z úkolů změní svůj stav, dostane se na začátek seznamu.

Každý řádek daného úkolu je aktivní a umožňuje uživateli se okamžitě přepnout na danou stránku zobrazující daný úkol. Tato funkcionalita je umožněna díky navigační komponentě `router-link`, která se stará o přesměrování na jiné stránky.

5.4.3 Informace o serveru

Karta zobrazující informace o serveru uživateli říká, jaké je aktuální využití serveru, tedy počítače, na kterém běží aplikace BOINC. Stejně jako u ostatních komponent vyšle SPA (single page application - jednostránková aplikace) požadavek na API, konkretně na koncový bod `/serverInfo/actualUsageData`. REST API odešle zpět aplikaci poslední záznam z tabulky `fc_server_usage`. Plnění této tabulky je popsáno v sekci 5.5.1. Uživatel se zde dozví aktuální využití procesoru, zaplnění paměti ram, dále aktuální rychlost zápisu a čtení na disk, aktuální přenosovou rychlost po síti a počet spuštěných a vypnutých služeb aplikace BOINC.

Poslední položkou výčtu je informace o spuštěných a zastavených službách systému. Dotaz je na API stejný, nicméně před uložením stavu je spočítáno, kolik služeb je aktivních a kolik nikoli.

Protože v současné době je v řešení tvorba náhradního softwaru za službu BOINC a zároveň tento framework neumožňuje získávaní aktuálního vytížení jednotlivých výpočetních jednotek, bylo implementováno pouze měření aktuálního využití serveru přidělujícího jednotlivé úkoly.

Dle původního návrhu měla tato služba měřit i aktuální teplotu procesoru a grafické karty. Vzhledem k tomu, že ne vždy jsou tyto informace dostupné, bylo bezpečnější tyto hodnoty vynechat, aby nevznikl dojem, že aplikace nefunguje, jak má.

Last active jobs		
MoreHashes	100%	finished
Nazdár	100%	exhausted
Nazdar	100%	finished
hojGen	100%	finished
hoj	100%	finished
test	100%	finished

Obrázek 5.8: Zobrazení posledních přidaných úkolů

Server	
CPU / RAM	47.8 % / 78.2 %
HDD read	767 kb/s
HDD write	11374 kb/s
Download	33 kb/s
Upload	21 kb/s
Services up/down	9 / 1

Obrázek 5.9: Tabulka s informacemi o serveru

5.4.4 Notifikace

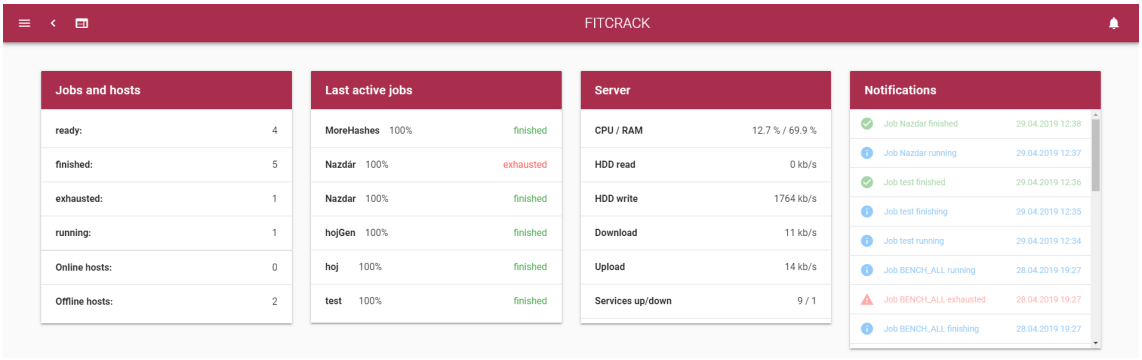
Karta zobrazující přečtené a nepřečtené notifikace je upravena pomocí kaskádových stylů tak, aby se bez problému vešla vedle dalších přehledových tabulek. Hodnota šířky (`width`) nastavena na 20% umožňuje, aby při zobrazení na standardním Full HD monitoru[6] byly

