Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Výkonová optimalizace webové aplikace v PHP

Bakalářská práce

Autor: Martin Hromádko
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové Duben 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 27.4.2015 Martin Hromádko

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Pavlu Křížovi, Ph.D. za metodické vedení práce.

Anotace

Cílem této práce je analyzovat existující webovou aplikaci a navrhnout vhodné výkonově-optimalizační úpravy, které mohou být následně implementovány. V práci jsou popsány možnosti výkonového testování aplikace a zajištění co nejreálnějších podmínek testování. Nedílnou součástí testování je vhodný měřicí software, práce obsahuje popis nejpoužívanějších nástrojů pro testování a měření zátěže aplikací. Při zkoumání webové aplikace bylo objeveno několik zásadních nedostatků, které značně ovlivnily rychlost odpovědi aplikace, jako například synchronní komunikace s externím API. Velmi zajímavých výsledků bylo dosaženo pomocí ukládání celých vygenerovaných stránek do mezipaměti. V testovacím prostředí došlo ke zrychlení odpovědi až o 3 600 %.

Annotation

Title: Performance Optimalization of PHP Web Application

The aim of this thesis is the analysis of real web application and design of suitable performance-optimizing adjustments that can be implemented. In the thesis some possibilities of performance measurement and creation of the most realistic testing conditions are described. An integral part of testing is appropriate measurement software, this thesis contains a description of the most widely used tools for measurement and testing of web application. During research a few essential mistakes, that substantially affected response speed of application like a synchronous communication with external API, were found. The most interesting results was achieved by using storing of whole generated pages to cache. The speed-up of result was about 3 600 % in test environment.

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek

1 ÚVOD

Testování výkonnosti webových aplikací se provádí vždy, když jsou potřeba přesné informace o tom, do jaké míry je aplikace připravena pružně reagovat na zátěž v ostrém provozu. Toto se provádí testováním dané webové stránky a sledováním serverové části aplikace. Pro dosažení relevantních výsledků je potřeba podmínky testování co nejvíce přiblížit podmínkám reálného provozu. Pouze relevantní výsledky mohou pomoci při identifikaci možných problémů a úzkých hrdel aplikace a rovněž mohou pomoci právě při řešení těchto problémů.

Existují různé druhy testování a u každého typu sledujeme jiné informace, proto je potřeba si před každým testováním určit, co chceme analyzovat, a podle toho použít odpovídající způsob testování. Například testovat rychlost načítání jednotlivých stránek aplikace. Stránky, které se načítají pomaleji, nám mohou pomoci odhalit úzká hrdla aplikace. Rovněž i zde záleží na kontextu aplikace a na požadavcích zákazníka. Můžeme také testovat maximální počet uživatelů, které je aplikace schopna obsloužit než „spadne".

Pro testování aplikací je potřeba použít i vhodný software. Problematice softwaru bude věnována nemalá pozornost, protože správný software dokáže ušetřit spoustu času při analýze aplikace i při implementaci následných úprav.

V následujících kapitolách bude zkoumána reálná webová aplikace, která slouží jako informační portál o politicích. Na Internetu je možné ji nalézt na adrese http://www.nasipolitici.cz/. Webovou aplikaci provozuje nezisková organizace – občanské sdružení Naši politici.

Pro tvorbu obsahu webu je využit proprietární software pro správu obsahu dodavatelské firmy. Samotný obsah jednotlivých stránek se nemění v závislosti na chování uživatele webové stránky, ale je pro každého uživatele stejný. Dynamický skriptovací jazyk PHP je zde použit pouze pro zobrazování obsahu.

Provozovatel webové stránky si stěžuje na zpomalení administrační části ve chvíli, kdy přijde na web více návštěvníků. Situace, kdy na web přichází více lidí, nastává hlavně v období před volbami a během nich. Proto bude prováděna analýza aktuálního řešení a budou hledána místa, která mohou aplikaci zpomalovat.

2 TYPY ZÁTĚŽOVÉHO TESTOVÁNÍ

Jak již bylo zmíněno výše, existuje mnoho přístupů k zátěžovému testování webových aplikací. Dále bude představeno několik z nich.

2.1 Smoke test

Název tohoto typu testu bývá nesprávně překládán jako “zahořovací test”. Smoke testy využíváme ve chvíli, kdy je aplikace dokončena, a lze ji spustit. Jedná se o krátký test sloužící jako rychlé ověření, zda je aplikace připravena na další testování. Obvykle tento test ověřuje, zda jsou všechny části aplikace implementovány, nainstalovány a spuštěny. Tyto testy se zaměřují pouze na hlavní funkce aplikace, které nejsou často upravované. Mohou být vhodně kombinovány s testy splněním. Jelikož smoke testy nebývají rozsáhlé, tak jsou často automatizovány. U velkých projektů jsou smoke testy podmínkou pro vstup aplikace do další úrovně testovaní. U menších projektů, kde bývá testování opomíjeno, nebo není příliš organizováno, se prování pouze smoke testy (1). Smoke testy tedy můžeme označit jako základní testy každé aplikace a je vhodné je provádět před všemi ostatními testy.

2.2 Výkonnostní test (performance test)

Výkonnostní testy slouží k určení nebo ověření požadovaných vlastností aplikace. Touto vlastností může být v případě webových aplikací například předpokládaná zátěž nebo maximální akceptovatelná doba odpovědi aplikace na požadavek.
Při hledání úzkých hrdel pomocí výkonnostního testu se provádí analýza na různých úrovních:

- na aplikační úrovni - hledání neefektivních konstrukcí výkonného kódu,
- na databázové úrovni - zda existuje možnost použití profilace a dotazových optimalizátorů,
- na úrovni operačního systému - sledujeme dopady na hardware serveru, na kterém aplikace běží - zatížení CPU, paměti, disku,
- na úrovni síťového provozu - analýza efektivity přenesených dat.

Při výkonnostním testování můžeme znát požadované vlastnosti, potom se jedná o tzv. "white box" testování a při testu pouze ověřujeme tyto vlastnosti V případě, že tyto vlastnosti neznáme, jedná se o "black box" testování a musíme vlastnosti softwaru určit (2).

2.3 Zátěžový test (load test)

Nejjednodušší formou výkonnostního testování je load test. Při tomto druhu testování zkoumáme chování aplikace při zatížení předpokládanou zátěží v provozu (3).
Touto zátěží může být předpokládaný počet uživatelů, kteří budou provádět určitý počet transakcí za časové období. Tímto testem získáme informace o časech odezvy aplikace ve všech důležitých transakcích. Pokud při takovém testu monitorujeme i databázi, aplikační server, atd., můžeme si ulehčit hledání úzkého hrdla aplikace.
Pro load testy je vhodné připravit opravdu velké množství rozličných dat, se kterými bude systém pracovat, abychom zajistili reálné prostředí pro testování (velké tabulky v databázi, hodně zanořených adresářů v souborovém systému, atd.).
Příklady zátěžového testování:

- objemové testování,
- testování mail serveru tisícovkami uživatelských schránek,
- testování životnosti / odolnosti,
- testování client-server aplikace tím, že necháme klienta po delší dobu komunikovat ve smyčce se serverem.

Cílem zátěžového testování je najít chyby, které nejsou odhalitelné při běžném testování, jako chyby správy paměti (memory management), úniky paměti (memory leaks), přetečení bufferů (buffer owerflows), atd., a ověřit zda aplikace splňuje naše požadavky na předpokládanou zátěž (1).

2.4 Test hraniční zátěže (stress test)

Při testování pomocí stress testu se hledá maximální možná zátěž, kterou je aplikace schopna obsloužit, než je přetížena a přestane fungovat. To může být realizováno pomocí totálního přetížení zdrojů nebo jejich odebrání (často nazývaném *negative testing*) (1). Hlavní význam tohoto testu je zjistit, jak se bude systém chovat při selhání a

následném obnovení. Cílem je zjistit, zda aplikace dokáže reagovat na danou situaci elegantně nebo se rovnou zastaví a bude nepoužitelná. Důležité je i obnovení, zde se sleduje, zda se aplikace zvládne vrátit zpět do funkčního stavu.
Příklady:

- překročení předpokládaného počtu aktivních uživatelů aplikace,
- přerušení síťové konektivity serveru, na kterém běží aplikace a následné připojení,
- vypnutí databázového serveru a jeho následný restart,
- vypnutí a následné zapnutí diskového uložiště,
- přetížení hardwarových zdrojů (CPU, paměť, disky, síť) na webovém i databázovém serveru.

Výsledkem testování hraniční zátěže je určení, jakou maximální zátěž je schopna aplikace obsloužit. Tento test slouží jako podklad pro další optimalizační úpravy aplikace.
Pokud se testuje i odebírání prostředků, tak by výsledkem měla být doporučení na úpravy aplikace, která zaručí optimální chování aplikace v případě nedostatku prostředků.

2.5 Spike test

Spike test se používá pro nárazové testování. Sleduje se chování aplikace ve výkonových špičkách, tedy ve chvílích, kdy se připojí najednou více uživatelů. Narozdíl od stress testu, nehledáme maximální možný výkon aplikace, ale pouze sledujeme, jak se aplikace bude chovat při skokovém zatížení.
Spike testy se používají při ověřování funkcionality mechanismů, které se spouštějí při určitém zatížení aplikace. Mechanismy spouštěné při zatížení aplikace se využívají hlavně při nasazení aplikace v cloudovém prostředí. Aplikace si v závislosti na předem zadaných limitech (využitá paměť, čas odpovědi atd.) může sama přidávat prostředky (další aplikační servery), které urychlí zpracování požadavků. Když zatížení klesne, aplikace sama odebere přidané prostředky a pracuje opět ve standardním režimu. Tento test se opakuje, dokud není přidávání a odebírání prostředků optimální.

2.6 Test odolnosti (soak test)

Test odolnosti se provádí pro určení, zda aplikace zvládne udržet určitou kontinuální zátěž (1).

Typické problémy identifikované tímto typem testu mohou být:

- Vážné úniky paměti, které mohou eventuálně skončit selháním.
- Výpadky spojení mezi vrstvami aplikace ve vícevrstvých aplikacích, které mohou za určitých okolností zastavit některé nebo všechny moduly systému.
- Neuzavření spojení s databázovým serverem, které může za určitých okolností zapříčinit zastavení aplikace.
- Razantní snížení reakčního času některých funkcí, jako například snížení efektivity některých vnitřních funkcí (4).

Typickým příkladem aplikace, která vyžaduje rozsáhlé testování odolnosti, je systém letové kontroly. Testy odolnosti takového systém většinou trvají od několika týdnů až po několik měsíců.

3 MODELOVÁNÍ REÁLNÝCH PODMÍNEK TESTOVÁNÍ

Pokud chceme získat přesné informace o tom, jak bude webová aplikace pracovat v reálném světě, musíme nasimulovat patřičné podmínky, ve kterých bude aplikace fungovat. To znamená namodelovat různá chování a jejich akce, které budou simulovat různé typy uživatelů používajících aplikaci. To nám dá jistotu, že aplikace bude reagovat stejně jako v reálných podmínkách. Pro správné a přesné testování bychom měli zvážit aspekty popsané níže.

3.1 Konfigurace

Před testováním je vhodné se ujistit, zda testovací prostředí je připraveno na zátěž, která bude pro testovací účely generována (3). Jedná se zejména o stejně výkonný hardware, aby nedošlo ke zkreslení výsledků pomalejším nebo rychlejším hardwarem. Důležitá je i příprava databáze, aby obsahovala data v množství odpovídajícím reálnému provozu, i to má vliv na kvalitu naměřených výsledků.

3.2 Scénáře

Scénáře jsou nejdůležitějším aspektem celého testování. Na přípravě scénářů by se měly spolupracovat všechny týmy podílejících se na vývoji aplikace, ať už to jsou obchodníci nebo produktoví manažeři. Cílem zapojení více lidí je zahrnout rozmanitost chování a různých možností používání aplikace. Pokud testujeme aplikaci, která běží v ostrém provozu, můžeme rovněž použít již zaznamenané postupy používaní, které získáme třeba na základě log souborů nebo ve zvláštních záznamech aplikace (3). Jak již bylo zmíněno výše, při výkonnostním testování se netestuje veškerá funkcionalita, tudíž není nutné, aby ve scénářích bylo zahrnuto například kliknutí na všechny možné odkazy ve webové aplikaci. Nicméně, scénáře by měly reflektovat typické užívání aplikace běžným uživatelem.

3.3 Mezičasy

Běžní uživatelé tráví určitý čas čtením, vyplňováním formulářů, sledováním videí nebo interakcí s uživatelským rozhraním před tím, než je odeslán požadavek na server. To nám vytváří určitý mezičas mezi požadavky. Tento čas má vysoký vliv na celkovou zátěž aplikace. Reakční doby jednotlivých uživatelů se liší - například někdo si čte celou

stránku, někdo jenom rychle projde text nebo známé obrazovky aplikace uživatel již zná a rychle se „prokliká“ jinam. Mezičasy při testování by měly tyto časy co nejvěrohodněji reflektovat (3). Často se používají náhodně generované mezičasy v rozsahu, který je možný strávit na dané stránce.

