VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

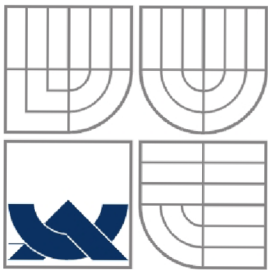
MOŽNOSTI OPTIMALIZACE VÝKONU LAMP (LINUX/APACHE/MYSQL/PHP)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

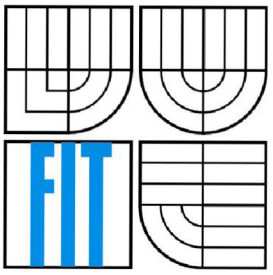
AUTOR PRÁCE
AUTHOR

Bc. PAVEL KOTLÁŘ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOŽNOSTI OPTIMALIZACE VÝKONU LAMP (LINUX/APACHE/MYSQL/PHP)

OPTIMIZATION OF LAMP (LINUX/APACHE/MYSQL/PHP)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PAVEL KOTLÁŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2009

Zadání práce:

1. Prostudujte problematiku výkonu, zátěže a optimalizace systémů LAMP.
2. Vytvořte modelovou aplikaci (sadu aplikací) v prostředí LAMP, které umožní vyhodnocovat optimalizace výkonu systému.
3. Optimalizujte výkon systému k dosažení nejlepších parametrů chodu modelové aplikace (zaměřte se na: nastavení na úrovni OS, nastavení Apache, nastavení PHP, nastavení MySQL).
4. Shrňte dosažené výsledky optimalizací do sady doporučení pro optimalizaci systému typu LAMP.
5. Aplikujte vytvořená doporučení na reálný systém (spolupráce s QCM); vyhodnoťte přínos dosažený optimalizací.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Abstrakt

Tato diplomová práce se zabývá tematikou optimalizace výkonu serverové platformy LAMP. Postupně se ve všech čtyřech součástech této platformy (v Linuxu, HTTP serveru Apache, MySQL databázi a interpretu jazyka PHP) snaží odhalovat problémová místa při běhu modelové webové aplikace vytvořené jako součást práce a odstraňovat je vhodným nastavením konfiguračních voleb či nasazením technologií zvyšující výkon dané součásti. Na základě takto získaných zkušeností je vypracována sada optimalizačních doporučení. Ty jsou nakonec ověřeny jejich aplikací na server provozující reálné webové aplikace.

Abstract

This work deals with topic of LAMP software bundle performance optimalization. Step by step, it tries to discover performance problems in all four parts of LAMP (in Linux, HTTP server Apache, MySQL database and PHP language interpreter). A model web application is created for these testing purposes. When a problem is found, a change in configuration files is done or a performance improving technology is applied to the corresponding part. A set of optimalization recommendations is compiled and verified on server running real web application.

Klíčová slova

Linux, Apache, MySQL, PHP, optimalizace, reverzní proxy, PHP akcelerátor

Keywords

Linux, Apache, MySQL, PHP, optimalization, reverse proxy, PHP accelerator

Citace

Kotlář Pavel: Možnosti optimalizace výkonu LAMP (Linux/Apache/MySQL/PHP), diplomová práce, Brno, FIT VUT v Brně, 2009

Možnosti optimalizace výkonu LAMP (linux/apache/mysql/php)

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedenímIng. Adama Herouta, Ph.D.
Další informace mi poskytl Ing. Jiří Vrba, Ph.D. (QCM s.r.o.).
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Pavel Kotlář
18.5.2009

Poděkování

Touto cestou bych rád poděkoval Ing. Adamovi Heroutovi, Ph.D. a Ing. Jiřímu Vrbovi, Ph.D. za množství praktických a cenných rad a připomínek, které mi v průběhu vypracování této diplomové práce poskytli.

Obsah

1 Úvod

Moderní webové aplikace bývají často poměrně náročné na výpočetní výkon. Když k tomu přidáme prudce rostoucí počet uživatelů, kteří využívají připojení k internetu, dospějeme k závěru, že provozovatelé populárních webových aplikací mají nebo brzo budou mít velký problém s výkonem svých serverů. Výpočetní výkony serverů sice neustále rostou, ale i tak nejsou schopny soupeřit s výše zmíněnou náročností aplikací v kombinaci růstem počtu uživatelů. Jistě je možné přepsat aplikaci tak, aby nebyla tolik náročná a na chvíli je problém vyřešen. Ale jednou přijde chvíle, kdy už nebude co měnit a pak přijde na řadu nákup nového serveru nebo optimalizace výkonu serveru stávajícího.

A právě optimalizací výkonu serveru založeného na platformě LAMP, provedenou bez nutnosti podstoupení větších zásahů do zdrojového kódu PHP skriptů, se zabývá tato diplomová práce. Schopnost nakonfigurovat svůj webový server na maximální výkon dává v dnešní době tolik potřebnou ekonomickou výhodu – šetří peníze, které by jinak bylo třeba zaplatit za upgrade serverů.