[6] Rozlišení 1080p - 1920 x 1080 px

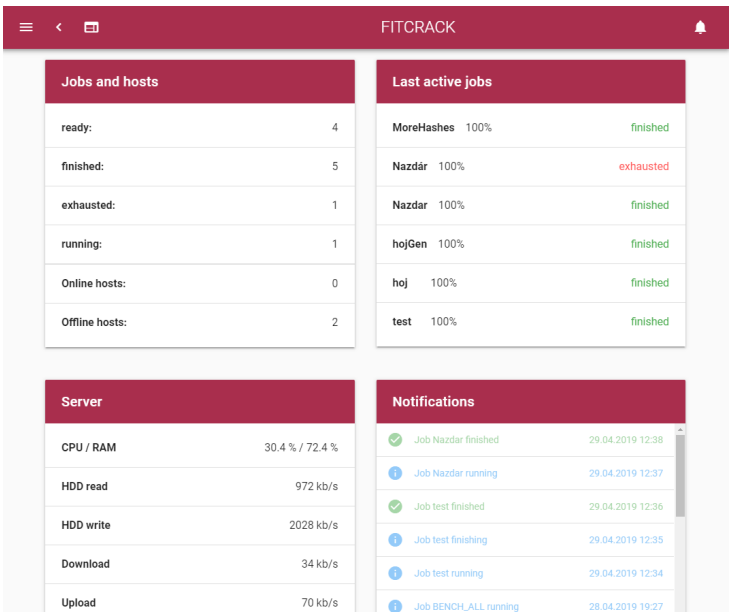
všechny čtyři hlavní informační karty zobrazeny vedle sebe (viz 5.10). Díky správnému využití `CSS Flex` (viz 3.4.2) dojde při zmenšení šířky okna k přeskládání všech karet tak, aby je uživatel viděl vždy bez deformací a na celé ploše obrazovky (viz 5.11).

5.4.5 Postup úkolů a aktivita hostů

Tyto dvě informace jsou implementovány jako grafy, mnou byla změněna pouze CSS hodnota `flex`, která je nastavena na 1 a tím je tak docíleno, že jsou grafy zobrazeny vedle sebe a jsou stejně široké. Zobrazovaná data obou grafů je možné přizpůsobit pomocí konfiguračních polí pod grafy.



Obrázek 5.10: Zobrazení první části informací na Full HD monitoru



Obrázek 5.11: Přeskládané informace na úzkém monitoru

5.5 Další úpravy mimo administrátorské rozhraní

5.5.1 Plnění tabulky `fc_server_usage`

Tvorba požadavku

Protože každá část aplikace Fitcrack může běžet na jiném stroji je nutné odeslat informace vytížení přímo ze serveru, na které běží instance BOINC. Tím pádem musí požadavky zasílat přímo počítač, na kterém běží právě část, které se měření týká. Z toho důvodu byl vytvořen samostatný script v jazyce Python. Pro funkčnost scriptu je nutné mít nainstalovanou knihovnu `psutil`, `time` a `urllib3`. Zároveň je potřeba ručně v rámci scriptu nastavit adresu REST API a přidat spuštění tohoto scritpu do některých z automatizovaných spouštěčů, například Cron[7]. Nejdříve jsou pomocí knihovny `psutil` zaznamenána data pro procentuální vytíženi procesoru a zaplnění paměti RAM. Protože knihovna neposkytuje informaci o aktuálním využití zápisu/čtení dat pevného disku a odeslání/přijetí dat po síti, je nutné nejdříve uložit informaci o počtu zapsaných/přečtených bytů, stejně tak jako odeslaných/přijatých bytů po síti. Poté je proces na 5 sekund uspán. Po probuzení procesu jsou data znovu odečtena. Od počtů nově uložených dat jsou odečteny data původní. Tímto je získán počet bytů za 5 sekund. Pro normalizovanou hodnotu je každá hodnota ještě podělena číslem 5 a převedena na bity za sekundu. Protože se standardní rychlost měří v bitech, je nutné ještě každou z hodnot vynásobit číslem 8. Takto nám vzniknou bity za sekundu. V poslední řadě jsou ještě hodnoty vyděleny číslem 1024 a převedeny tak na Kb/s. V dalším kroku jsou všechny hodnoty zaslány na REST API za pomocí knihovny `urllib3`. Všechna data jsou zasílána v rámci jednoho požadavku.

Zpracování požadavku

Pro možnost plnění informací o aktuálním vytížení serveru, na kterém běží aplikace BOINC byla v rámci endpointu `serverInfo` přidána metoda `saveData`, která ukládá přijatá data. Data se na REST API zasílají pomocí HTTP GET požadavku, kde jsou v parametrech požadavku zadány všechny hodnoty pro každý sloupec v tabulce. Definice sloupců tabulky je popsána v tabulce 5.2. Ve chvíli přijetí požadavku obsahující data, jsou parametry požadavku pomocí knihovny `flask_restplus` rozděleny do jednotlivých proměnných. Na základě těchto proměnných je vytvořen a uložen nový řádek do tabulky.

Název	Typ	Popis
id	bigint(20)	Primární klíč, identifikátor
time	timestamp	Čas příchodu informace
cpu	decimal(10,0)	Vytížení procesoru v procentech
ram	decimal(10,0)	Vytížení paměti RAM v procentech
net_recv	int(11)	Aktuální rychlost přijímaných dat po síti
net_sent	int(11)	Aktuální rychlost odesílání dat po síti
hdd_read	int(11)	Aktuální rychlost čtení dat z pevného disku
hdd_write	int(11)	Aktuální rychlost zápisu dat na pevný disk

Tabulka 5.2: Tabulka držící informace využití zdrojů serveru

[7] http://man7.org/linux/man-pages/man5/crontab.5.html

5.6 Úprava instalačního scriptu

V rámci instalčního scriptu, byly upraveny následující soubory a složky.