Nutno poznamenat, že pokud bychom do zátěžového testu neumístily mezičasy, tak by se nám z něj stal spíš test hraniční zátěže, ale získané výsledky by plně nereflektovaly realitu.

3.4 Vzory užívání webové aplikace

Je důležité si uvědomit, že většina webových aplikací má specifické časy, kdy k ní uživatelé přistupují. Některé aplikace mají globální dopad, některé bývají navštěvovány v typických pracovních hodinách. Jiné aplikace bývají navštěvovány nepřetržitě nebo třeba jenom jednou týdně i méně. Například u e-shop aplikací uživatelé využívají toho, že mají “otevřeno” nepřetržitě, ale můžeme vysledovat časy, ve kterých na tyto stránky chodí více či méně lidí. V těchto časech se vyskytují extrémní špičky potřebného výkonu aplikace. Můžeme rovněž zmínit příklad reklam na předvánoční nakupování nebo případ velké sportovní akce - tyto situace mohou rovněž vytvářet podobné špičky. V případě výukové aplikace se může největší špička pravidelně vyskytovat kolem osmé hodiny ráno, kdy přijdou školáci do školy atd. (3). Je nanejvýše vhodné, aby tyto vzory užívání i výkonové špičky byly zohledněny při testování.

3.5 Mobilní zařízení

Obsah webové stránky při prohlížení z různých zařízení, např. z chytrého mobilního telefonu, se může lišit od klasického webu. Většinou zobrazují méně obsahu a tím i klesá náročnost. Při testování je vhodné zohlednit i tento aspekt.

3.6 Rychlost připojení

Při testování aplikace je vhodné vzít v potaz průměrnou rychlost připojení uživatelů, kteří budou aplikaci používat. V dobách, kdy bylo běžné připojení 56 kbit/s, byla rychlost linky důsledkem pomalé odpovědi aplikace. Dnes je průměrná rychlost připojení v České republice 8,1 Mbit/s (výsledek měření Akamai z roku 2013), ale i přes

to je vhodné simulovat různé rychlosti připojení a zkusit, jak se budou nižší rychlosti projevovat na chování aplikace u uživatele.

3.7 Geograficky vzdálené lokace

Provedení testů s výše zmíněnými aspekty nám může přinést kvalitní výsledky. Pokud však chceme dosáhnout maximálně reálného výsledku, měli bychom provést i vzdálené testy z geograficky oddělené lokace (3). Toto testování nám umožní sledovat chování a rychlost aplikace při připojení z větší vzdálenosti (více zařízení, přes která musí data projít) při reálném síťovém provozu (i ve špičkách). Pokud testujeme aplikaci s globálním dopadem je výhodné, když je k dispozici pobočka na jiném kontinentě, odkud mohou aplikaci otestovat. To může při analýze poskytnout cenné informace o tom, jak rychle funguje aplikace při nutnosti přenosu dat na velkou vzdálenost.

4 SOFTWARE PRO ZÁTĚŽOVÉ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

Při optimalizaci a testování aplikací je potřeba zvolit vhodné nástroje, které při analýze pomohou určit úzká hrdla aplikace. V následující části jsou popsány nejčastěji používané nástroje pro výkonové testování aplikací.

4.1 Apache Benchmark

Aplikace pro příkazový řádek určená pro měření výkonu webových serverů. Je vyvíjen pod Apache Foundation a původně byl určen pro testování výkonnosti Apache HTTP Serveru (5).
Kromě nejdůležitějšího nastavení celkového počtu a množství paralelně probíhajících požadavků podporuje i další možnosti, mezi které patří:

- podpora základní autentizace na úrovni HTTP protokolu,
- vlastní nastavení parametrů cookies a hlaviček požadavku,
- specifikace zabezpečeného protokolu (SSL/TLS),
- podpora proxy serverů,
- nebo nastavení HTTP parametrů POST a PUT.

Výstup testování (Ukázka 1) nástrojem Apache Benchmark není nijak rozsáhlý – odpovídá jeho charakteru. Tento nástroj testuje pouze rychlost odpovědi serveru na požadavek. Nesimuluje chování webového prohlížeče, to znamená, že nestahuje další součásti stránky, jako jsou kaskádové styly nebo obrázky.
Z výstupu jsou nejdůležitější následující metriky:

- *Requests per second* – počet zpracovaných požadavků za 1 vteřinu.
- *Time per request* – průměrný čas, za který je zpracován 1 požadavek.
- *Percentage of the requests served within a certain time* – procento všech požadavků, které byly zpracovány do určité doby.

Metrika „*Time per request*“ se ve výstupu nachází dvakrát, protože pro její výpočet jsou použity dva rozdílné vzorce. V prvním případě se hodnota vypočítá podle vzorce

$\frac{p \cdot t \cdot 1000}{d}$, kde *p* je počet paralelních požadavků, *t* je doba, po kterou trval test, a *d* je počet provedených testů. Druhá hodnota je vypočítána podle vzorce $\frac{t \cdot 1000}{d}$, kde *t* je doba, po kterou trval test, a *d* je počet provedených testů. Jednotkou obou vypočtených hodnot jsou milisekundy.

Procento požadavků vykonaných v určitém čase obvykle má mírně stoupající charakter. K vysokým skokům odezvy dochází hlavně v případě, když je v aplikaci nějakým způsobem implementována mezipaměť. Několik prvních požadavků může mít pomalou odezvu, protože v mezipaměti nejsou připravena žádná data. Od chvíle, kdy je zpracován první požadavek, jsou další požadavky mnohem rychlejší.

```
…
Concurrency Level:      25
Time taken for tests:   25.202 seconds
Complete requests:      200
Failed requests:        199
   (Connect: 0, Receive: 0, Length: 199, Exceptions: 0)
Write errors:           0
Total transferred:      5135606 bytes
HTML transferred:       5096331 bytes
Requests per second:    7.94 [#/sec] (mean)
Time per request:       3150.232 [ms] (mean)
Time per request:       126.009 [ms] (mean, across all concurrent requests)
Transfer rate:          199.00 [Kbytes/sec] received

Percentage of the requests served within a certain time (ms)
  50%    253
  66%    354
  75%    543
  80%    646
  90%  20836
  95%  21256
  98%  25085
  99%  25184
 100%  25201 (longest request)
```

Ukázka 1 - Výstup z programu Apache Benchmark (zdroj: vlastní tvorba)

Nástroj Apache Bench je jedním ze základních nástrojů pro testování webových aplikací. Jeho výstupy slouží jako podklady pro návrh optimalizace serverové části aplikace.

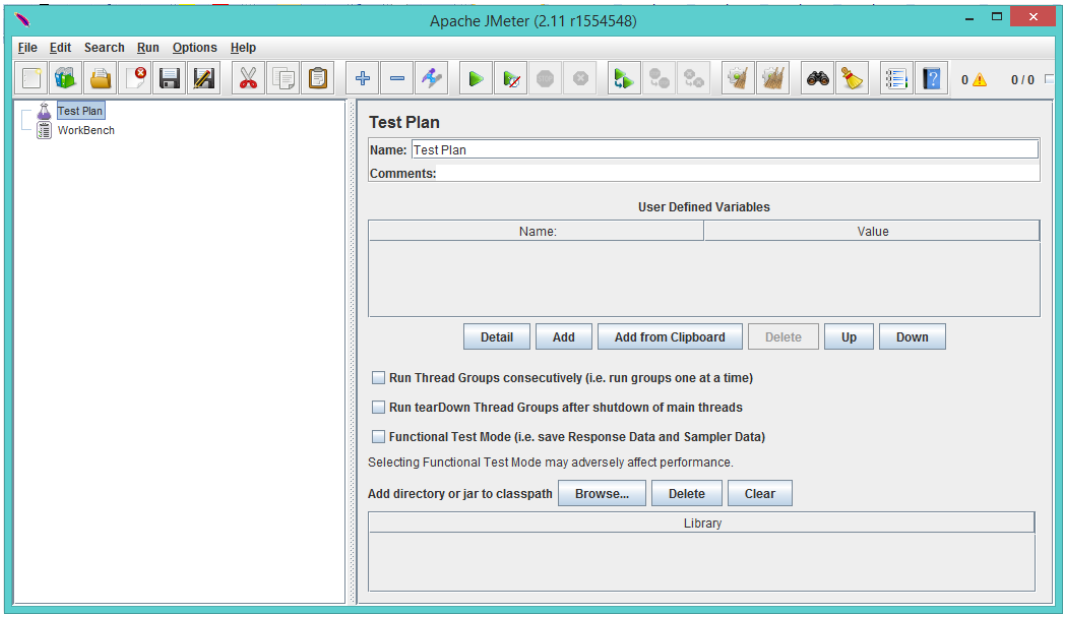
4.2 Apache JMeter

Open source aplikace napsaná v čisté Javě. Používá se pro zátěžové testy a měření výkonu aplikací. Původně určen pouze pro testování webových aplikací, ale byl rozšířen o další funkcionalitu.

JMeter může být použit pro testování jak statických, tak i dynamických zdrojů (soubory, webové aplikace v PHP, Java atd., Java objekty, databáze a dotazy, FTP (File Transfer Protokol) servery a další).
JMeter zvládne testovat různé server a protokoly:

- Web - HTTP, HTTPS,
- SOAP (Simple Object Access Protocol),
- FTP,
- Databáze pomocí JDBC driveru,
- LDAP (Lightweight Directory Access Protocol),
- Mail – SMTP (Simple Mail Transfer Protocol), POP3 (Post Office Protokol), IMAP (Internet Message Access Protocol) a jejich secure verze,
- NoSQL databáze (MongoDB),
- i nižší protokoly jako je TCP (Transmission Control Protocol).

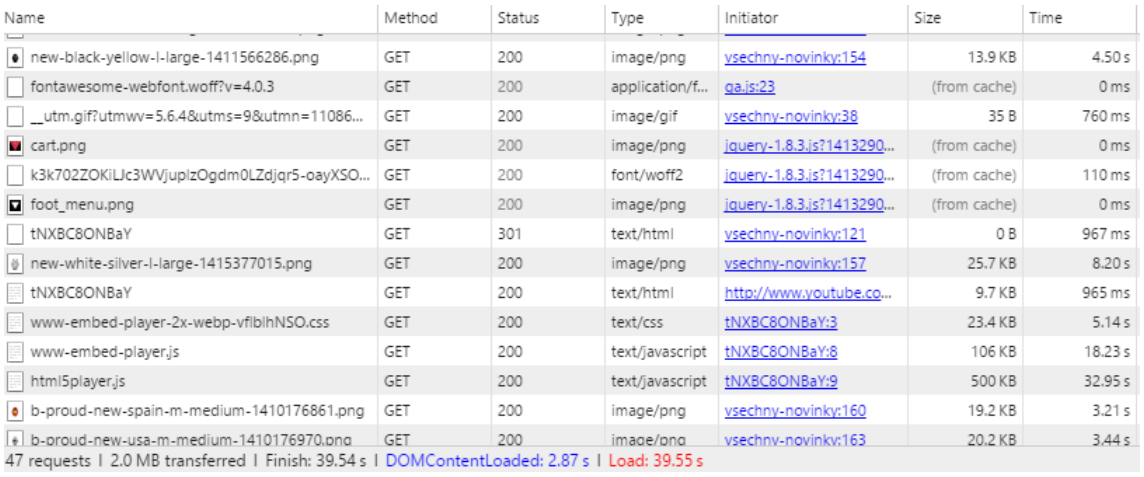
Na rozdíl od Apache Bench je JMeter multivláknový a dovoluje tak testovat
a zaznamenávat různé zdroje v jednu chvíli (6).
Důležitým faktem je, že JMeter není webový prohlížeč, a proto nevykonává žádné jiné úkony, které běžně prohlížeč vykonává - renderování stránky, vykonávání Javascriptu, atd. Tuto skutečnost je potřeba při testování JMetrem vzít v potaz, protože výsledky mohou být zkreslené tím, že webová aplikace může provádět náročné požadavky na server až Javascriptem.