1.1 Členění práce

Celá práce je logicky členěna do tematických kapitol. První – úvodní kapitola, kterou právě čtete je úvodem do problematiky a přibližuje proč a co je v této práci řešeno. Následuje druhá kapitola pojednávající obecně o platformě LAMP, aby na ni následně navázala další už s konkrétním popisem možností, kterými lze dosáhnout zvýšení výkonu dané součásti LAMP. Ve čtvrté kapitole je řešena problematika testování výkonu, popsána modelová aplikace a prezentovány výsledky testů a z nich plynoucí sada doporučení. V předposlední kapitole jsou pak popsány zkušenosti získané po aplikaci těchto doporučení na reálném serveru. Poslední kapitola shrnuje dosažené výsledky a přínosy.

2 Platforma LAMP

Existuje celá řada nejrůznějších webových serverů, databázových serverů, serverových operačních systémů a programovacích či skriptovacích jazyků, ve kterých lze vytvořit malé či naopak velmi rozsáhlé webové aplikace. Přesto velká část z nich je vyvíjena a provozována na platformě nazývané LAMP [3]. Jednoduše řečeno jde o kombinaci čtyř samostatných softwarových produktů, které tvoří mocný a snadno použitelný celek. Jako serverový operační systém je použit GNU/Linux. Funkci webového serveru zastává Apache. Jako databázový server je zvoleno MySQL. A to celé je spojuje použitý skriptovací jazyk – nejčastěji PHP. Nicméně existují i alternativní definice LAMP využívající místo PHP jazyky Perl nebo Python.

Čím si však LAMP platforma získala svou popularitu? Především ji hraje do karet její cena. Všechny součásti jsou licencovány pod svobodnými licencemi a je tedy možné je využívat zdarma. V druhé řadě pak jistě vděčí své oblíbenosti i rozsáhlé komunitě, která lidem radí, jak vyřešit jejich problémy s touto platformou. Nakonec je však třeba i zmínit solidní výkon, který je možné s tímto systémem dosáhnout a velkou konfigurovatelnost, která umožňuje maximální přizpůsobení uživatelovým potřebám.

Jak již bylo zmíněno výše, LAMP je sestaven ze čtyř součástí, které budou přiblíženy v následujících podkapitolách.

2.1 Linux

GNU/Linux je v systému LAMP použit jako serverový operační systém. Jeho základ tvoří jádro Linux. Na něm pak běží ostatní potřebné programy tvořící funkční OS. Ty pocházejí z projektu GNU. Jak jádro, tak základní obslužné programy jsou šířeny pod licencí GNU GPL a jsou tedy jako zbytek platformy dostupný bezplatně a včetně zdrojových kódů. Díky stabilitě linuxu a dostupnému zdrojovému kódu jej používají (s vlastními modifikacemi) jako serverový OS mnohé významné IT firmy jako např. Google.

2.2 Apache HTTP server

Apache HTTP server je nejrozšířenější HTTP(S) server na internetu. Podle posledních dostupných údajů z dubna 2009 [4] používán pro provoz webových prezentací u necelých 46% internetových domén. Jeho vývoj je zaštiťován neziskovou organizací Apache Software Foundation a samotný server je vydán pod Apache Licencí. Kvalita, stabilita a vhodná licence také umožnila vzniknout velké řadě komerčních odvozených produktů [5].

V současné době existují 3 vývojové větve – 1.3, 2.0 a 2.2. Jako obvykle nejvyšší verze obsahují oproti starším nové funkce a jsou u nich provedeny optimalizace výkonu. Z tohoto důvodu je pro nás nejzajímavější vývojová větev 2.2, které se v této práci budeme věnovat. Ostatní dvě jsou více či méně výběhovou záležitostí a při příští vlně upgradu serverů lze očekávat jejich náhradu právě verzí 2.2.x.

2.3 MySQL

„MySQL je databázový řídící systém patřící do kategorie Open Source software, postavený na jazyku SQL (SQL je zkratka pro Standard Query Language), který je rychlý, spolehlivý, jednoduše použitelný a vhodný pro aplikace téměř jakékoliv velikosti.“[11] Zhruba před rokem došlo ke koupi společnosti MySQL AB, která se starala o vývoj MySQL databáze, společností Sun Microsystems. Sun byl následně na konci dubna 2009 koupen společností Oracle, významným hráčem na trhu s databázovými systémy. Po těchto změnách je otázkou další směr, jakým se bude vývoj MySQL ubírat.