./server/sql/10_create_tables.sql

Tento script v rámci instalace zajišťuje vytvoření všech potřebných tabulek v rámci databáze potřebných pro administrační část systému fitcrack.

- Vytvoření tabulky `fc_job_status`, která v sobě uchovává změny stavů jednotlivých úkolů.
- Vytvoření tabulky `fc_server_usage`, která zajišťuje uchování historie vytížení jednotlivých komponent části hostující službu BOINC.
- Vytvoření omezení pro tabulku `fc_job_status`, které zajišťuje držení cizího klíče nad sloupcem `job_id`.

./server/sql/20_create_triggers.sql

Spuštěním tohoto scriptu je zajištěno vytvoření všech potřebných spouštěčů (triggerů) pro jednotlivé tabulky. Mnou byly přidány dva nutné spouštěče.

- Spouštěč pro vložení nového záznamu do tabulky `fc_job_status` po vytvoření nového záznamu v tabulce `fc_job`.
- Spouštěč pro vytvoření nového záznamu v tabulce `fc_job_status` před aktualizací každého úkolu.

./webadmin

Tato složka obsahuje implementaci REST API a klientské aplikace. Upraveno bylo několik desítek souborů tak, aby bylo maximálně splněno zadání této práce.

./measureUsage.py

Tento soubor slouží pro odesílání aktuálního vytížení serveru. Pro správnou funkci je nutné jej nahrát na server ručně a zavést do některé z plánovacích aplikací (viz kapitola 5.5.1)

Kapitola 6

Testování

Kromě samotné přípravy a tvorby jakéhokoliv projektu je také neodmyslitelnou součástí testování. To má za úkol zajistit nalezení problémů a také zjistit dostatečné informace o kvalitě produktu. Každé testování produktu musí začít stanovením cílů. Minimálním základem pro testování je ověření, že bylo splněno zadání práce. Samozřejmostí by také mělo být testování menších částí produktu a zjišťováním, že daná část vrací správné výsledky. V rámci této práce jsem se zaměřil především na testování všech nových komponent, které byly do systému přidány.

Testování probíhalo také nad daty z již produkčního prostředí. V tomto prostředí se nachází asi 60 úkolů, které jsou ve většině případů dokončené. Na úkolech se podílelo 16 výpočetních uzlů.

6.1 Testování REST API

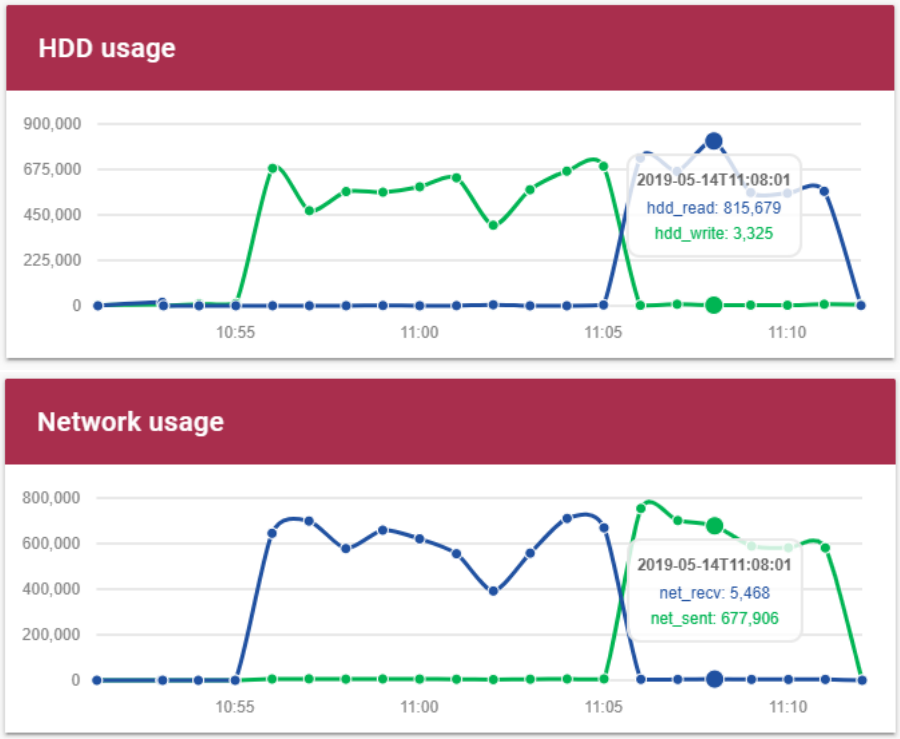
Testování REST API probíhalo již za tvorby nových komponent. Díky definovanému modelu odpovědí bylo možné správně upravit a opravit kód tak, aby byla zajištěna správná funkčnost. Každý test probíhal formou odeslání požadavku na REST API a v případě navrácení výsledku bez chyby, porovnání s očekávanými hodnotami. Kontrolováno bylo především zda proběhne zpracování bez problému, tedy návratové kódy v rámci HTTP hlavičky a syntaktickou správnost výsledku, tedy zda výsledek odpovídá správné struktuře modelu odpovědi. Všechny odpovědi jsou ve formátu JSON.