Obrázek 1 - Uživatelské rozhraní aplikace JMeter (zdroj: vlastní tvorba)

4.3 Integrované testovací nástroje v prohlížečích

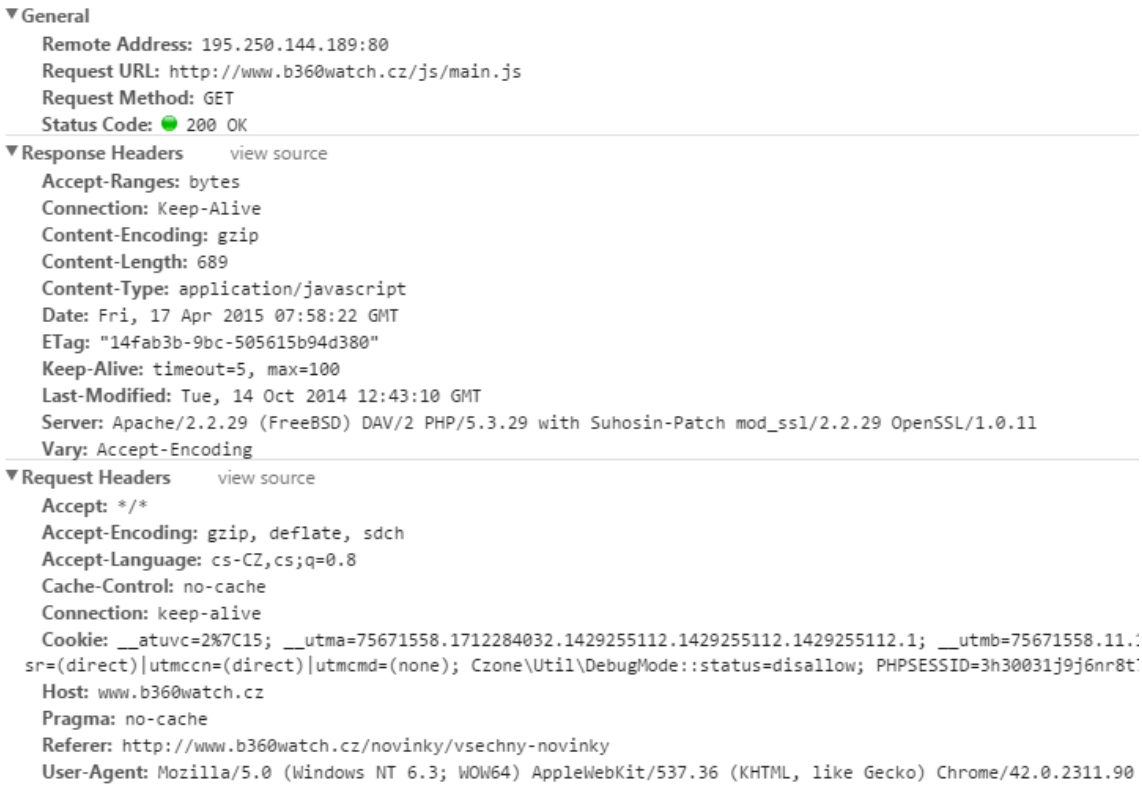
Pro testování rychlosti načítání webové stránky poskytuje většina předních internetových prohlížečů režim pro vývojáře, který obsahuje množství funkcí, a mezi nimi i měření rychlosti načítání webové stránky. Hlavní výhodou těchto nástrojů je možnost sledování dalších dotazů na server. Mezi tyto dotazy patří načítání obrázků, připojených souborů, stylů webové stránky a další dotazy generované Javascriptem (AJAX).



Obrázek 2 – Nástroje pro měření stahovaných souborů v prohlížeči Google Chrome - obecné (zdroj: vlastní tvorba)

Na obrázku (Obrázek 2) jsou zobrazeny stahované soubory a základní informace o jejich přenosu. Výstup obsahuje HTTP metodu přenosu, číselný identifikátor stavu vyřízení

požadavku, typ přeneseného souboru, jeho URI, velikost a čas, za který byl soubor přenesen ze serveru. To jsou ovšem velice strohé informace, pro detailnější analýzu je potřeba otevřít detail přenosu.



Obrázek 3 - Detailní výstup přenosu souboru z nástroje prohlížeče Google Chrome (zdroj: vlastní tvorba)

Na dalším obrázku (Obrázek 3) je zobrazen detailní popis přenosu souboru, který poskytují nástroje v prohlížeči Google Chrome. Hlavní část (*„General“*) obsahuje stejné informace jako výpis zkrácený. V další části obsahují informace o HTTP hlavičkách požadavků a jejich odpovědí.

Vývojářské nástroje prohlížeče Google Chrome obsahují i možnost simulace různých rychlostí připojení k Internetu a tím sledovat rychlost načítání webové stránky v reálném prostředí. Jedná se hlavně o simulace připojení skrze mobilní telefonní sítě.

Integrované nástroje ve webových prohlížečích jsou nejsnazší cestou, jak naměřit rychlost načítání aplikace.

4.4 PhantomJS

Obsáhlejší informace, než poskytují ladící prostřední prohlížeče, může poskytnout i program PhantomJS. Jedná se o CLI (Console Line Interface) verzi prohlížečového jádra Webkit. Nad tímto jádrem je zpřístupněno JavaScriptové API pro přístup k elementům webové stránky stejným způsobem jako v konzoli prohlížeče (7).

Výhodou tohoto programu oproti například Apache Bench je v tom, že PhantomJS stahuje všechny ostatní soubory webové stránky stejně jako webový prohlížeč. Tím pádem se simuluje kompletní požadavek tak, jako by přišel od reálného uživatele. Stahování dodatečných souborů (hlavně obrázků a videí) může mít vliv na vytížení serveru.

Pro výkonové testování je potřeba připravit vlastní knihovnu, která bude s PhantomJS spolupracovat nebo použít již nějaká hotová řešení třetí strany, protože ve výchozím stavu PhantomJS například neumí vyslat více paralelních požadavků na jednu stránku.

Možnost využít PhantomJS pro výkonové testování webových aplikací není jediným možným způsobem použití. Využívá se hlavně pro automatizované testování té části aplikace, která běží ve webovém prohlížeči. Díky výše zmiňovanému JavaScript API je možné automatizovaně testovat veškeré prvky uživatelského rozhraní, aniž by bylo potřeba klikat ve webovém prohlížeči.

Instalace PhantomJS je opravdu jednoduchá. Stačí stáhnout verzi pro správný operační systém. Každá distribuce obsahuje spustitelný soubor s výkonným kódem. Pokud je používaný systém Microsoft Windows, tak stačí stažený archív rozbalit na nějaké místo a cestu do adresáře bin přidat do proměnné PATH systému. Potom je již PhantomJS dostupný skrze příkazovou řádku pod příkazem *phantomjs*.

Pokud spustíme PhantomJS bez parametrů příkazové řádky, spustí se interaktivní konzole. Do konzole můžeme psát příkazy, které jsou ihned vykonávány.

```
1 phantomjs> console.log('Hello, world!');
2 Hello, world!
3 undefined
4 phantomjs>
```

Ukázka 2 - Funkce konzole PhantomJS (zdroj: vlastní tvorba)

V ukázce (Ukázka 2) je zobrazena práce interaktivní konzole. Na prvním řádku je zobrazen Prompt (text „phantomjs>") a příkaz, který byl zadán. V případě ukázky se jedná o vypsání textu „Hello, world!" do konzole, což je v tomto případě na standartní výstup systému. Druhý a třetí řádek jsou již výstupem vykonávaného příkazu. Na řádku 2 se nachází již zmiňovaný text a na řádku 3 návratová hodnota příkazu. Použitý příkaz *console.log* nemá žádnou návratovou hodnotu. V běhovém prostředí se taková hodnota označuje jako nedefinovaná, a proto je na řádku 3 text „undefined". Čtvrtý řádek opět zobrazuje Promt a čeká na další vstup od uživatele.

Druhá možnost, jak pracovat s programem, je napsat skript do zvláštního souboru a cestu k tomuto souboru přidat jako parametr příkazu.

Postup, jak spustit skript ve PhantomJS, je jednoduchý. Nejprve je potřeba vytvořit soubor s příkazy, které mají výt vykonány.

```
1 var page = require('webpage').create();
2 page.open('http://phantomjs.org/', function(status) {
3   console.log("Status: " + status);
4   if(status === "success") {
5    page.render('img/phantomjs.png');
6   }
7   phantom.exit();
8 });
```

Ukázka 3 - Příklad skriptu pro načtení stránky pomocí PhantomJS (zdroj: vlastní tvorba)

Ukázka (Ukázka 3) nastiňuje možnost využití PhantomJS pro uložení stránky ve formě obrázku na určené místo. Na prvním řádku je načítán modul pro práci se stránkami. Na dalším řádku se otevře spojení na danou URL („http://phantomjs.org/"), a je definována návratová funkce, která se zavolá po navázání komunikace se serverem. Funkci je předán parametr obsahující výsledek pokusu o vytvoření spojení. V návratové funkci se vypíše získaný status. Následuje podmínka ověřující, jestli načtení stránky proběhlo úspěšně. Pokud proběhlo úspěšně, tak se pomocí příkazu *page.render* uloží aktuální vzhled stránky do obrázku. Následuje ukončení zpracování pomocí *phantom.exit()*.

Pro PhantomJS existuje mnoho nástrojů a frameworků pro testování, následně si zmíníme jen dva nejvhodnější pro řešení výkonnosti webových aplikací.

4.4.1 Phantomas

Phantomas je open-source knihovna pro sběr informací o webové stránce využívající PhantomJS (8).
Slouží k měření množství stažených dat a časů, za jak dlouho byly jednotlivé části staženy.
Mezi hlavní vlastnosti patří:

- Modulární přístup – každá metrika je generována jako samostatný „modul".
- Jádro nástroje Phantomas se chová jako emitor událostí, na který může být každý modul napojen.
- Možnost exportovat výstupy ve formátech JSON a CSV, které mohou být použity pro automatizované reportovací a monitorovací nástroje.
- Jednoduchá integrace s Continuous Integration tools pomocí formátu TAP.
- Snadné použití v jiných aplikacích skrze CommonJS modul.
- Pomocí správy profilů může Phantomas emulovat různá zařízení jako mobil nebo tablet (8).

Instalace nástroje Phantomas vyžaduje nainstalované prostředí *Node.js* a jeho balíčkovací systém NPM. Instalace se provádí přímo skrze balíčkovací systém, pomocí jednoduchého příkazu *„npm install phantomas"*. Pro samotný chod je potřeba mít nainstalovaný a funkční PhantomJS.
Měření nástrojem Phantomas se spouští pomocí příkazové řádky příkazem „*phantomas*", kterému je skrze parametr předána URL stránka, která se má analyzovat.

```
phantomas v1.6.0 metrics
* requests: 74
* gzipRequests: 2
* postRequests: 0
* httpsRequests: 2
* notFound: 0
* bodySize: 1339559
* contentLength: 1339559
* httpTrafficCompleted: 14793
* timeToFirstByte: 1359
* timeToLastByte: 1529
* ajaxRequests: 1
* htmlCount: 3
* htmlSize: 43627
* cssCount: 3
* cssSize: 101682
* jsCount: 10
* jsSize: 392548
* jsonCount: 0
* jsonSize: 0
* imageCount: 57
* imageSize: 801701
…
```

Ukázka 4 - Výstup programu Phantomas (zdroj: vlastní tvorba)

Výsledek měření (Ukázka 4) je velice rozsáhlý. Výpis obsahuje pouze čistá zaměřená data bez jakékoliv nápovědy, jak výsledek interpretovat. Je tedy potřeba se zaměřit přímo na konkrétní hodnoty a pomocí nich si odvodit potřebné úpravy, které by bylo vhodné na testované aplikaci udělat.

Mezi nejzajímavější informace, které lze z výstupu vyčíst, patří následující:

- *requests* - Celkový počet požadavků, které byly odeslány na server. Pod tuto metriku spadá veškeré načítání dalších souborů (skriptů, stylů, obrázků) a AJAX požadavky.
- *timeToFirstByte* - Čas od odeslání požadavku do příjmu prvního bajtu odpovědi. Hodnota této metriky je udávána v milisekundách a určuje rychlost odpovědi serveru.
- *contentLenght* – Velikost celé stránky po kompletním načtení udávaná v bajtech.
- *httpTrafficCompleted* – Čas potřebný k přijetí posledního bajtu posledního požadavku. Určuje potřebnou dobu k úplnému načtení webové stránky.
- *cssCount, cssSize* – Počet a velikost stahovaných souborů se styly.

- *jsCount, jsSize* – Počet a velikost stahovaných souborů s Javascriptem. Společně s počty a velikostí souborů se styly ukazuje tato metrika, kde je možné sjednocovat a zmenšovat dané soubory.
- *imageCount, imageSize* – Počet a velikost stahovaných obrázků. Ukazují, zda je potřeba optimalizovat zobrazování obrázků, pomocí jejich zmenšení nebo pomocí jiných optimalizačních technik.
- *domains* – Počet domén, na které byly posílány požadavky během zpracování stránky.

Z výstupů nástroje Phantomas je možné zjistit také informace o zpracování kódu Javascriptu. V případě, že je použita knihovna jQuery, tak analyzuje její chování a zobrazí výstupy. Phantomas dokáže rovněž sledovat všechna přesměrování, která byla provedena během zpracování stránky. Analyzuje využití cookies i lokálního úložiště prohlížeče. Při práci s mezipamětí (caching) webového prohlížeče dokáže Phantomas poskytnout informace o aktuálním nastavení pomocí meta-tagů použitých ve stránce.
Celkově je Phantomas velmi silný nástroj, ale je potřeba mít zkušenosti s optimalizací webových aplikací, aby bylo možné správně identifikovat úzká hrdla webové stránky a navrhnout potřebné úpravy.