2.4 PHP

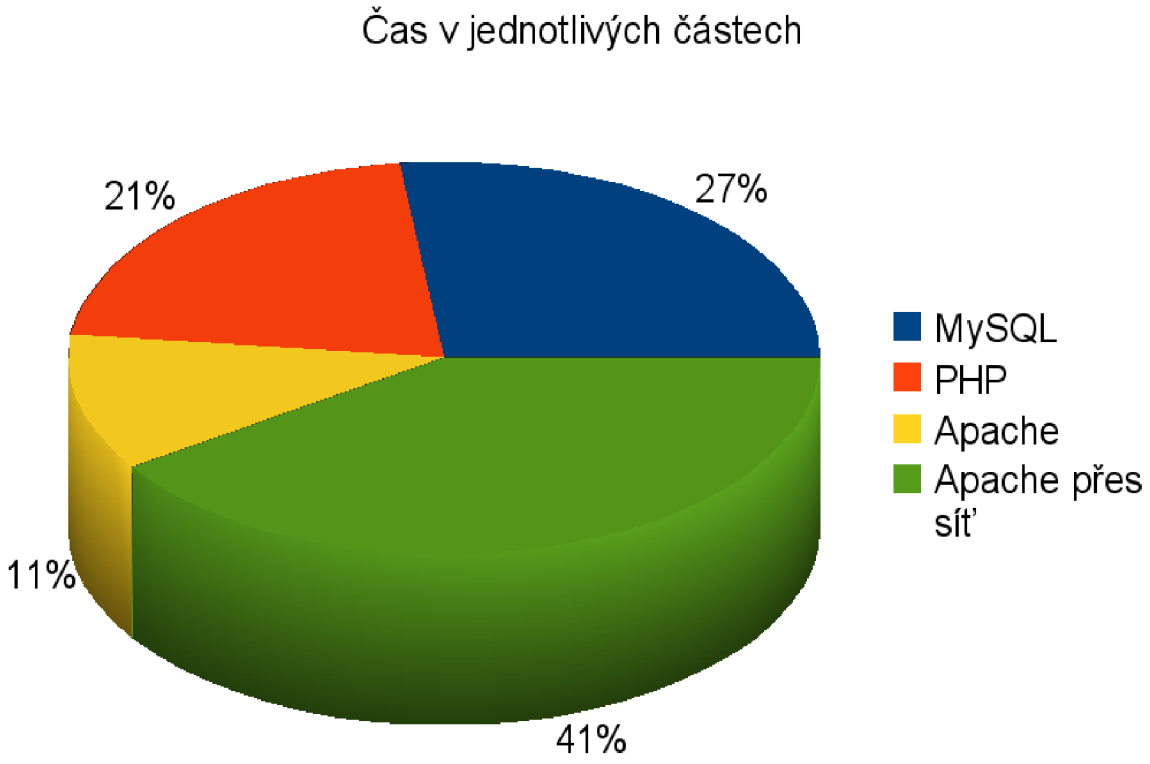
PHP je rekurzivní zkratkou PHP: Hypertext Preprocessor. Jazyk PHP je interpretovaný všeobecně použitelný skriptovací jazyk využívaný převážně k tvorbě webových aplikací. „PHP vyvinul Rasmus Lerdorf, Zeev Suraski a Andi Gutmans. První verze se objevila v roce 1994.“ [11] V současné době je nejnovější verzí PHP verze 5.2.9 a pilnpracuje se na verzi 6. „Systém PHP obsahuje celou řadu vestavěných funkcí, které provádí důležité, ale také běžné požadavky: umí se připojit k databázi (podporuje celou řadu typů databází včetně MySQL...), odeslat e-mail, vytvořit PDF dokumenty a mnohé další věci.

Systém PHP je ve světě hodně rozšířený a na webu je k dispozici nepřeberné množství výukových materiálů, příkladů a odborné pomoci. Syntaxe se podobá syntaxi jazyka Perl, C a C++ a proto zkušeným programátorům jeho zvládnutí nedělá problémy.“ [11]

3 Možnosti optimalizace

V této kapitole budou postupně rozebrány jednotlivé možnosti optimalizace dle příslušnosti k jednotlivým součástem tvořícím LAMP platformu. Vždy jedna kapitola bude věnována jedné součásti.

Abychom mohli provádět optimalizace LAMP platformy, je vhodné si nejprve udělat představu, ve které komponentě se vyskytuje největší prostor pro zlepšení. Přestože metodika testování a výsledky budou popsána až v pozdějších podkapitolách, dovolíme si ukázat jeden graf s výsledkem testu už nyní. A to právě ten, který nám ukáže, ve kterých komponentách se stráví kolik času při zpracování části modelové aplikace za použití jednoduché přehledové metody popsané v další části této práce.



Graf 1: Poměr času stráveného v komponentách

Jak jasně plyne z výsledků testu znázorněných na grafu 1, nejvíce času trvá vykonání dotazů do MySQL a odbavení vygenerované PHP stránky přes síť. Těmto částem tak bude věnována náležitá pozornost.

3.1 Linux (OS)

Graf 1 ukázal velký nárůst času potřebného k vyřízení požadavku přes síť oproti požadavku přes localhost. Toto lze interpretovat tak, že v linuxovém jádru a hlavně jeho síťovém subsystému je velký prostor k provedení optimalizací. Ale začněme od základů, tedy uložení souborů.