6.2 Testování měření výkonu

Mimo základní testování struktury odpovědí bylo testováno také to, zda chodí správná data v každém výsledku. Ideálním příkladem může být test na kontrolu správných hodnot v dotazu na aktuální vytížení serveru. Postupně byl zatěžován počítač na různé oblasti, které pak byly měřeny (využití procesoru, paměti RAM, přenos dat po síti a zápis/čtení pevného disku). Zde byla odhalena chyba při převodu hodnot na správnou jednotku. Všechny nalezené chyby byly následně opraveny.

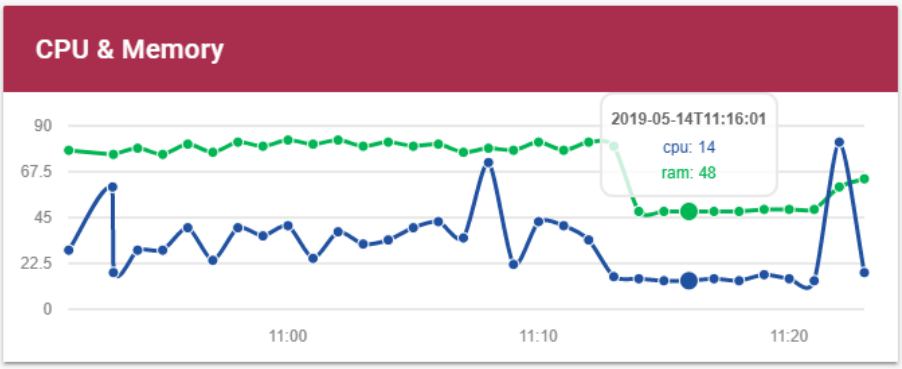
Na server byl nejdříve nahráván velký soubor, chování je vidět na obrázku 6.1. V čase 10:55 vzrostla hodnota zápisu na pevný disk a zároveň rychlost příjmu dat na druhém grafu. V čase 11:05 byl test ukončen a zároveň spuštěn test opačný - tedy data odesílána ze serveru pryč. V rámci grafů je tedy vidět, že vzrostla hodnota čtení z disku a odesílání dat.

Naměřené hodnoty odpovídali referenčním hodnotám naměřených ve správci úloh systému Windows, který data v rámci testu odesílal a pak přijímal.



Obrázek 6.1: Výsledek grafů měření využití disku a síťové karty

Měření využití procesoru a operační paměti bylo nejdříve provedeno s několika spuštěnými aplikacemi. Přibližně v čase 11:13 byly všechny mnou spuštěné aplikace ukončeny a v provozu ponecháno pouze měření zátěže, REST API a databáze MySQL. Z grafu je znatelné, že zaplnění paměti i využití procesoru po ukončení všech aplikací výrazně kleslo. Výsledek tohoto měření je na obrázku 6.2.



Obrázek 6.2: Výsledek grafů využití procesoru a paměti RAM

6.3 Testování databázových spouštěčů

Dále byla testována také správné funkčnost databázových spouštěčů. V tomto případě byl vytvořen nový úkol, se kterým se postupně manipulovalo (byl zastaven, znovu spuštěn, dokončen úspěšně i neúspěšně). Poté byl uskutečněn dotaz na všechny změny v rámci tohoto úkolu a porovnáno s předpokládanými hodnotami. Předpokládané hodnoty s hodnotami z databáze se sobě odpovídaly.

Současně s tímto testem bylo také kontrolováno, zda se na hlavním informačním panelu (dashboardu) zobrazují úkoly ve správném pořadí na základě poslední změny stavu.

6.4 Test výpočtu efektivity

Cílem testování výpočtu efektivity je ověření správnosti výpočtu a prokázání fungující závislosti pracujících hostů na celkovém čase. Mým předpokladem bylo, že počet přidělených výpočetních jednotek k úkolu zajistí vyšší efektivitu a nižší celkový čas řešení úkolu.

V rámci tohoto testu bylo analyzováno několik úkolů s rozdílnou složitostí. V rámci těchto úkolů se vyskytovaly jednoduché úlohy s velikostí prostoru pod dva miliony, 14 milionů, 114 milionů a nejsložitější úloha obsahovala 8 bilionů řetězců k ověření.