4.4.2 Y-Slow

YSlow je nástroj pro výkonovou analýzu webové stránky podle doporučení serveru Yahoo! (9). Tento nástroj nejprve otestuje stránku a připraví výstup, který obsahuje naměřené hodnoty včetně doporučení, které změny by měly být provedeny, aby byla webová aplikace rychlejší.

```
TAP version 13
1..24
not ok 1 C (71) overall score
not ok 2 F (32) ynumreq: Make fewer HTTP requests
  ---
  message: This page has 10 external Javascript scripts.  Try combining
them into one.
This page has 3 external stylesheets.  Try combining them into one.
This page has 18 external background images.  Try combining them with
CSS sprites.
  ...
not ok 3 F (0) ycdn: Use a Content Delivery Network (CDN)
  ---
  message: There are 69 static components that are not on CDN. <p>You
can specify CDN hostnames in your preferences. See <a
href="https://github.com/marcelduran/yslow/wiki/FAQ#wiki-faq_cdn">YSlow
FAQ</a> for details.</p>
  offenders:
    - "www.nasipolitici: 64 components, 1118.8K"
    - "connect.facebook.net: 1 component, 169.2K"
    - "www.google.com: 3 components, 29.1K (27.0K GZip)"
    - "pr.prchecker.info: 1 component, 0.5K"
  ...
ok 4 A (100) yemptysrc: Avoid empty src or href
not ok 5 F (0) yexpires: Add Expires headers
```

Obrázek 4 - Výstup z programu YSlow (zdroj: vlastní tvorba)

Výstup aplikace YSlow se může zdát na první pohled trochu nepřehledný, ale dají se v něm dozvědět zajímavá doporučení na zlepšení aplikace. Na rozdíl od nástroje Phantomas poskytuje YSlow interpretace naměřených výsledků a krátká doporučení, jak dosáhnout zlepšení. V některých případech odkazuje na webové zdroje, kde se dá najít řešení daného nedostatku.

YSlow se při analýze webové stránky zaměřuje na soubory stahované společně se stránkou. Kontroluje, zda jsou soubory se styly a skripty na stránce sloučeny a zmenšeny. V případě obrázků, které jsou používány jako pozadí prvků, doporučuje vytvoření Image sprite. Jak fuguje CSS sprite je vysvětleno v následující kapitole. Rovněž doporučuje u další souborů, jako jsou obrázky nebo připojené soubory, využít CDN (Content Delivery Network). Content Delivery Network, je síť úložišť, která distribuují statické soubory. Výhoda CDN je v tom, že při jejich použití klesá zátěž aplikačních serverů, protože soubory jsou distribuovány z jiných serverů. Rovněž dokáží garantovat rychlou dostupnost souborů v různých geografických lokacích. CDN se využívá hlavně u

aplikací s vysokou návštěvností a globálním dosahem. Typickým příkladem aplikací hojně využívajících služeb CDN jsou například sociální sítě.
YSlow existuje nejen jako rozšíření PhantomJS, ale i jako doplněk do různých prohlížečů nebo samostatná aplikace analyzující soubory typu HAR.
Tento nástroj je vhodnější pro běžné vývojáře díky přehlednému výstupu a srozumitelnou interpretací výsledků měření.

4.5 Xdebug

Xdebug je ladící a profilovací nástroj pro PHP. Společně s ladícími informacemi Xdebug poskytuje vývojářům i další informace jako například:

- spotřeba paměti daného PHP skriptu,
- celkový počet volání každé funkce,
- celkový čas strávený vykonáváním funkce,
- kompletní trasování funkce (10).

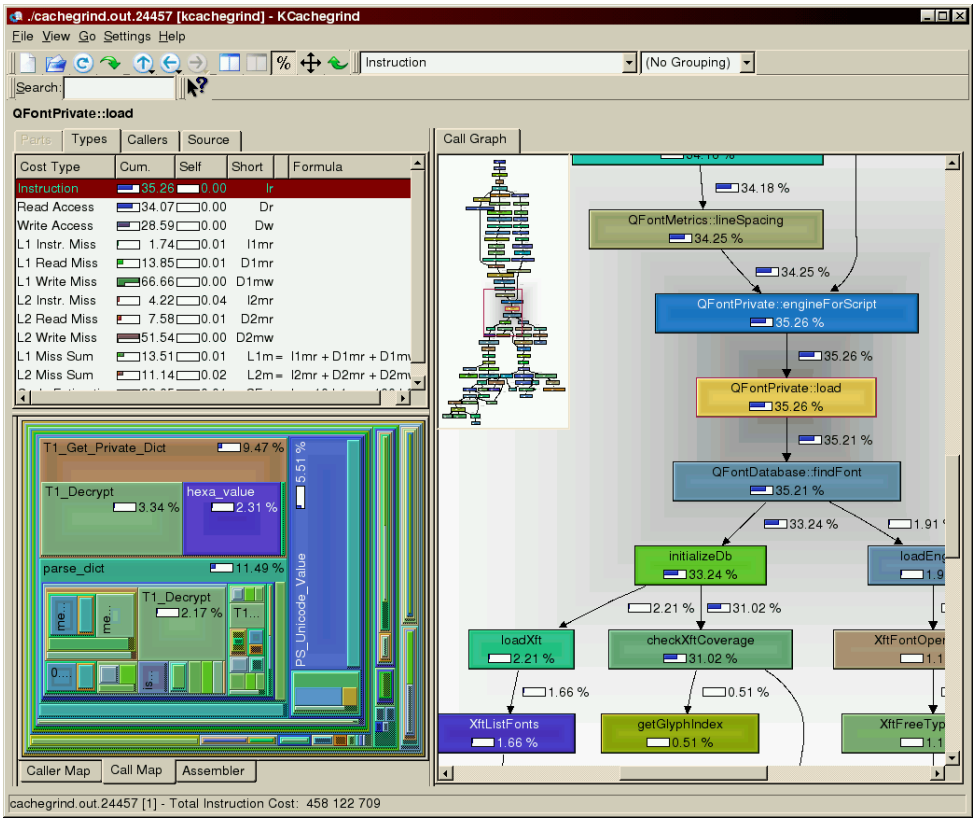
Tento nástroj je díky výše zmíněným vlastnostem vhodný pro odhalování úzkých hrdel webové aplikace.
Výstup měření Xdebug (Obrázek 5) je pouze textový soubor obsahující naměřená data, ve kterých se uživatel programu těžko orientuje. Proto se na analýzu naměřených dat používají aplikace, které umožňují pracovat s těmito daty lepším způsobem. Tyto aplikace nejen, že umí zobrazovat data získaná z měření, ale i umožňují jednoduché procházení trasování skriptu.

```
…
fn=Loader::LoadClass
36 14098
cfl=php:internal
cfn=php::in_array
calls=1 0 0
38 2
cfl=php:internal
cfn=php::basename
calls=1 0 0
40 7
cfl=php:internal
cfn=php::strtolower
calls=1 0 0
40 1
cfl=php:internal
cfn=php::file_exists
calls=1 0 0
41 5700
…
```

Obrázek 5 - Výstup z programu Xdebug (zdroj: vlastní tvorba)

4.5.1 KCachegrind

KCachegrind je nástroj na vizualizaci výstupů z různých nástrojů pro profilování aplikací pro platformu Linux. Tento nástroj je určen hlavně na profilování aplikací napsaných v jazyce C/C++, ale umí zobrazovat i data nasbíraná profilovacími nástroji jiných jazyků (11). Díky tomu je možné KCacheGrind použít na analýzu výstupu z Xdebug.



Obrázek 6 - Ukázka rozhraní KCachegrind
(zdroj: 12)

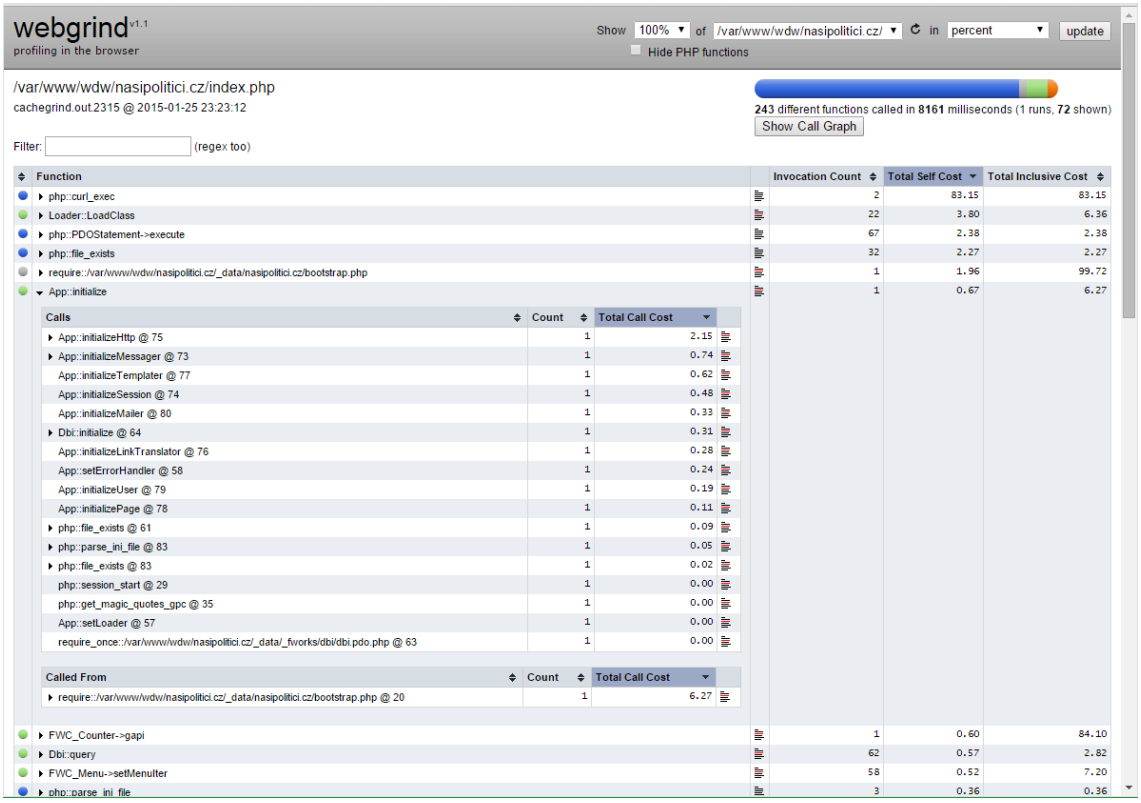
Jak je vidět na ukázce (Obrázek 6), aplikace poskytuje opravdu komplexní pohled na data získaná profilovacím nástrojem včetně praktického zobrazení volaných funkcí pomocí stromového zobrazení.

Pro platformu Windows existuje port aplikace s názvem WinCacheGrind, který ale neobsahuje veškerou funkcionalitu jako má KCachegrind.

4.5.2 Webgrind

Webgrind je grafické rozhraní pro Xdebug založené na webových technologiích. Hlavní výhodou oproti ostatním nástrojům je možnost prohlížet export přímo na serveru, který profilujeme, nemusíme tedy přenášet soubory na jiné zařízení. Umí rovněž otevřít přímo soubor se skriptem a dohledat místo, kde se zmiňovaný kód nachází. Rozhraní aplikace (Obrázek 7) je jednoduché, ale i přes to poskytuje všechny potřebné informace, které při ladění využijeme. Na ukázce je vidět, jak Webgrind zobrazuje závislosti zkoumané

funkce – zobrazí odkaz na funkce, které dále volá, i funkci, která zkoumanou funkci zavolala.



Obrázek 7 - Ukázka rozhraní Webgrind (zdroj: vlastní tvorba)

5 ANALÝZA EXISTUJÍCÍ APLIKACE

5.1 Popis aplikace

Analyzovaná webová aplikace je veřejně dostupná webová stránka. Aplikace je naprogramována v jazyce PHP a dodavatel aplikace použil proprietární framework. Analýza byla tím pádem ztížena, protože bylo potřeba analyzovat chování frameworku, abychom nevhodně neinterpretovali výsledky analýz a měření.

5.2 Měření

Veškerá měření testované aplikace jsou měřena na speciálním testovacím serveru. Jako testovací prostředí byl zvolen virtualizovaný počítač na běžném notebooku. Jako operační systém byl použit systém CentOS, který je open-source verzí Red Hat Enterprise Linuxu, a je běžně používán na serverových stanicích. Webovou aplikaci pohání webový server Apache Web Server společně s databází MySQL. Celkový výkon virtualizovaného stroje je mnohem nižší než výkon produkčního serveru, na kterém je provozována webová aplikace. Takto nastavené testovací prostředí sice neodpovídá doporučení odborníků, ale nižší výkon pomůže rychleji odhalit úzká hrdla aplikace. Měření bylo prováděno vždy přímo z virtualizovaného stroje.

5.2.1 Měření serverové části

Výstup Apache Bench je při měření aplikací, kde se zaměřujeme na optimalizaci serverové části aplikace, brán jako rozhodující faktor. Jak již bylo zmíněno v popisu nástroje, Apache Bench testuje pouze rychlost odpovědí serveru, nikoliv však kompletní zatížení, které vygeneruje prohlížeč stahující i další časti stránky jako obrázky, klientské skripty, styly atd. Před další analýzou a úpravami byly naměřeny referenční hodnoty, o které se budou opírat všechny změny na straně serveru.

Rámcové shrnutí měření před úpravami je zobrazeno v následující tabulce (Tabulka 1). Nízké hodnoty požadavků za vteřinu i vysoká průměrná doba zpracování požadavku jsou způsobeny nejen nízkým výkonem testovacího stroje, ale i chováním aplikace. Kompletní výstupy měření se nachází v přílohách (Přílohy 1,2,3).