3.1.1 Optimalizace filesystému

Problematika výběru nejvhodnějšího souborového systému použitého na discích serveru je poměrně složitá a může značně ovlivňovat výkon při čtení a zápisu souborů. Pro každou aplikaci bude ten nejlepší filesystém jiný, podle toho, jak s ním pracuje. Pro nasazení, kde se provádí převážně čtení velkého počtu malých souborů je vhodný jiný souborový systém než pro aplikaci, která naopak provádí zápisy velkých souborů. Při hledání optimální varianty je dobré vycházet ze seznamu těch nejběžněji používaných filesystémů dostupných ve stabilní verzi pro aktuální linuxový kernel:

- **EXT3** – standardní linuxový FS,
- **EXT4** – nástupce ext3 s novými vlastnostmi,
- **XFS** – vyvinutý společností Silicon Graphics původně pro OS IRIX, je dobrý při práci s velkými soubory,
- **ReiserFS** – FS od společnosti Namesys, je dobrý při práci s malými soubory,
- **JFS** – FS od IBM, původně určený pro AIX.

Po výběru konkrétního souborového systému bychom ještě měli věnovat pozornost jeho nastavení. Jedním z těch, které nás zajímají je velikost bloků. Jeho hodnotu bychom měli zvolit podle velikosti typického souboru nebo můžeme provést zátěžový test a vybrat na základě jeho výsledků. Obecně výchozí hodnota velkému počtu případů plně dostačuje.

Na čem se shoduje mnoho příruček o optimalizaci webových serverů[18] je nastavení parametru *noatime* na oddílu, kde máme webový obsah. Standardně je totiž při každém přístupu k souboru aktualizován *atime* záznam – tedy čas posledního přístupu k danému souboru. Přínost tohoto nastavení ani nestojí za otestování, neboť bude na hraně měřitelnosti. Aplikace vlastnosti noatime v konfiguračním souboru */etc/fstab* by na ukázkovém oddílu /dev/sda2 mohla vypadat zhruba následovně:

/dev/sda2 /test ext3 noatime 1 2

Když už se pohybujeme v oblasti disků, je třeba zmínit i nastavení parametrů disků, což se v linuxovém světě provádí utilitou *hdparm*. Těch program umí nastavit poměrně hodně, z hlediska výkonu bychom si měli ověřit snad pouze to, zda je při přístupu na disk využíváno (U)DMA. Pokud by totiž nebylo, jedná se nejpravděpodobněji o problém s nesprávným ovladačem řadiče disků. To je samozřejmě doprovázeno velkou degradací výkonu a je třeba tento problém vyřešit.

3.1.2 Použití ramdisku

V linuxovém jadře je už velice dlouho podpora tzv. ramdisků, tedy virtuálních disků, jejichž celý obsah je umístěn v operační paměti počítače. Takovýto disk pak má výborné parametry co se týče přistupové doby a rychlosti čtení i zápisu. Operační paměť je dnes poměrně levnou zálažitostí, takže pokud se to někde hodí není problém tohoto využít. Nevýhodou ramdisku je však skutečnost, že pokud dojde k vypnutí počítače, dojde i ke ztrátě veškerých v ramdisku uložených dat. Toto se dá vhodným způsobem ošetřit pro případ korektního vypnutí vytvořením initskriptu, který překopíruje data z ramdisku na pevný disk. Ale v případě např. výpadku napájení jsou data ztracena. Proto je rozumné ho používat pouze na data, která nejsou kritická a můžeme je obnovit ze zálohy.

V případě webových aplikací lze uvažovat o použití ramdisku pro uložení PHP skriptů případně statického obsahu (pokud je jeho velikost relativně malá). Od tohoto lze očekávat zrychlení zpracování skriptů a propustnosti u statického obsahu. Na druhou stranu rozdíl patrně nebude nikterak velký kvůli použití cache pro soubory co jsou na pevných discích. Naopak použití pro umístění datových souborů databáze MySQL určite není dobrý nápad, zde se dají očekávat časté změny a tím roste riziko ztráty dat a problém s jejich trvalým ukládáním na disk.

V linuxu se vyskytuje několik možností, jak takovýto ramdisk vytvořit, v současné době je nejvhodnější implementace tmpfs. Výtvoření ramdisku se provádí buď ručně pomocí utility mount nebo zápisem do /etc/fstab, čímž se zajistí jeho vytvoření i po rebootu. Řádek ohledně ramdisku by v něm mohl vypadat přibližně následovně:

none /var/www/ramdisk tmpfs size=256M,nr_inodes=10M,mode=0777 0 0

Poté už by jen zbývalo zajistit jeho naplnění daty.

3.1.3 Parametry jádra

Naprostá většina návodů se tématu optimalizace parametrů jádra vyhýbá. Bylo tak těžké najít alespoň pár rad jak na to. Jde především o nastavení týkající se síťového subsystému. Zde je aplikace těchto nastavení ukázána ve formě příkazů v shellu, nicméně pro trvalé používání je vhodnější použít aplikaci přes sysctl.