Hodnota efektivity by se měla dle očekávání měnit na základě náročnosti úkolu. V případě snadných úkolů by hodnota efektivity měla být kolem 10 - 20 procent. U náročnějších úkolů je očekávaná efektivita vyšší, než 60 procent a v případě velmi náročných úkolů zadaných k řešení kolem 95 procent.

Na obrázku 6.3 jsou vidět jednotlivé úkoly seřazené dle obtížnosti. Každá z úloh měla přiřazen počet výpočetních uzlů, které se podílely na skutečné práci. Na základě těchto výsledků jsem potvrdil, že efektivita hledání hesla stoupá s náročností úkolu.

6.4.1 Testování nad daty z produkce

U některých úkolů jsem si všiml, že efektivita celkového výpočtu je vzhledem k složitosti relativně nízká.

Na obrázku 6.4 vlevo je znázorněn relativně jednoduchý úkol s velikostí testovaného prostoru 114 milionů hesel. Jednalo se o slovníkový úkol, který se pokoušel rozšifrovat hash typu MD5. Na tomto úkolu se podílel jeden host a celkový čas lámání hesla byl 55 sekund. Zde byla vypočtena efektivita 81%.

Následně byl vybrán úkol (viz obrázek 6.4 vpravo) se složitostí zhruba dvakrát vyšší. Tento úkol se pokoušel také prolomit hash typu MD5 slovníkovým útokem. Čas hledání hesla zabral jednu minutu a 41 sekund. Vzhledem k dvojnásobné složitosti úkolu odpovídá i celkový čas řešení zhruba dvojnásobku času řešení předchozího úkolu.

Vzhledem k tomu, že se na druhém úkolu podílely dvě výpočetní jednotky, očekával bych, že se čas druhého úkolu bude přibližně rovnat času prvního úkolu.

Na obrázku 6.5 jsou znázorněny pracovní jednotky, které byly vygenerovány pro řešení úkolu vpravo na obrázku 6.4. Nejdříve byla pro každý výpočetní uzel vytvořena pracovní jednotka pro změření výkonu. Očekával bych, že po těchto dvou pracovních jednotkách se prostor k prohledání rozdělí mezi tyto dva výpočetní uzly a tím pádem bude výpočet díky paralelnímu zpracování urychlen. Tento fakt se ale nepotvrdil a celý prostor k prohledání byl přidělen pouze jedné výpočetní jednotce. Tento stav se projeví především na efektivitě, kdy je u prvního úkolu vidět, že většina výkonu uzlu je soustředěna k řešení úlohy, ale v

bcrypt-pcfg-elite

Operations:	
Comment:	
Job keyspace:	1796520
Status:	finished
Hash type:	bcrypt $2*$, Blowfish (Unix)
Added:	14.11.2018 16:04
Cracking time:	0:12:30
Progress:	100%
Start time:	14.11.2018 16:04
End time:	14.11.2018 16:10
Efficiency:	23.65 %
Seconds per workunit:	60

sha1-ahoj

Operations:	
Comment:	
Job keyspace:	14343297
Status:	exhausted
Hash type:	SHA1
Added:	14.11.2018 12:41
Cracking time:	0:00:11
Progress:	100%
Start time:	14.11.2018 12:42
End time:	14.11.2018 13:01
Efficiency:	60.97 %
Seconds per workunit:	60

hc421-test-dict

Operations:	
Comment:	fitcrack8
Job keyspace:	114746370
Status:	exhausted
Hash type:	MD5
Added:	16.11.2018 09:47
Cracking time:	0:00:55
Progress:	100%
Start time:	16.11.2018 09:47
End time:	16.11.2018 09:55
Efficiency:	81.11 %
Seconds per workunit:	60

zkouska_sha1_2workunits

Operations:	
Comment:	
Job keyspace:	8031810176000
Status:	exhausted
Hash type:	SHA1
Added:	20.11.2018 12:16
Cracking time:	1:05:51
Progress:	100%
Start time:	20.11.2018 12:17
End time:	20.11.2018 13:42
Efficiency:	98.04 %
Seconds per workunit:	60

Obrázek 6.3: Odlišně složité úkoly s odlišnou efektivitou

hc421-test-dict	
Operations:	
Comment:	fitcrack8
Job keyspace:	114746370
Status:	exhausted
Hash type:	MD5
Added:	16.11.2018 09:47
Cracking time:	0:00:55
Progress:	100%
Start time:	16.11.2018 09:47
End time:	16.11.2018 09:55
Efficiency:	81.11 %
Seconds per workunit:	60

rcokyou16 TEST LUKAS	
Operations:	
Comment:	
Job keyspace:	228152161
Status:	exhausted
Hash type:	MD5
Added:	21.11.2018 15:13
Cracking time:	0:01:41
Progress:	100%
Start time:	21.11.2018 15:13
End time:	21.11.2018 15:23
Efficiency:	29.70 %
Seconds per workunit:	60