Tabulka 1 - Výsledky měření před optimalizací (zdroj: vlastní měření)

	Požadavků za s (průměr)	Průměrná doba zpracování požadavku [ms]
Hlavní stránka	1,13	17732,988
Stránka s článkem	2,99	6692,051
Detail politika	1,83	10910,637

5.2.2 Měření klientské části

Výstup z programu YSlow nedopadl dobře. Aplikace nás upozornila na značné množství problematických míst v klientské části aplikace, která by bylo vhodné optimalizovat. Jeden z hlavních problémů je v množství stahovaných souborů ze serveru. V případě námi testované webové stránky se jedná o 10 souborů s JavaScriptem, jejichž počet doporučuje YSlow snížit nebo úplně sloučit do jediného. Obrázky, které jsou používané jako pozadí elementů, doporučuje složit do tzv. Image sprites – to jsou kolekce obrázků v jednom obrázku.

Tato technika se nejčastěji využívá u různých navigačních prvků a tlačítek, kdy se do jednoho souboru naskládá více prvků včetně jejich variant a následně se používá jeden obrázek, u kterého je měněna pozice pozadí. Při vhodném použití dokáže image sprite výrazně snížit datovou náročnost webové stránky.

Dalším doporučením je použít gzip kompresi pro přenos dat ze serveru – toto opatření může přispět ke zrychlení webu.

Posledním doporučením YSlow je provedení minifikace CSS a JavaScript souborů – jedná se o odstranění všech „whitespace“ znaků (mezery, tabulátory, odřádkování), které způsobují, že je soubor roztažený. Na produkčních serverech je nevhodné ladit skripty, a proto není potřeba, aby se kdokoliv v souboru se skriptem lehce orientoval. Tato úprava má více efektů, nejen že bude přenos dat o něco menší než u nezmenšeného souboru, ale jak bylo zmíněno výše, soubor nebude čitelný, a z toho důvodu je menší riziko případného zneužití skriptů použitých na stránce.

5.2.3 Profilování serverové části

Pro profilování serverové části byl použit program Xdebug a výsledky měření byly analyzovány v aplikaci Webgrind. Analýza probíhala na stránce s detailem politika, protože je to, hned po hlavní stránce, nejčastěji navštěvovanou stránkou webu.

Profilování aplikace ukázalo, že nejnáročnějšími operacemi aplikace jsou funkce, které využívají funkci *curl_exec*, která zprostředkovává dotazy na externí zdroje. V případě stránky s detailem politika zabírá načtení informací o něm 37 % celkového času běhu aplikace. O načtení informací o politikovi se stará funkce *search*. Tato funkce obsahuje mimo jiné i výše zmíněnou funkci *curl_exec*, která zabírá 95% celkového času spotřebovaného celou funkcí *search*. Příčinou použití funkce *curl_exec* je potřeba načtení dat z externího API.

Druhou nejnáročnější operací je komunikace s databází. Při načítání testované stránky bylo vykováno celkem 205 dotazů na databázi, které zabraly 23 % času zpracování stránky.

Třetí nejnáročnější operací při zpracování požadavku bylo vykreslení dat do šablony webové stránky – funkce *renderTemplate* třídy *Smarty_Internal_Template*. Vykreslení šablon bylo voláno 91krát a tyto operace si vyžádaly 8 % času vykreslování stránky.

Posledními funkcemi, které zabraly více než 5 % celkového času zpracování požadavku, jsou funkce *file_exists* a *filemtime*, které zabraly shodně 7 % času.

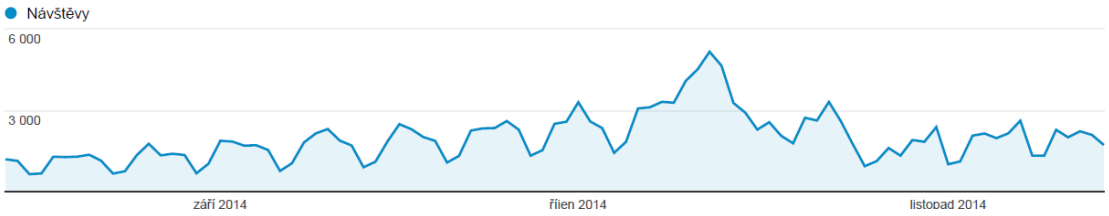
5.3 Analýza kódu

Při konzultaci s provozovateli webu bylo doporučeno zaměřit se i na externí API jiných podobně zaměřených služeb, které zkoumaná webová stránka využívá. Při prohledávání kódu bylo nalezeno několik míst, která nebyla úplně vhodně řešena a mohla by způsobovat zpomalení aplikace. Jednalo se především o synchronní požadavky na API externích služeb.

5.4 Výstup z Google Analytics

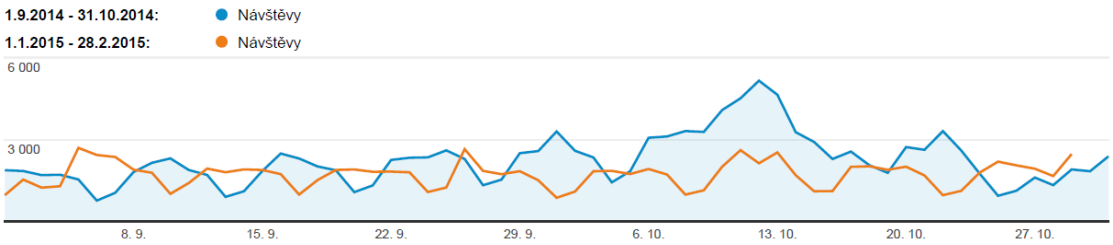
Provozovatelem webu byl poskytnut přístup do administrace Google Analytics, aby bylo možné určit počet návštěv webu v určitém časovém období – nejdůležitější je období před a během voleb do senátu, které proběhly v říjnu 2014.

Na obrázku (Obrázek 8) je zobrazen počet denních návštěv od 14. 8. 2014 do 14. 11. 2014. Je možné si všimnout, že srpnové hodnoty oscilují od 700 do 1500 návštěv denně, ale s blížícími se volbami se tyto hodnoty postupně zvyšují až na dvojnásobek. Nejvyšší počet návštěv je 12. října, to web navštívilo 5 156 návštěvníků. Takováto hodnota je již trojnásobkem běžné denní návštěvnosti webu.



Obrázek 8 - Denní počet návštěv před a během voleb v říjnu 2014 (zdroj: Google Analytics)

Další obrázek porovnává období voleb do senátu v říjnu 2014 s běžným provozem během ledna a února roku 2015. Zde můžeme sledovat nárůst běžného provozu způsobený pravděpodobně politickými kauzami a propagací webu ze strany provozovatele. Každopádně je zde vidět rozdíl mezi běžnou denní návštěvností a návštěvností před volbami, který je více než dvojnásobkem běžné denní návštěvnosti webu.



Obrázek 9 - Porovnání denního počtu návštěv během voleb a mimo ně (zdroj: Google Analytics)

Pro návrh úprav je potřeba uvažovat fakt, že běžná denní návštěvnost se pohybuje zhruba mezi 1 000 – 2 500 návštěv webu denně a během voleb může dosahovat až 6 000 návštěv za jeden den.

5.5 Hosting

Testovaná webová aplikace běží v produkční verzi na virtuálním stroji, jehož poskytovatel ho charakterizuje jako vhodný pro následující služby:

- Server pro web multikina s on-line rezervačním systémem, který využívá přes 1 500 lidí denně.
- Server webdesignéra pro 40 webů a e-shopů svých klientů postavených na systémech Joomla a OXID eShop.
- Server webdesignérského studia pro 50 na míru postavených webů a e-shopů svých klientů, největší z nich s denní návštěvností kolem 800 uživatelů (12).

Konkrétní parametry virtuálního stroje jsou:

- 2 přidělená jádra procesoru,
- 4 GB RAM,
- 100 GB prostoru na disku (30 GB pro systém a databázi a 70 GB pro uživatelská data),
- maximální rychlost linky 1 Gbit/s.

Na serveru je nasazen LAMP server, tzn. server má jako operační systém nainstalován Linux Debian, webové služby zajišťuje Apache Web Server, databáze běží na MySQL serveru a o skriptování webových stránek se stará PHP.

6 NÁVRH A IMPLEMENTACE OPTIMALIZOVANÉHO ŘEŠENÍ

6.1 Optimalizace webového serveru

Nedílnou součástí optimalizací webových aplikací je i nutnost zaměřit se na server, na kterém aplikace běží.

6.1.1 Popis funkce webového serveru

Webové servery musí při každém příchozím požadavku provést několik procesů dle specifikace protokolu HTTP. Tato sekvence kroků se nazývá „Request Processing Pipeline“ (volně přeloženo jako sekvence zpracování požadavku) a popisuje, které akce je potřeba vykonat při zpracovávání požadavku na server. Každý webový server má tyto kroky implementované rozdílně, ale v principu se skládají ze stejných částí.

- *Naslouchač požadavků (Request Listener)*. Tato komponenta je zodpovědná za zachycení síťového připojení od prohlížeče a přečtení požadavku ze socketu.
- *Analýza požadavku (Request Parsing)*. Vezme požadavek a rozdělí ho do datové struktury, aby mohl být jednoduše interpretován ostatními částmi webového serveru. Většina webových serverů poskytuje tato data běžícím aplikacím jako objekt požadavku.
- *Filtr vstupu (Input Filter)*. Tato komponenta má odpovědnost za veškeré transformace, které mají být aplikovány na požadavek. Pokud server umí například přepisovat URL (URL rewriting), tak bývá realizována právě v této části.
- *Mapování URI (URI Mapping)*. V tomto místě jsou URI mapovány na fyzické adresáře a soubory na serveru. V této části přichází na řadu například bezpečnost nebo nastavení adresářů.
- *Zpracování požadavku (Request Handling)*. Webový server načte obsah stránky z disku nebo dočasné paměti (pokud ji podporuje).
- *Výstupní filtr (Output Filter)*. Pro dynamické jazyky jako právě PHP je toto místem, kde se provede transformace zdrojového kódu do výstupního HTML. Pokud má server zapnutou kompresi výstupu, tak je prováděna taky zde.

- *Přenos výstupu (Output Transmission)*. Poslední fáze zpracování požadavku na server, v této části se vygenerovaný obsah přenese do prohlížeče.

Webový server kromě pouhého zpracování požadavku umí i další věci, jako například zapisovat do žurnálu přístupů, zapisovat chyby do žurnálu chyb nebo provádět kompresi výstupu (10).

6.1.2 Typy webových serverů

Hardware každého serveru se liší. Některé servery mají více procesorů a velkou RAM, jiné jenom jeden procesor a velice omezenou RAM. S těmito odlišnostmi se nesou i rozdíly mezi typy jednotlivých webových serverů. A pro určité konfigurace hardwaru je vhodné použít jiný typ webového serveru.

První typ je Prefork/Fork webový server. Je to procesově založený webový server, který pro každý příchozí požadavek vytvoří fork procesu pro splnění požadavku. Tento typ je doporučeno používat na serverech s jedním CPU.

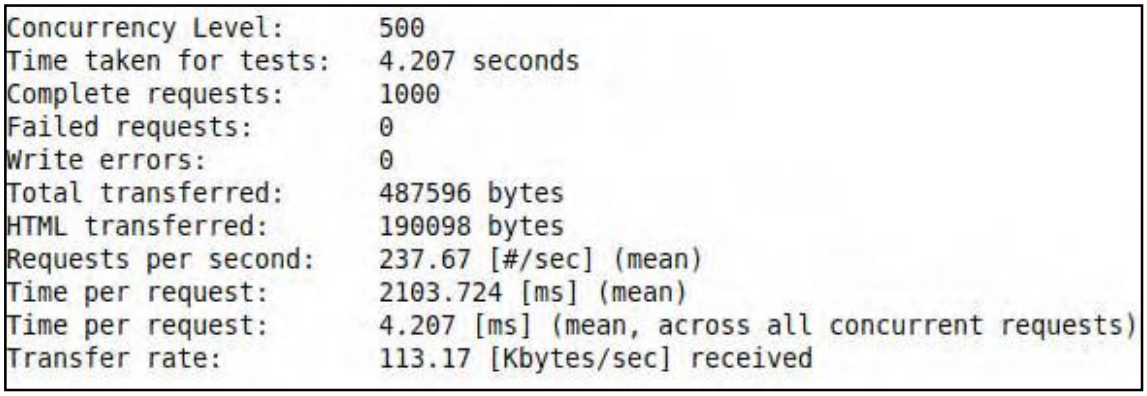
Druhým typem je Threaded web server, který pro každý požadavek vytvoří nové vlákno, které se stará o zpracování požadavku. Na rozdíl od typu Prefork výkon tohoto typu serveru roste na mutli-procesorových systémech (10).