Následující nastavení snižuje práci, kterou musí odvést TCP stack [18] odstraněním režie TCP paketů :

 # echo 0 > /proc/sys/net/ipv4/tcp_sack

- vypíná TCP Selective Acknowledgements

 # echo 0 > /proc/sys/net/ipv4/tcp_timestamps

- vypíná přidávání časových razítek (12B) do hlavičky TCP packetů (dle RFC 1323)

Kromě toho je vhodné provést nastavení zvětšující buffery socketů a TCP[19]:

 # echo 25165824 > /proc/sys/net/core/rmem_max

- nastavuje Maximum TCP Receive Window

 # echo 25165824 > /proc/sys/net/core/rmem_default

- nastavuje Default Receive Window

 # echo "25165824 25165824" > /proc/sys/net/ipv4/tcp_rmem

- nastavuje paměť rezervovanou pro TCP přijímací buffer (pro každé spojení)

 # echo "4096 65536 25165824" > /proc/sys/net/ipv4/tcp_wmem

- nastavuje paměť rezervovanou pro TCP odesílací buffer (pro každé spojení)
- minimumální, výchozí a maximální hodnotu

 # echo 25165824 > /proc/sys/net/core/wmem_max

- Maximum TCP Send Window

 # echo 65536 > /proc/sys/net/core/wmem_default

- nastavuje Default Send Window

Dalším nastavením pak změníme různé časování ohledně spojení:

 # echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout

- umožňuje rychleji uzavírat stará spojení a zůstane tak více volných prostředků pro nová spojení

 # echo 1800 >/proc/sys/net/ipv4/tcp_keepalive_time

- zmenší čas mezii posledním odeslaným packetem a první keepalive výzvou

3.1.4 Verze jádra

Pro provoz našeho serveru bychom měli používat rozumně aktuální verze linuxového jádra. Jádra řady 2.4 jsou již několik let pouze ve stavu občasné údržby kritických chyb, z hlediska podpory novějšího hardware (ve světě serverů velký problém) a výkonu hodně zaostává. Jedinou rozumnou volbou je tedy řada 2.6, která je aktivně vyvíjena a jsou do ní přidávány nové vlastnosti, které mají vliv na výkon celého systému. I v této řadě je nutné používat novější verze kvůli bezpečnostním záplatám, pokud nejsou v používané distribuci linuxu backportovány. Minimálně lze doporučit verzi 2.6.23, do které byla přidána nová implementace plánovače přidělování procesorového času – CFS (Completely Fair Scheduler). Ten by měl spravedlivěji přidělovat procesor a tedy dá se čekat, že přináší menší zlepšení výkonu.

3.1.5 Nepotřebné služby a moduly

Nepotřebné služby jsou udávány jako problém v naprosté většině návodů k zabezpečení serveru. Ani z hlediska výkonu není žádoucí nechávat na serveru běžet nepoužívané služby. V každém případě zbytečně zabírají operační paměť, která se může mnohem lépe využít např. jako cache. Čas od času pak spotřebují i trochu procesorového času, opět zcela zbytečně. Nepotřebné služby na servery prostě nepatří.

Co platí o službách platí i v případě modulů jádra. Pokud je nepoužíváme, neměli bychom je mít natažené v jádře našeho serveru.

3.2 Apache

Na internetu se vyskytuje nepřeberné množství různých tutoriálů zabývajících se problematikou ladění výkonu webového serveru Apache. První místo, kde dostaneme rady ohledně výkonu je přímo na webu Apache [12].

3.2.1 Konfigurační volby

Výše zmíněný návod od autorů Apache zmiňuje seznam problematických voleb a nebo těch, které vyžadují zvláštní pozornost:

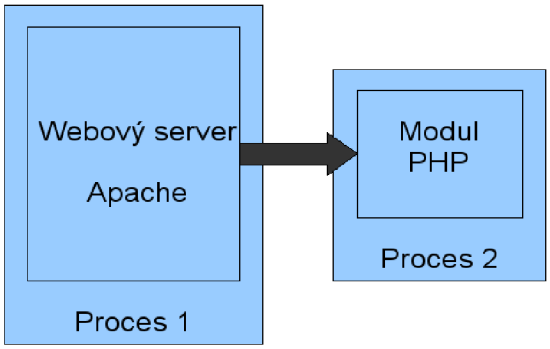
- ***AllowOverride*** – tato volba určuje, která nastavení můžeme ovlivnit vlastním .htaccess souborem. To je jistě užitečná věc, ale zapnutí znamená, že se Apache pokusí otevřít tento soubor v každém jednotlivém adresáři po cestě až k vyžádanému souboru. Pro maximální výkon je vhodné nastavit hodnotu None,
- ***DirectoryIndex*** – určuje pořadí, v jakém se bude hledat výchozí stránka v adresáři (index.html, index.php, index.htm atd.). Je tedy vhodné nastavit je v pořadí, v jakém se nejčastěji vyskytují, aby nemuselo být zbytečně vyhledáváno několik neexistujících výchozích stránek než Apache narazí na první existující. Ty nepoužívané samozřejmě odstraníme,
- ***HostnameLookups*** – asi nejhorší co pro výkon svého serveru můžete udělat je zapnout tuto volbu, které překládá IP adresy na reverzní DNS záznamy pro použití v logu – server musí odeslat dotaz na DNS server, ten se pak postupně dopátrává odpovědi v hierarchii master serverů a i pokud máme štěstí a odpověď je uložena v cache zanáší tato volba nepřijatelné zpoždění. K tomuto účelu existují nástroje, které provedou tento překlad následně po odrotování logu,
- ***EnableMMAP*** – tato volba umožňuje serveru Apache memory-mapping při načítání souborů při zpracování dotazu. Měla by zajistit zlepšení výkonu, takže správná hodnota je On,
- ***EnableSendfile*** – umožňuje využívat podpory sendfile v jádře pro obsluhu statického obsahu – Apache ani nemusí načíst obsah souboru, o to se postará jádro. Je tedy vhodné nastavit na On,
- ***KeepAliveTimeout*** – nastaví, za jak dlouho jsou uzavřeny nepoužívané Keep-alive spojení. Doporučovaný čas 5 vteřin by měl být dostatečný na to, aby se klient případně vzpamatoval a poslal další požadavek. Včasné uzavření nepoužívaných spojení umožní obsloužení dalších čekajících klientů,