Obrázek 6.4: Porovnání testovaných úkolů

Workunits | Work: 1 | Benchmark: 2 | Avg keyspace: 228,152,161

Host	Progress	Cracking time	Generated	Start index	Keyspace	Retry	Finished	Log
h11 (root)	100	0:00:20	21.11.2018 15:13	0	0	No	Yes	
h12 (root)	100	0:00:20	21.11.2018 15:13	0	0	No	Yes	
h12 (root)	100	0:01:00	21.11.2018 15:15	0	228152161	No	Yes	

Workunits per page 15 1-3 of 3

Obrázek 6.5: Vytvořené pracovní jednotky k řešení úkolu

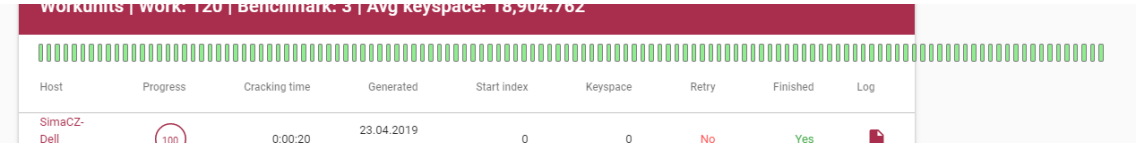
případě druhého úkolu je tento výkon dodáván pouze jedním výpočetním uzlem i přes to, že k řešení úkolu jsou přiděleny uzly dva.

Efektivita je počítána na základě celkového počtu hostů, abychom tak mohli po vyřešení úkolu určit, zda jsme přidělili adekvátní počet uzlů k úkolu s danou složitostí. Toto vede ke dvěma možnostem.

1. K úkolům s nízkou složitostí nemá smysl přidělovat více výpočetních jednotek,
2. nebo systém k přiřazování úkolů nevyhodnocuje situaci správně (nerozdělí prostor k prohledání mezi jednotlivé uzly.)

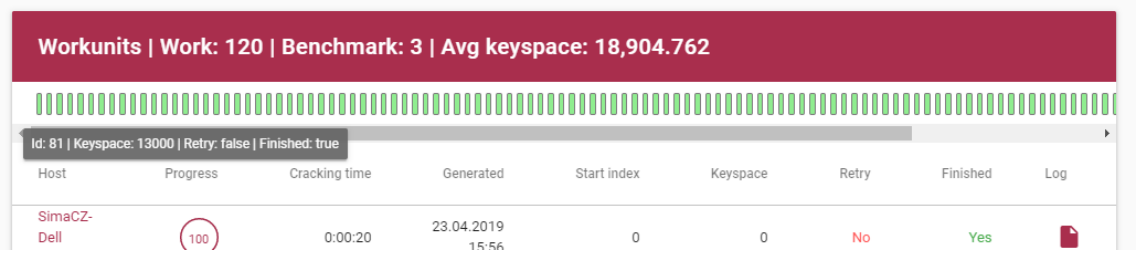
6.5 Test grafického zobrazení pracovních balíčků

Testováno bylo také vyšší množství pracovních balíčků, v rámci grafického zobrazení. Bylo odhaleno, že při tomto vyšším počtu komponenta přetéká (viz obrázek 6.6).



Obrázek 6.6: Přetečení pracovních balíčků

Vzhledem k této nalezené chybě bylo doplněno v rámci implementace řešení za pomocí posuvné lišty. Vzhledem k tomu, že jsem si při testování všiml také toho, že po najetí myši informace zakrývá také hlavičku tabulky, byl celý prostor pro grafické zobrazení jednotek zvětšen a tooltip byl přesunut pod element.



Obrázek 6.7: Opravené přetečení

6.6 Experiment s velkým množstvím pracovních jednotek

Cílem tohoto testu bylo změření odolnosti stránky s detailem úkolu v rámci uživatelské aplikace vůči zpracování velkého množství dat. Tento test byl prováděn z důvodu přidání několika výpočtů nad přijatými daty. Bylo zjišťováno, zda velký počet přijatých dat ovlivní rychlost vykreslování stránky. K tomuto experimentu jsem se rozhodl otestovat velké množství dat předaných jednostránkové aplikace před a po změnách, mnou provedených.

Nejdříve jsem si v rámci REST API vygeneroval velký objem pracovních balíčků (řádově v jednotkách stovek), který jsem následně poslal do administrátorského rozhraní. V rámci rozhraní jsem si vypsal aktuální čas včetně milisekund do konzole prohlížeče a to na začátku mnou přidané funkce a na jejím konci.