6.1.3 Porování Apache Web Server a Nginx

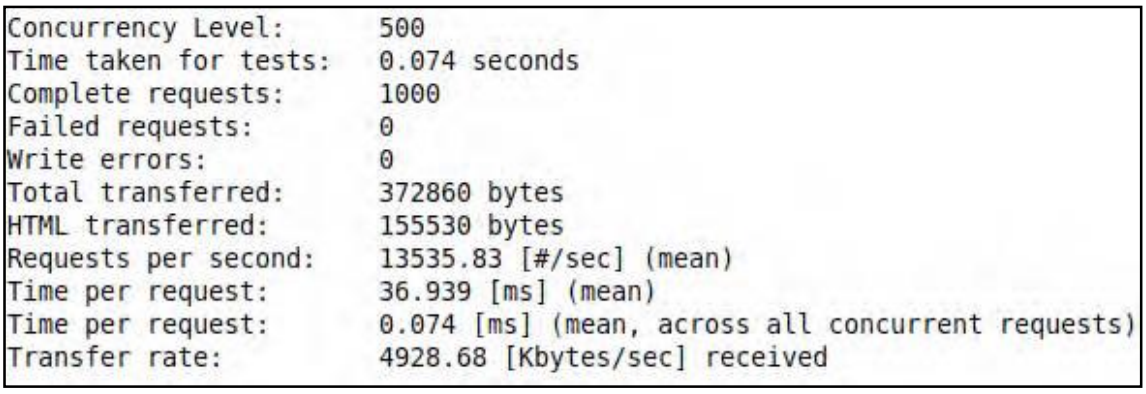
Apache Web Server je světově nejpoužívanější webový server. Je to typický příklad Prefork serveru. Jeho hlavní výhodou je jeho komplexnost a možnost rozšiřovat jeho funkcionalitu pomocí modulů, a to i za běhu webového serveru. Apache Web Server by se dal charakterizovat jako webový víceúčelový software. Jeho komplexnost je ve většině případů spíše na škodu, protože díky tomu nepatří server mezi nejrychlejší a ve většině případů jeho potenciál provozovatelé webů nevyužijí.

Nginx je asynchronní webový server. To znamená, že na rozdíl od Apache při souběžných příchozích požadavcích vytvoří pouze málo dalších vláken anebo využívají pouze jedno stávající. Díky tomu má server nízkou spotřebu RAM při velkých zátěžích. Nevýhodou serveru Nginx je při provozu PHP aplikace nutnost připravit zvláštní FastCGI server, který bude zpracovávat skripty – Apache má podporu PHP pomocí modulu mod_php.

Dle měření (10) je Nginx při obsluze statických (neskriptovaných) souborů mnohonásobně rychlejší než Apache, velkou roli zde hraje to, že Nginx je asynchronní a při čekání na načtení souboru nečeká celý proces, ale může přijímat další požadavky.



Obrázek 10 - Výsledek Apache web server - statické soubory (zdroj: 11)



Obrázek 11 - Výsledek Nginx - statické soubory (zdroj: 11)

Na obrázcích (Obrázek 10, Obrázek 11) jsou zobrazeny výstupy měření prováděné autorem knihy (10). Při porovnávání těchto měření jsou rozhodující hodnoty počtu obsloužených požadavků. V případě statických souborů je rychlejší Nginx a to v poměru 13 535 ku 237 vyřízených požadavků.

Při použití PHP skriptu není předpokládán tak vysoký rozdíl mezi počtem zpracovaných požadavků, protože oba dva servery používají pro zpracování FastCGI server. Jen přístup k němu je rozdílný.

```
Server Software: Apache/2.2.16
Server Hostname: localhost
Server Port: 80
Document Path: /test.php
Document Length: 88890 bytes
Concurrency Level: 500
Time taken for tests: 16.961 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 89081000 bytes
HTML transferred: 88890000 bytes
Requests per second: 58.96 [#/sec] (mean)
Time per request: 8480.315 [ms] (mean)
Time per request: 16.961 [ms] (mean, across all concurrent
requests)
Transfer rate: 5129.12 [Kbytes/sec] received
```

Obrázek 12 - Výsledek měření Apche Web Server (zdroj: 11)

```
Server Software: nginx/0.7.67
Server Hostname: localhost
Server Port: 80
Document Path: /test.php
Document Length: 88890 bytes
Concurrency Level: 500
Time taken for tests: 9.152 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 89048000 bytes
HTML transferred: 88890000 bytes
Requests per second: 109.27 [#/sec] (mean)
Time per request: 4575.926 [ms] (mean)
Time per request: 9.152 [ms] (mean, across all concurrent
requests)
Transfer rate: 9502.00 [Kbytes/sec] received
```

Obrázek 13 - Výsledek měření Nginx Server (zdroj: 11)

Provedené testy (Obrázek 12 a Obrázek 13) autorem knihy (10) ukazují, že rozdíl mezi počtem zpracovaných požadavků je opravdu mnohem nižší než v případě statických souborů. Server Nginx zvládl odbavit 109 požadavků za vteřinu, což je o 94 % více než v případě Apache Web Serveru, který dokázal zpracovat pouze 58 požadavků za vteřinu.

6.2 Optimalizace vstupně-výstupních operací

Při profilování testované aplikace byla nalezena místa, která značně zpomalují zpracování požadavku na aplikaci. V první řadě to jsou dotazy na externí API, u kterých je nejvhodnější řešení použití cache. Více o úpravách a použití Opcode cache se nachází v následující kapitole.

6.2.1 Optimalizace databázových operací

Druhou nejnáročnější operací bylo vykonávání dotazů na databázi. Během načítání stránky s detailem politika bylo provedeno celkem 205 dotazů na databázi. Takový počet dotazů je příliš vysoký na tak jednoduchou stránku, proto bylo potřeba najít místa, kde se provádí zbytečně moc databázových operací.

První místo, které se jeví jako jasný adept, je načítání stromové struktury menu. V testované aplikaci se nejprve načtou kořenové položky, které se následně prochází a načítají se k nim položky podřízené. Celý proces probíhá rekurzivně, aby bylo pokryto nekonečně mnoho zanoření.

Existuje několik možných typů úprav, které mohou toto neoptimální řešení nahradit. První možnost je upravit dotaz tak, aby vracel kompletní strom zanoření menu (Hierarchical query). Zde je možné narazit na zásadní problém, kterým je omezený počet zanoření. Při implementaci tohoto řešení je potřeba přesně specifikovat maximální počet podřízených položek, který se bude načítat, a to není pro univerzální CMS systém žádoucí.

Další možností, jak zrychlit načítání je použit tzv. Nested Set Model. Tato úprava je při správném použití velmi rychlá. Tato úprava vyžaduje zásadní úpravy ve struktuře databázové tabulky. Z tohoto důvodu nebude tato úprava ve zkoumané aplikaci implementována.

Třetí možnost je načtení kompletního seznamu zobrazovaných položek menu. K přípravě stromu zanoření jsou potřeba celkem tři iterace skrze seznam položek menu. V první iteraci se prochází načtený seznam položek. Položky se připraví na příjem potomků ze stromu a následně jsou umístěny do pole, pod indexem, který odpovídá identifikátoru z databáze. Toto pole se předává do dalšího cyklu. V další iteraci se znovu prochází jednotlivé položky. Pokud daná položka má nadřazenou položku, tak je nadřazená položka nalezena v poli a do pole jejích potomků je přidána aktuální položka.

Tímto způsobem se vytvoří požadovaný strom zanoření, ale zanořené položky se v seznamu nacházejí vícekrát. A to přímo v poli všech položek a u své nadřazené položky. Proto je potřeba provést třetí iteraci nad polem, ve které jsou z pole odstraněny všechny položky, které mají nějakou nadřazenou položku. Tím se seznam položek pročistí a je připraven k vykreslení na stránce či k případnému uložení do cache.

6.2.2 Optimalizace načítání souborů

Dle provedeného měření je jednou z nejnáročnějších operací práce se soubory.
Při analýze aplikace bylo objeveno nevhodné načítání konfiguračních souborů. Konfigurace zkoumané aplikace se skládá ze tří souborů:

1) Soubor *app.ini*, který obsahuje nastavení frameworku pro konkrétní aplikaci. Zde se nastavuje například maximální počet zanoření diskuze, id specializovaných článků tak, aby se zobrazovali ve správných rubrikách, atd.
2) Soubor *links.php*, ve kterém jsou pomocí pole definovány vzory URL adres pro jednotlivé rubriky webu.
3) Soubor *system.php*, který obsahuje definice globálních konstant. Tyto konstanty obsahují konfigurace připojení na databázové servery, definice odesílatele emailových zpráv vytvořených systémem nebo typ hash funkce, který se má použít pro ukládání hesel.

Všechny tyto konfigurace by bylo vhodné sjednotit do jediného konfiguračního souboru tak, jako tomu je u známých open-source frameworků. Dle architektury frameworku zkoumané aplikace lze soudit, že na první pohled jednoduchá úprava konfiguračních souborů bude vyžadovat zásadní úpravy v architektuře. Bude potřeba kompletně změnit přístup k načítání jeho komponent.
V dalším kroku by bylo vhodné se zaměřit na načítání frameworku a jeho částí. Pro optimalizaci načítání PHP kódu je možné použít podobných technik jako v případě CSS stylů a Javascriptu – sloučení a zmenšení (minifikaci). Tuto úpravu lze provést i v případě celých frameworků. Krásným příkladem je Nette Framework, který byl distribuován právě ve minifikované verzi jako jeden soubor obsahující kompletní kód frameworku. Po provedení minifikace je možné z kódu odstranit načítání tříd frameworku a použít jediný příkaz *include* s cestou na zmenšenou verzi. Tato úprava má

svá úskalí. Jelikož zmenšená verze kompletního frameworku může mít i několik megabajtů, může načítání takového souboru zabrat více času než několika menších. Proto je dobré najít části frameworku, které jsou nejčastěji načítány, a provést minifikaci jen těchto částí. Ostatní části frameworku budou následně načítány skrze automatické načítání tříd frameworku.

6.3 Použití Opcode cache

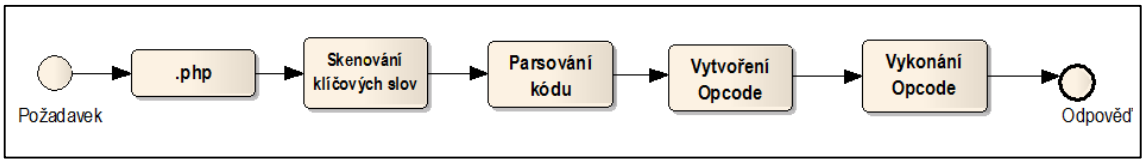
Odstraňováním nepotřebných procesů se vždy docílí lepších výsledků. To samé platí i pro zpracování skriptů v PHP. Aby bylo možné pochopit fungování mezipaměti kódu, je nejprve potřeba si vysvětlit, jak vlastně funguje zpracování PHP skriptu.

6.3.1 Životní cyklus PHP

Jazyk PHP patří do rodiny interpretovaných jazyků. To znamená, že programátorem napsaný kód se před spuštěním programu nemusí nijak zpracovávat (kompilovat). Pro spuštění skriptů se používá speciální program – interpret.

V případě PHP se využívá kombinace interpretace a kompilace kódu. Při spuštění programu v PHP interpret kompiluje skript do skupin instrukcí (v angl. *opcodes*). Tyto instrukce jsou následně jedna po druhé vykonávány. Celý proces se opakuje při každém spuštění skriptu (13).

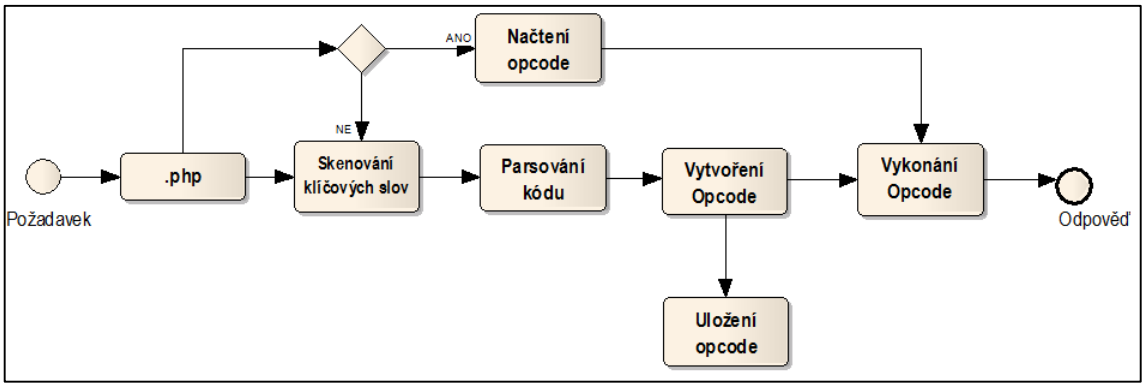
Obrázek 14 ukazuje detailní popis zpracování souboru PHP skriptu interpretem.



Obrázek 14 - Životní cykus PHP skriptu bez opcode cache (zdroj: převzato z 11)

A právě nutnost znovu překládat kód při každém spuštění skriptu vedla k vytvoření nástrojů, které zachytí přeložené kódy a uloží je do mezipaměti. Vznikly tzv. „PHP akcelerátory“, které zrychlí vykonávání skriptů napsaných v PHP pomocí mezipaměti kódu. Tyto akcelerátory jsou hojně používány pro zrychlení PHP aplikací s rozsáhlou bussiness logikou, které ke svému běhu potřebují velké množství souborů se skripty.