- ***MaxSpareServers*** – nastavuje, kolik nejvíce nečinných Apache procesů může být spuštěno. Pokud je jich více, jsou ukončeny. Jejich ukončením se uvolní zbytečně zabraná paměť pro jiné použití,
- ***MinSpareServers*** – opak předchozí volby, pokud je málo nečinných procesů jsou vytvořeny další. Tato vytvořená rezerva umožňuje reagovat na nápory požadavků,
- ***Options*** – specifikuje vlastnosti dostupné v daném adresáři. Zde je z hlediska výkonu problematické, pokud je nastavena hodnota SymLinksIfOwnerMatch, protože pak Apache musí v každém adresáři po cestě k vyžádanému souboru zbytečně spouštět funkci lstat(2). Je tedy vhodné povolit symbolické odkazy v rámci systému a až v adresáři, kde máme umístěny webové stránky nastavit hodnotu -FollowSymLinks +SymLinksIfOwnerMatch,
- ***StartServers*** – nastavuje, kolik procesů server Apache vytvoří ihned po svém nastartování. Je třeba zvolit rozumnou hodnotu vzhledem k velikosti dostupné operační paměti a ostatním důležitým parametrům týkajících se serveru.

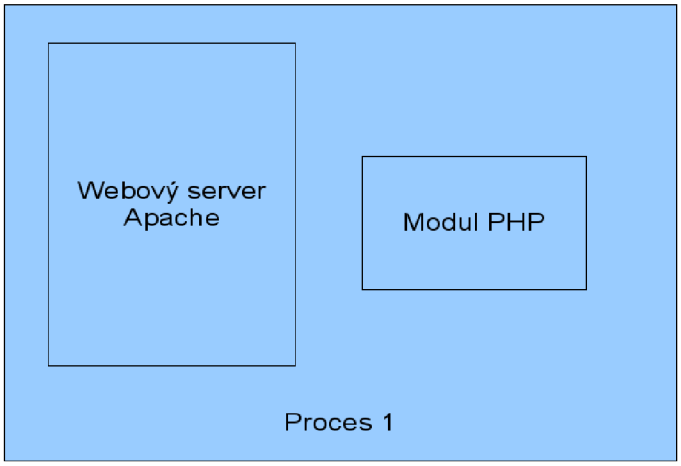
Autoři Apache to sice ve svém návodu neuvádí, ale rozhodně zapněte volbu KeepAlive – umožní klientům poslat skrze jedno navázané spojení více požadavků a ušetří tak režii spojenou s navazováním nového spojení.

3.2.2 Způsob zpracování PHP

Existují různé způsoby, jakými webový server Apache zpracovává PHP skripty[10]. Dva základní způsoby ilustrují následující obrázky 1 a 2.



Obrázek 1: PHP jako CGI



Obrázek 2: PHP jako modul Apache

„Když je PHP zkompilováno jako modul Apache, jak je znázorněno na výše uvedeném obrázku, běží ve stejném adresovém prostoru jako sám proces webového serveru a proto poskytuje podstatně vyšší výkon než tradiční interprety CGI, které se spouští jako samostatné procesy. Některé funkce jako např. Trvalé spojení s databází (...) jsou dostupné pouze pokud je PHP nainstalováno jako modul. Při použití PHP jako interpretu vznikají určité starosti ohledně zabezpečení.“ [2] Vzhledem k tomuto známému faktu o nízkém výkonu PHP spouštěného pomocí CGI tuto možnost nebudeme ani uvažovat. Nicméně ještě existuje třetí možnost zpracování PHP – FastCGI. U ní se dá očekávat přibližně stejný nebo i mírně vyšší výkon v porovnání s PHP jako modul [9]. FastCGI se narozdíl od CGI po zpracování každého požadavku neukončuje. Zpracovává tedy více skriptů na jedno spuštění. Navíc je možné distribuovat FastCGI programy mezi více serverů.