Na obrázku 6.8 jsou vidět mnou provedená dvě měření, přičemž na levém obrázku jsou zobrazeny časy bez mnou přidané funkce a vpravo po přidání kódu. V prvním případě se doba zpracování výsledku pohybuje kolem 270 milisekund (zprůměrováno). V případě druhém je čas průměrně okolo 300 milisekund. Na základě těchto výsledků zhodnocuji, že přítomnost funkce je sice znát, nicméně na délce vykreslování stránky nemá dopad.

```
Data received 11:08:54 PM 204  ms
End of function 11:08:54 PM 498  ms
Data received 11:09:01 PM 428  ms
End of function 11:09:01 PM 677  ms
```

```
Data received 11:08:00 PM 800  ms
End of function 11:08:01 PM 180  ms
Data received 11:08:08 PM 231  ms
End of function 11:08:08 PM 451  ms
```

Obrázek 6.8: Měření zpracování před mou a po mé úpravě

Testováno bylo i větší množství dat (v řádech desítek tisíců). Se zpracováním takto velkého objemu dat byl problém při odesílání z REST API do uživatelské aplikace. Na obrázku 6.9 je vidět, že odeslání požadavku proběhlo pod jednu milisekundu, nicméně doba do přijetí prvního bytu byla 41 sekund. Samotný příjem dat zabral pouze málo přes jednu sekundu.

Obrázek 6.9: Měření přenosu velkého objemu dat

Kapitola 7

Možná další rozšíření

V rámci práce jsem se setkal s několika oblastmi, kde mě zasáhla inspirace z jiných zdrojů.

7.1 Možnost personalizace

Konkrétně se jedná především o možnost úpravy hlavního zobrazovacího panelu. Podle mého názoru by bylo dobré, kdyby si mohl každý uživatel sám rozhodnout o tom, jaký druh informací je pro něj důležitý a mít možnost si takové informace zobrazovat jako první. Proto jsem přemýšlel nad možností přesouvání tabulek a grafů nejen na domovské stránce, ale také například na detailním zobrazení úkolů, výpočetních uzlů, nebo kartě serveru. Ideálním přístupem pro uživatele by bylo možnost přesouvat tabulky pomocí funkce drag and drop (přesunutí pomocí chycením a tažením myši).

Další možností personalizace by mohlo být sestavení vlastních zobrazovacích panelů na domovské obrazovce, ve kterých by si uživatel zvolil které informace přesně chce vidět, například průběh jednoho konkrétního úkolu nebo zobrazení jednoho konkrétního grafu.

7.1.1 Možnost změny velikostí

Především v rámci předělání dashboardu jsem si všiml, že několik obrazovek je relativně hodně prostorově neúsporných. Příkladem může být zobrazení detailního přehledu úkolů nebo detail zobrazení pracovní jednotky. Především na těchto obrazovkách mi přijde škoda nevyužitého prostoru po stranách tabulek. Toto rozšíření může úzce souviset s možností personalizace, kdy si uživatel zvolí, jakým způsobem chce zobrazovat informace na obrazovce. Mimo změny veškerých mezer v rámci všech obrazovek, bych také zvážil možnost změny velikosti písma. Z hlediska UX[1] může možnost této změny přispět k uživatelsky přívětivějšímu a přístupnějšímu rozhraní.

7.1.2 Tmavé téma

Trendem posledních let je také možnost zvolit si komplexní barevné téma, na základě kterého se přebarví celá aplikace. Minimálním základem by mělo být světlé (současné) a tmavé téma.

[1] User experience, česky uživatelská zkušenost

7.1.3 Možnost lokalizace

Vzhledem k tomu, že celá aplikace je vedena v celosvětově používaném jazyce, nepovažuji tento bod za nejdůležitější. Nicméně by bylo určitě dobré mít možnost přepnout jazyk v rámci celé webové aplikace.

7.2 Tvorba vlastních grafů

Další z možných změn, který by mohly být zapracovány do dalšího vývoje může být tvorba vlastního grafu. Uživatel by si tak sám mohl nadefinovat, jaké proměnné chce sledovat v závislosti na konkrétních jím určených veličinách.

7.3 Rozšíření oprávnění

Čeho jsem si také za průběhu vývoje všiml, byla možnost tvorby uživatelských účtů. V rámci dalších rozšíření by mohlo být také implementováno sofistikovanější řešení oprávnění. Jedním z nejdůležitějších oprávnění je dle mého názoru možnost pouze sledovat stav bez jakýchkoliv úprav. Rozlišované by mělo také být to, zda je přihlášený pouze uživatel nebo jako správce, tedy zda má oprávnění měnit či dokonce mazat položky v systému.