Samotná mezipaměť je implementována jako součást (resp. plug-in) zpracovacího mechanismu interpretu. Není potřeba žádným způsobem upravovat existující kód, ale ani není možné zasahovat do této paměti přímo z kódu skriptu.



Obrázek 15 - životní cyklus PHP s opcode cache (zdroj: převzato z 11)

Jak je vidět na obrázku (Obrázek 15), díky opcode cache mohou být přeskočeny kroky zpracování, které by se jinak musely vykonávat vždy.

6.3.2 Nasazení opcode cache

Nasadit libovolnou opcode cache by mělo být prvním krokem při řešení pomalé odezvy u aplikací ve chvíli, kdy řešíme zpomalení z důvodu velkého množství kódu.

U krátkých skriptů je rozdíl v řádech ms. Dle knihy (10) je rozdíl při plnění jednorozměrného pole s tisícem položek zhruba 0,1 ms. Opravdové zrychlení je znatelné až u velkých aplikací (desítky až stovky tisíc řádků kódu), kde se rozdíly v rychlostech pohybují v řádech sekund.

Na produkčním serveru zkoumaného webu je již opcode cache nasazena. Jedná se o APC ve verzi *3.1.13*. V tomto případě můžeme pouze upravit některá nastavení (např. míru optimalizace, velikost sdílené paměti atd.).

6.4 Ukládání proměnných do cache

Jak je uvedeno v předchozí kapitole, jedním z předpokládaných míst, kde by mohlo docházet ke zpomalení, je napojení na API z jiných služeb. Tato místa byla nalezena a pro jejich optimalizaci bylo potřeba nalézt vhodné řešení ukládání dat získaných z těchto API. Nakonec bylo zvoleno řešení pomocí in-memory cache.

Při zpracování požadavku musí aplikace načítat různá data z různých míst (databáze, API, atd.), která následně zobrazuje uživateli. V případě jednoho dotazu na databázi se musí nejprve navázat spojení s databázovým serverem, následně odeslat dotaz, zpracovat výsledek a uzavřít spojení. Celý tento proces je jeden z časově nejnáročnějších úkonů, které musí aplikace provést. Pokud načítáme data, která se často nemění (menu

aplikace apod.), můžeme načtená data uložit do paměti. Při každém dalším dotazu tato data načteme z paměti a nemusíme se znovu dotazovat databáze.
V případě zkoumané aplikace bylo ukládání do cache použito v případě API, které načítalo číselné hodnocení politiků, které se následně přepočítávalo na zobrazené hvězdy. Cache byla použita i zásadním způsobem v životním cyklu celé aplikace – tato úprava aplikace bude detailně popsána v další kapitole.
Při implementaci cache byla opět použita APC, protože je již nasazena jako opcode cache a k ukládání proměnných poslouží stejně dobře.

6.5 Request cache

Při analýze webu bylo zjištěno, že web se chová staticky – zobrazovaná data jsou stejná pro všechny návštěvníky. Zobrazovaná data se pouze načítají z databáze. Z toho vznikla myšlenka, že by bylo možné ukládat obsah vygenerované stránky. Při dalších návštěvách stránky se stejnou URL adresou by se pouze tento obsah načetl a zobrazil.
Aby bylo možné zachytávat výstup aplikace, je nutné buď využít output bufferu PHP, nebo výstup z šablonovacího systému aplikace. Obě dvě řešení jsou ekvivalentní a záleží pouze na přístupu programátora při implementaci nástroje.

6.5.1 Popis request cache

Pro zkoumanou aplikaci byl vytvořen nástroj, který poskytuje rozhraní pro ukládání obsahu stránek do APC. Nástroj umožňuje nastavení doby, po kterou bude stránka uložena v paměti. Umožňuje nastavit odlišné hodnoty pro jednotlivé stránky nebo skupiny stánek se stejným začátkem adresy. Nástroj dovoluje zadat, že má být stránka ponechána v paměti po nekonečně dlouhou dobu, respektive dokud nebude záznam smazán. To již vyžaduje spolupráci s administrací webové aplikace, aby se při případných změnách dané stránky smazal záznam z request cache. Tato možnost je vhodná hlavně pro informační stránky webu, které se mění opravdu zřídka, jako například stránka s kontakty apod.
Request cache se spouští ještě před načtením a spuštěním samotné aplikace, následně zkontroluje, zda je již stránka v paměti uložena. Pokud se požadovaná stránka nachází v paměti, je vykreslena a vykonávání dalšího kódu je zastaveno. Pokud se však stránka

v cache nenachází, tak se pokračuje ve spuštění aplikace a zpracování požadavku aplikací. Výstup aplikace se ještě před vykreslením zachytí a uloží do paměti.

6.5.2 Nasazení request cache

Zkoumaná aplikace je postavena na proprietárním frameworku od firmy, která web vytvářela. Stejně jako většina frameworků má i tento logickou strukturu uspořádání souborů s kódem. Vstupním místem každé aplikace je soubor *bootstrap*, kde se nachází spouštěcí kód aplikace. A přesně tento soubor je to správné místo pro umístění request cache. Načtení cache je prováděno samostatně, aby se ušetřilo co nejvíce času.

V případě zkoumané aplikace bylo potřeba udělat úpravy v administraci, protože některé stránky mohly být uloženy bez časového omezení. Jedná se hlavně o stránky s detaily politiků, které se až tak často nemění, ale jsou o to více navštěvovány. Jelikož je administrace navržena dost specifickým způsobem, nebylo možné to samé udělat i u stránek s textem, protože není možné identifikovat, jakou URL bude mít stránka obsahující daný text. Samotná implementace mechanismu odstraňování uložených stránek z paměti nebyla nijak obtížná, ale jak je zmíněno výše, není možné tuto úprava implementovat univerzálně pro všechny stránky.

```
// Soubor bootstrap.php nebo index.php
require_once(lib/cache/MiddlewareRequestCache.php');

$requestCache = new MiddlewareRequestCache();
try{
      echo $requestCache->getPageData($_SERVER['REQUEST_URI']);
      exit();
} catch (PageNotFoundException $e){

}

ob_start();

$settings = array(
      'static'=>array(
            '/' => 60
      ),
      'dynamic'=>array(
            '/cs/politik/' => MiddlewareRequestCache::INFINITY,
            '/cs/kauzy/detail/' => 300
      ),
      'default'=>120
);
$requestCache->init($settings);

// … kod aplikace

$requestCache->savePageData($_SERVER['REQUEST_URI'],
ob_get_contents());

ob_flush();
```

Ukázka 5 - Konfigurace request cache (zdroj: vlastní tvorba)

Ukázka (Ukázka 5) zobrazuje jakým způsobem nasadit request cache do jakéhokoliv projektu, který využívá ke zpracování požadavku dispatcher v podobě souboru *index.php* nebo *bootstrap.php*. Při implementaci requst cache je potřeba nejprve připojit knihovnu obsahující potřebné třídy do projektu a následně pomocí příkazu *require_once* načíst třídu *MiddlewareRequestCache.php*, která se stará o práci s APC. V dalším kroku je provedena inicializace třídy, která v základní konfiguraci zvládne obsloužit požadavek na načtení stránky. Následně se provede test, zda stránka s aktuálním URI není již vytvořena. Pokud již v cache stránka existuje, provede se její vykreslení a ukončí se zpracování pomocí příkazu *exit*. Jestliže se stránka v paměti nenachází, pokračuje se ve zpracování stránky. Nejprve se spustí output buffer PHP pomocí příkazu *ob_start*. Do output bufferu se ukládá kompletní výstup generovaný skriptem. Potom je vhodné

připravit konfiguraci pro ukládání stránek do mezipaměti a pomocí příkazu *init* ji vložit do request cache. Potom následuje veškerá logika webové aplikace se všemi náležitostmi. Po ukončení zpracování požadavku se pomocí funkce *savePageData* uloží do mezipaměti obsah output bufferu, který získáme pomocí příkazu *ob_get_contents*. Nakonec veškerý obsah odešleme na výstup aplikace za použití příkazu *ob_flush*.

Mazání konkrétní položky paměti lze provést pomocí funkce *invalidatePage*, které se předá URI stránky, která má být odstraněna.

Request cache je mocný nástroj při provozu webů se statickým obsahem. Pomocí této mezipaměti je možné získat vyšší výkon aplikací, aniž by bylo potřeba měnit strukturu aplikace nebo hardware serveru.

6.6 Změna hostingu

Jak již bylo zmíněno v analýze, v aktuálním stavu web běží na virtuálním serveru. Virtuální server dokáže poskytovat stabilní prostředí pro běh webových aplikací, ale problém může nastat ve chvíli, kdy je potřeba pružně reagovat na výkyvy návštěvnosti respektive zátěže webové aplikace. Ne u všech poskytovatelů je možné měnit parametry virtuálního stroje agilně v závislosti na situaci. Ve většině případů přichází nutnost objednat vyšší úroveň služby a absolvovat celé kolečko objednání znovu a případná migrace nebo alespoň restart serveru. Pak rovněž přichází problém s fakturacemi, protože většinou se virtuální server objednává na období (nejčastěji rok).

Odpověď na tyto požadavky mají cloudové služby. Jejich koncepce odpovídá požadavkům aplikací s potřebou dynamicky škálovat výkon. Cloudové služby mají velkou výhodu v tom, že platíte pouze za to, co doopravdy využijete. Při hostování aplikací je možné lehce přidávat výpočetní jednotky, které se budou starat o zpracování požadavků, aniž by bylo potřeba provádět složité objednávání dalších služeb.

Při výběru cloudové platformy pro hostování webové aplikace je dobré se zaměřit na umístění datového centra poskytovatele. Pokud je webová aplikace zaměřená lokálně, je dobré umístit aplikaci do nejbližšího datového centra. Protože zvýšení výkonu, které je možné získat přesunem aplikace do cloudu, se může ztratit kvůli velkému zpoždění, které vznikne nutností posílat data na velkou vzdálenost. Rovněž je potřeba uvědomit si fakt, že při hostování aplikace zákazník platí za výpočetní jednotku, za úložiště, databázi i za datový přenos. Takže při výpočtu ceny je třeba si dát pozor i na tyto prvky a

zahrnout je do cenové kalkulace. I když se může na první pohled zdát, že provoz webové aplikace na cloudové platformě může být dražší, ve výsledku může být nakonec provozovatel aplikace překvapen, že platí za provoz méně než u virtuálního stroje, který by byl vybaven tak, aby zvládal zátěžové špičky.

6.7 Úpravy klientské části aplikace

Možným krokem optimalizačního procesu jsou i úpravy součástí aplikace, které se stahují společně se stránkou. Jedná se hlavně o soubory s Javascriptem a kaskádovými styly. V kapitole 5 jsou popsána veškerá doporučení, pomocí kterých by se mohlo zrychlit načítání webové stránky.

Úpravy klientské části nejsou součástí úprav, které byly provedeny ve webové aplikaci, protože jejich dopad u tohoto typu webové aplikace není natolik zásadní, jako úpravy provedené na serverové části aplikace.

Při nasazení většiny ze zmíněných úprav je rovněž důležitá i součinnost vývojáře webové aplikace, protože musí připravit tým na nové postupy, které by bylo nutno provádět při jakékoliv úpravě na klientské části aplikace.

7 VÝSLEDKY TESTOVÁNÍ PO OPTIMALIZACI

Testování úprav probíhalo v speciálním testovacím prostředí, které je oddělené od produkční verze. Jelikož je tato práce zaměřena na optimalizaci serverové části webové aplikace, bylo prováděno měření pomocí Apache Bench. Při zhodnocování výsledků bude brán zřetel hlavně na rychlost odpovědi serveru na požadavek.

7.1 Výsledky testování po úpravě načítání menu

Po úpravě načítání struktury menu bylo znovu provedeno měření rychlosti odpovědi aplikace a celkové profilování aplikace. Celý výsledek měření pomocí nástroje Apache Bench je uveden v příloze (Příloha 7).

Rozdíl mezi výsledky měření před a po provedení úprav není nijak vysoký, ale i přes to je patrné určité zlepšení. Aplikace v testovacím prostředí dokázala obsloužit o 0,3 požadavku více a celkový rozptyl rychlosti odpovědí na požadavky se pohyboval od 9 576 – 17 496 ms. Při profilování aplikace byly odhaleny zásadnější změny. Celkový počet požadavků na databázi klesl z původních 205 na aktuálních 119. Tato hodnota stále není optimální a bude potřeba hledat další místa, která nehospodárně využívají databázi.

7.2 Výsledky testování po nasazení cache

V tabulce (Tabulka 2) jsou uvedeny změřené hodnoty softwarem Apache Bench po optimalizaci aplikace. Jak je již na první pohled patrné, došlo k velkému zlepšení oproti hodnotám před optimalizací. Kompletní výsledky měření jsou uvedeny v přílohách.