3.2.3 Logování

Apache umožňuje hostovat více webů (domén) na jednom serveru – tzv. virtual hosting. Běžně se používá nastavení, kdy pro každý hostovaný web je udržován zvláštní přístupový (access_log) a chybový (error_log) logovací soubor. Toto nastavení však při větším počtu hostovaných domén znamená, že Apache bude potřebovat mít otevřených několik stovek logovacích souborů. To mu jistě z hlediska výkonu moc nepomůže a navíc to vyčerpává systémové zdroje – file descriptors. Výchozí limit bývá 1024 otevřených soborů pro jeden proces. Proto je třeba zvážit možnost logovat všechny požadavky (a chyby) do centrálního logovacího souboru. Stačí nastavit vhodný formát logu a pro existující parsery logů nebude problém jednou za den (nebo nastavený časový interval) provést rozdělení tohoto velkého logu na logy pro jednotlivé domény. Velký výkonnostní zisk tato změna nejspíše nepřinese, ale i tak stojí za zvážení u serverů obsluhujících velké množství domén.

3.2.4 Reverzní proxy

Klasická proxy funguje jako prostředník mezi klientem (nejčastěji uživatelovým webovým prohlížečem) a webovým serverem někde v internetu (nebo alespoň mimo jeho lokální síť). Proxy mívá dva hlavní účely – umožňuje filtrovat a zrychlovat provoz mimo svou síť a šetřit přenášená data (traffic) kešováním protékajícího obsahu. Pokud si jeden klient vyžádá stránku z internetu, proxy server ji získá (včetně obrázků a dalších vložených objektů), uloží do cache a odešle klientovi. Objeví-li se následně b2hem nastaveného časového limitu druhý klient vyžadující tu stejnou webovou stránku, je ihned obsloužen z cache – načtení stránky tak bylo zrychleno a ušetřil se traffic. Reverzní proxy[16] funguje na stejném principu, pouze je namířena obráceným směrem. Klienti z internetu od ní žádají obsah a pokud ho reverzní proxy má v cache, jsou obslouženi přímo z ní. Není-li tomu tak, je požadovaný dokument nejprve získán z backend serverů, které jsou za reverzní proxy schovány. „Reverzní proxy, která může být velmi malá, načte požadavek od uživatele, a až jej celý načte, předá ho dále na aplikační server. I kdyby to bylo po 100megabitové síti, stráví aplikační server komunikací s uživatelem tisíckrát menší dobu. Pokud navíc generování stránky je netriviální úkol, můžete použít reverzní proxy s cachovaním a váš velký aplikační server nemusí trávit svůj drahocenný čas poskytováním statických stránek.“[13]

V podstatě je možné aplikovat dva rozdílné přístupy. Buď použijeme reverzní proxy pouze na kešování dat a o všechny požadavky se bude i nadále starat Apache, nebo provedeme složitější instalaci a budeme rozdělovat zátěž mezi Apache (necháme ho zpracovat pouze dynamicky generovaný obsah – tedy PHP stránky) a jiný jednoduchý a rychlý webový server, např.: lighttpd [14], thttpd, boa, který se postará o obsluhu statického obsahu. Druhá možnost je z hlediska výkonu tou nejlepší, neboť Apache je sice skvělý, výkonný a všestranný webový server, ale v obsluze statického obsahu nejvýkonnější není a na nejlepšího má poměrně velkou výkonostní ztrátu.

Co se týká samotných reverzních proxy, je existuje několik běžně používaných implementací:

- ***Apache mod_proxy*** – proxy, která je přímo součástí Apache[14],
- ***Squid*** – známá klasická proxy, jde použít i jako reverzní[15],
- ***Varnish*** – výkonný HTTP akcelerátor s flexibilním nastavením pomocí vlastního konfiguračního jazyka,
- ***Nginx*** – jedná se o reverzní proxy kombinovanou s webovým serverem.

Další užitečná věc na reverzních proxy je, že pokud zátěž webového serveru i do budoucna bude růst a přeroste možnosti jednoho fyzického serveru, je u většiny reverzních proxy možné používat je jako load balancery a rozdělovat tak zátěž mezi 2 a více fyzických serverů.

3.2.5 mod_cache

Jedná se o modul pro Apache, který zajišťuje kešování v rámci serveru Apache. Používá se třeba v modulu implementujícím funkcionalitu proxy serveru - mod_proxy. Nás však zajímají moduly mod_disk_cache a mod_mem_cache[17]. Tyto moduly si ukládají a čtou obsah z cache na základě klíčů tvořených z URI. **Mod_disk_cache** si udržuje cache na discích (kde je k dispozici velký prostor pro cache ale je k němu pomalejší přístup), zatímco **mod_mem_cache** má obsah své cache uložený v operační paměti (s tím, že dostupná paměť bývá silně omezena, což vykupuje rychlostí čtení). Základní konfiguraci nastavující kešování v rámci celého filesystému na disk (do adresáře /var/www/cache) a expiraci cache po jedné hodině vypadá následovně:

- CacheEnable disk /
- CacheRoot /var/www/cache
- CacheDefaultExpire 3600
- CacheMinExpire 3600

Pro použití operační paměti stačí první volbu změnit na CacheEnable mem /.