Kapitola 8

Závěr

Tato práce měla za úkol analyzovat data, která se nacházejí v systémech BOINC a Fitcrack a navrhnout analytický dashboard právě pro systém Fitcrack. Mimo samotný koncept dashboardu je v práci navrhnuto i několik doplňujících informací v různých částech do administrační části systému. V rámci návrhu bylo také přidáno několik nových funkcionalit do serverové části, jako je sledování zdrojů serveru v reálném čase, grafické zobrazení pracovních balíčků, nebo sledování zdrojů serverové části sytému Fitcrack.

Po samotném návrhu následovala implementace, která zahrnovala jak úpravu klientské aplikace, tak i změny v serverové části administračního rozhraní nebo databázi. V rámci celého systému byla implmenetována většina navrhovaných komponent. Z důvodu paralelní tvorby náhrady systému BOINC za nový bylo z implementace vynecháno několik částí, které byly založeny právě na tomto systému. V první řadě byl přepracován především celý dashboard, který ukazuje více informací na lépe využitém prostoru. Dále byla doplněna stránka pro zobrazení informací o serveru, nebo upravena stránka zobrazení detailu úkolu.

Za každým pomyslným milníkem v rámci implementace byla vždy nová funkcionalita testována, a tím ověřeno správné fungování vždy po implementaci. Díky testování bylo odhaleno několik chyb, které byly záhy opraveny.

Myslím si, že celý systém je dobře propracovaný a vývoj jde správným směrem. V samotném systému se uživatel rychle zorientuje a nemá problém s jakýmkoliv úkonem. Rád bych se na vývoji systému podílel i do budoucna. Chtěl bych se zaměřit především na proces instalace celého systému, který pro méně zkušené uživatele může být složitý.

Literatura

[1] Anderson, D. P.: BOINC: A System for Public-Resource Computing and Storage. [Online; navštíveno 20.01.2019].
URL https://boinc.berkeley.edu/grid_paper_04.pdf

[2] Belshe, M.; Peon, R.; Thomson, M.: Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor, May 2015.
URL http://www.rfc-editor.org/rfc/rfc7540.txt

[3] Berners-Lee, T.; Fielding, R. T.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. STD 66, RFC Editor, January 2005.
URL http://www.rfc-editor.org/rfc/rfc3986.txt

[4] Fielding, R.; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, RFC Editor, June 2014.
URL http://www.rfc-editor.org/rfc/rfc7230.txt

[5] Fielding, R.; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, June 2014.
URL http://www.rfc-editor.org/rfc/rfc7231.txt

[6] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Irvine doctoral dissertation, University of California, 2000, iBSN: 0-599-87118-0.

[7] Hansoti, B.: *Business Intelligence Dashboard in Decision Making*. [Online; navštíveno 30.01.2019].
URL https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1015&context=techdirproj

[8] Hranický, R.; Holkovič, M.; Matoušek, P.; aj.: On Efficiency of Distributed Password Recovery. *The Journal of Digital Forensics, Security and Law*, ročník 11, č. 2, 2016: s. 79–96, ISSN 1558-7215.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11276

[9] Hranický, R.; Matoušek, P.; Ryšavý, O.; aj.: Experimental Evaluation of Password Recovery in Encrypted Documents. In *Proceedings of ICISSP 2016*, SciTePress - Science and Technology Publications, 2016, ISBN 978-989-758-167-0, s. 299–306, doi:10.5220/0005685802990306.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11052

[10] Hranický, R.; Zobal, L.; Večeřa, V.; aj.: The architecture of Fitcrack distributed password cracking system. Technická zpráva, 2018.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11887

[11] Hynek, J.: Informační dashboardy. [Online; navštíveno 21.01.2019].
URL https://www.fit.vutbr.cz/study/courses/WAP/private/podklady/Opory/OPISDash.pdf

[12] Massé, M.: *Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2012, iSBN: 978-1-449-31050-9.

[13] Múčka, M.: *Webová aplikace pro vzdálenou správu systému Fitcrack*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=21121

[14] Passaglia, A.: *Vue.js 2 Cookbook*. Packt Publishing Ltd, 2017, iSBN: 978-1-78646-809-3.

[15] Rescorla, E.: HTTP Over TLS. RFC 2818, RFC Editor, May 2000.
URL http://www.rfc-editor.org/rfc/rfc2818.txt

[16] Richardson, J.: *Tips for Implementers: The Basics of Good Dashboard Design*. [Online; navštíveno 8.5.2019].
URL http://www.umsl.edu/~sauterv/DSS/171685.html

Příloha A

Obsah přiloženého paměťového média

Na DVD, které je přiloženo k této práci se nachází:

- tato práce ve formátu `.pdf`,
- zdrojové soubory textu této bakalářské práce v jazyce LaTeX
- složka `fitcrack` obsahující veškeré zdrojové kódy z oficiálního GitHubu aplikace Fitcrack doplněné o mé změny včetně kompilované klientské aplikace