Tabulka 2 - Výsledky měření po optimalizaci (zdroj: vlastní měření)

	Požadavků za s (průměr)	Průměrná doba zpracování požadavku [ms]
Hlavní stránka	41,21	485,306
Stránka s článkem	62,34	320,813
Detail politika	51,68	387,023

Pokud se podíváme na výsledky měření (Přílohy 4,5,6) tak zjišťujeme, že 95 % požadavků zvládá server obsloužit do 350 ms. Přitom před optimalizací server nezvládl více jak 50 % požadavků pod 9 000 ms. Zde se naplno projevila síla cachování celých stránek.

Je nutné ovšem podotknout, že testování na produkčním serveru může přinést odlišné výsledky. Díky vyššímu výkonu serveru nebude rozdíl pravděpodobně tak velký, jako tomu je u testovacího prostředí.

8 ZÁVĚR

Zpomalení administrační části webové aplikace je způsobeno velkým počtem požadavků na databázi, které potřebuje ke svému chodu právě administrace. Ani samotný web není k využívání databáze zrovna šetrný a při příchodu více návštěvníků dochází ke značnému zpomalení odpovědi nejen webu, ale i administrace. Za slabé místo této aplikace by se dala považovat databáze respektive databázový server. Pravdou je, že nehospodárné využívání databáze ve webové aplikaci dokáže při vyšším počtu návštěvníků způsobit značné potíže s rychlostí odpovědi aplikace.

Provedené úpravy, hlavně request cache, tvoří stěžejní část veškerých úprav zkoumané webové aplikace. Vliv těchto úprav na chování aplikace je nesporný a nasazení úprav vyřeší aktuální problém zpomalování aplikace při kolísání návštěvnosti v době, kdy probíhají volby.

Při dalších úpravách aplikace je potřeba se zaměřit na optimalizaci práce s databází, a to hlavně v části webové aplikace, kde je návštěvnost nejvyšší. Administraci není potřeba v aktuálním stavu nijak upravovat, protože ji využívají pouze správci webu, takže zde není žádná velká návštěvnost, která by mohla způsobovat zpomalení aplikace.

Jako další krok úprav je možné doporučit změnu webového serveru. Jak již bylo zmíněno výše, pokud bude chtít provozovatel webu ještě více zrychlit aplikaci s minimálními náklady, může změnit webový server. Když místo aktuálně používaného Apache Web Serveru nasadí Nginx, získá v kombinaci s request cache velice rychlé odpovědi na požadavky.

Následně bude vhodné se zaměřit na optimalizaci načítání frameworku a jeho komponent včetně konfigurací. Pokud ale již nebude k plynulému provozu stačit aktuální konfigurace serveru, je nasnadě přesun na cloudové služby. Před samotným přesunem je potřeba zvážit všechny aspekty přechodu ať už technologické nebo právní.

9 REJSTŘÍK

10 SEZNAM POUŽITÉ LITERATURY

1. **Hlava, Tomáš.** Smoke testy. *Testování softwaru.* [Online] 2011. [Citace: 29. 6 2014.] http://testovanisoftwaru.cz/druhy-typy-a-kategorie-testu/smoke-testy/.

2. **Gheorghiou, Grig.** Performance vs. load vs. stress testing. *Agile Testing.* [Online] 2005. [Citace: 29. 6 2014.] http://agiletesting.blogspot.cz/2005/02/performance-vs-load-vs-stress-testing.html.

3. **Bortz, Robin.** Web Performance Testing - Test objectives and Real Life Monitoring. *SQAZONE.* [Online] 2010. [Citace: 29. 6 2014.] http://www.sqazone.net/modules/news/article.php?storyid=551.

4. **RPM Solutions Pty Ltd.** Soak Tests. *Loadtest.com.* [Online] 2004. [Citace: 14. 4 2015.] http://loadtest.com.au/types_of_tests/soak_tests.htm.

5. **Apache Foundation.** ab - Apache HTTP server benchmarking tool. *Apache.org.* [Online] 2015. [Citace: 16. 4 2015.] http://httpd.apache.org/docs/2.2/programs/ab.html.

6. —. *Apache JMeter.* [Online] 1999,2013. [Citace: 29. 6 2014.] http://jmeter.apache.org/.

7. **Hidayat, Ariay.** *PhantomJS.* [Online] 2014. [Citace: 1. 11 2014.] http://yslow.org/.

8. **GitHub.** *Phantomas by macbre.* [Online] 2014. [Citace: 1. 11 2014.] http://macbre.github.io/phantomas/.

9. **Duran, Marcel.** *Official Open Source Project Website.* [Online] 2014. [Citace: 1. 11 2014.] http://yslow.org/.

10. **Padilla, Armando a Hawkins, Tim.** *Pro PHP application performance tuning PHP Web projects for maximum performance.* Berkeley, California : Apress, 2010. ISBN 978-143-0228-998.

11. **Weidendorfer, Josef.** *KCachegind.* [Online] 2013. [Citace: 28. 1 2015.] http://kcachegrind.sourceforge.net/html/Home.html.

12. **THINline interactive.** Porovnání virtuálních serverů. *Spolehlivé servery.* [Online] 2015. [Citace: 10. 4 2015.] http://www.spolehlive-servery.cz/virtualni-servery/porovnani-virtualu/.

13. **Future Publishing Limited.** Interpreting vs. Compiling. *TuxRadar.* [Online] 2015. [Citace: 9. 04 2015.] http://www.tuxradar.com/practicalphp/2/2/2.

Příloha 1

Výstup měření Apache Bench před úpravou – Hlavní stránka

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:          /
Document Length:        25481 bytes

Concurrency Level:      20
Time taken for tests:   886.649 seconds
Complete requests:      1000
Failed requests:        989
   (Connect: 0, Receive: 0, Length: 989, Exceptions: 0)
Write errors:           0
Total transferred:      25854417 bytes
HTML transferred:       25487417 bytes
Requests per second:    1.13 [#/sec] (mean)
Time per request:       17732.988 [ms] (mean)
Time per request:          886.649 [ms] (mean, across all concurrent
requests)
Transfer rate:          28.48 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   4.4      0      54
Processing:  8540 17688 2977.8  17613   28227
Waiting:     8529 17591 2972.4  17525   28163
Total:       8541 17689 2977.2  17613   28227

Percentage of the requests served within a certain time (ms)
  50%  17613
  66%  18803
  75%  19651
  80%  20175
  90%  21374
  95%  23086
  98%  24181
  99%  24737
 100%  28227 (longest request)
```

Příloha 2

Výstup měření Apache Bench před úpravou – Stránka s článkem

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:            /cs/kauzy/detail/152-pripad-technologickeho-
centra-pisek
Document Length:        18595 bytes

Concurrency Level:      20
Time taken for tests:   334.603 seconds
Complete requests:      1000
Failed requests:        638
   (Connect: 0, Receive: 0, Length: 638, Exceptions: 0)
Write errors:           0
Total transferred:      18980018 bytes
HTML transferred:       18613018 bytes
Requests per second:    2.99 [#/sec] (mean)
Time per request:       6692.051 [ms] (mean)
Time per request:          334.603 [ms] (mean, across all concurrent
requests)
Transfer rate:          55.39 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   5.2      0      65
Processing:  4522 6646 1811.9   6059   14330
Waiting:     4451 6545 1792.2   5953   14253
Total:       4522 6647 1811.8   6060   14330

Percentage of the requests served within a certain time (ms)
  50%   6060
  66%   7032
  75%   7516
  80%   7912
  90%   8804
  95%  10458
  98%  12349
  99%  13234
 100%  14330 (longest request)
```

Příloha 3

Výstup měření Apache Bench před úpravou – Detail politika

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright     1996     Adam     Twiss,     Zeus     Technology     Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:          /cs/politik/709-marian-hosek
Document Length:        23986 bytes

Concurrency Level:      20
Time taken for tests:   545.532 seconds
Complete requests:      1000
Failed requests:        697
   (Connect: 0, Receive: 0, Length: 697, Exceptions: 0)
Write errors:           0
Non-2xx responses:      89
Total transferred:      23469321 bytes
HTML transferred:       23100808 bytes
Requests per second:    1.83 [#/sec] (mean)
Time per request:       10910.637 [ms] (mean)
Time per request:          545.532 [ms] (mean, across all concurrent
requests)
Transfer rate:          42.01 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1  11.9      0     337
Processing:  1167 10826 5202.0   9301   37684
Waiting:     1154 10756 5193.8   9242   37624
Total:       1167 10827 5202.5   9301   37684

Percentage of the requests served within a certain time (ms)
  50%   9301
  66%  11235
  75%  12660
  80%  13746
  90%  17673
  95%  21832
  98%  26266
  99%  30800
 100%  37684 (longest request)
```

Příloha 4

Výstup měření Apache Bench po úpravě – Hlavní stránka

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:          /
Document Length:        25729 bytes

Concurrency Level:      20
Time taken for tests:   24.265 seconds
Complete requests:      1000
Failed requests:        999
   (Connect: 0, Receive: 0, Length: 999, Exceptions: 0)
Write errors:           0
Total transferred:      25658795 bytes
HTML transferred:       25482895 bytes
Requests per second:    41.21 [#/sec] (mean)
Time per request:       485.306 [ms] (mean)
Time per request:           24.265 [ms] (mean, across all concurrent
requests)
Transfer rate:          1032.64 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   6.7      0      79
Processing:    88  483 1919.7    206   15691
Waiting:       79  476 1896.3    203   15522
Total:         88  485 1923.8    207   15758

Percentage of the requests served within a certain time (ms)
  50%    207
  66%    230
  75%    244
  80%    252
  90%    282
  95%    327
  98%  12705
  99%  13784
 100%  15758 (longest request)
```

Příloha 5

Výstup měření Apache Bench po úpravě – Stránka s článkem

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:            /cs/kauzy/detail/152-pripad-technologickeho-
centra-pisek
Document Length:        18616 bytes

Concurrency Level:      20
Time taken for tests:   16.041 seconds
Complete requests:      1000
Failed requests:        970
   (Connect: 0, Receive: 0, Length: 970, Exceptions: 0)
Write errors:           0
Total transferred:      18802634 bytes
HTML transferred:       18626734 bytes
Requests per second:    62.34 [#/sec] (mean)
Time per request:       320.813 [ms] (mean)
Time per request:          16.041 [ms] (mean, across all concurrent
requests)
Transfer rate:          1144.71 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   2.2      0      19
Processing:   101  320 771.8    206    7159
Waiting:       99  317 770.1    204    7157
Total:        101  320 773.5    206    7174

Percentage of the requests served within a certain time (ms)
  50%    206
  66%    229
  75%    246
  80%    256
  90%    287
  95%    318
  98%   4879
  99%   5144
 100%   7174 (longest request)
```

Příloha 6

Výstup měření Apache Bench po úpravě – Detail politika

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient)


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:          /cs/politik/709-marian-hosek
Document Length:        24001 bytes

Concurrency Level:      20
Time taken for tests:   19.351 seconds
Complete requests:      1000
Failed requests:        928
   (Connect: 0, Receive: 0, Length: 928, Exceptions: 0)
Write errors:           0
Non-2xx responses:      5
Total transferred:      21492648 bytes
HTML transferred:       21316663 bytes
Requests per second:    51.68 [#/sec] (mean)
Time per request:       387.023 [ms] (mean)
Time per request:           19.351 [ms] (mean, across all concurrent
requests)
Transfer rate:          1084.63 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   4.7      0      63
Processing:    20  386 1822.8    153   18480
Waiting:       20  384 1821.1    151   18479
Total:         20  386 1826.4    153   18499

Percentage of the requests served within a certain time (ms)
  50%    153
  66%    177
  75%    194
  80%    202
  90%    231
  95%    257
  98%   4555
  99%  15522
 100%  18499 (longest request)
```

Příloha 7

Výstup měření Apache Bench po úpravě načítání menu – Detail politika

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright     1996     Adam     Twiss,     Zeus     Technology     Ltd,
http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking www.nasipolitici (be patient).....done


Server Software:        Apache/2.2.15
Server Hostname:        www.nasipolitici
Server Port:            80

Document Path:          /cs/politik/709-marian-hosek
Document Length:        13462 bytes

Concurrency Level:      20
Time taken for tests:   46.691 seconds
Complete requests:      100
Failed requests:        94
   (Connect: 0, Receive: 0, Length: 94, Exceptions: 0)
Write errors:           0
Non-2xx responses:      15
Total transferred:      2225654 bytes
HTML transferred:       2188699 bytes
Requests per second:    2.14 [#/sec] (mean)
Time per request:       9338.231 [ms] (mean)
Time per request:          466.912 [ms] (mean, across all concurrent
requests)
Transfer rate:          46.55 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5  12.7      0      41
Processing:  3387 8961 2572.9   9576   17496
Waiting:     3322 8887 2562.9   9495   17394
Total:       3387 8966 2572.0   9576   17496

Percentage of the requests served within a certain time (ms)
  50%   9576
  66%   9783
  75%   9869
  80%   9959
  90%  10765
  95%  15849
  98%  16834
  99%  17496
 100%  17496 (longest request)
```

Příloha 8

Obsah CD

- *Zdrojove kody* – Složka obsahuje nové a změněné soubory aplikace.
- *Výstupy před úpravou* – složka obsahující naměřené výstupy před úpravou aplikace
- *Výstupy po úpravě* – složka obsahuje naměřené výstupy po úpravě aplikace