3.2.6 mod_gzip

Mod_gzip je modul Apache, který se stará o kompresi odesílaného obsahu. Server podle HTTP hlavičky zaslané klientem pozná, zda je schopen akceptovat zkomprimovaný obsah. Pokud tomu tak je tak se před odesláním odpovědi tato komprese provede. Na jednu stranu to stojí nějaký procesorový čas navíc, na straně druhé šetří kapacitu přípojné linky – pokud je na hranici své kapacity a procesor nepracuje naplno, může komprese obsahu pomoci obsloužit pár uživatelů navíc (přestože takovýto stav na hraně kapacity je zcela jistě nežádoucí). Taktéž pokud máme server připojený na lince s měřěnými přenosy dat můžeme tímto způsobem něco ušetřit.

3.2.7 Obsluha statického obsahu jiným HTTP serverem

Jak již bylo napsáno v přechozí části o reverzní proxy, oddělení obsluhy statického obsahu mimo Apache je z hlediska výkonu obecně dobrý nápad. Na obsluhu statického obsahu se dá použít mnoho různých jednoduchých HTTP serverů, např.:

- **Boa**,
- **Lighttpd**,
- **Thttpd**.

Tyto servery jsou totiž většinou pro tento účel přímo určeny a měly by zvládnout obsloužit více klientů naráz a odeslat jim data rychleji.

Nasazením jednoduchého serveru však kromě tohoto zvýšení výkonu ještě ušetříme operační paměť, neboť o obsluhu těchto požadavků se nebudou starat nabobtnalé procesy Apache, ale mnohem

menší procesy (případně pouze jeden proces u serverů, které neprovádí fork() pro každý příchozí požadavek). Na testovacím serveru byly naměřeny tyto velikosti procesů:

- Apache – 3,6MB,
- Boa – 0,53MB,
- Lighttpd – 1MB,
- Thttpd – 0,74MB.

Co je u oddělení obsluhy statického obsahu trochu problematické je fakt, že se s jeho využitím musí do jisté míry počítat už při návrhu provozované webové aplikace. Nejde však o nic složitého, pouze je nutné statický obsah ukládat (a následně na něj odkazovat) do jednoho (nebo více) pro tento účel určených adresářů. Ten se pak použije jako document root HTTP serveru určeného pro obsluhu statického obsahu.

Nejlepší možností pak je vytvořit pro tento obsah vlastní subdoménu (např. static.domena.cz) a tu nasmerovat na další (veřejnou) IP adresu patřící našemu webovému serveru, kde bude HTTP server pro statický obsah naslouchat. Nicméně dnešní situace s přidělováním IP adres je v době, kdy se blíží jejich vyčerpání, složitá a proto může nastat i situace, kdy nemáme k dispozici další IP adresu. V tom případě je nutné využít jiný port (např. oblíbený 8080) na stejné IP adrese jako naslouchá Apache. S tímto bychom se mohli spokojit, pokud by šlo o vnitřní síť bez firewallů a proxy, avšak kvůli jejich výskytu na internetu by se část uživatelů k tomuto statickému obsahu vůbec nemusela dostat. Nezbyde nám tedy nic jiného než použít reverzní proxy, která se bude starat o přeposílání požadavků buď na Apache (dynamický obsah) nebo na HTTP server pro statický obsah. Proxy bude používat standardní port 80, Apache budeme muset přesunout například na 8000. Rozlišovat, co kam poslat, se v tomto případě bude podle hlavičky *Host:*. Tím bude tento problém s někdy blokovaným portem vyřešen.

Pokud bychom museli oddělení provést už u hotové webové aplikace, kdy se s tímto nepočítalo, máme ve většině případů stále možnost. V této situaci budeme muset opět nasadit reverzní proxy, ovšem v tomto případě bude rozhodovat co kam přeposlat podle přípony požadovaného souboru. Avšak konfigurace nám zabere více času z důvodu nutnosti provedení zabezpečení (zcela jistě nechceme, aby server pro statický obsah vyzradil zdrojové kódy PHP skriptů).

Existují však i webové aplikace, které neumožní oddělení, protože i odesílání statických souborů je zapouzdřeno PHP skriptem, který kontroluje různé náležitosti (oprávnění, počet stažení atd.).

V každém případě je vhodné statický obsah směrovat na samostatnou subdoménu, kromě úspor výkonu na straně serveru se dá očekávat zrychlené načítání i na straně klienta, neboť standardně webové prohlížeče při načítání dat z více domén použijí více spojení a nemusí se tolik čekat, než se dotáhne předchozí soubor.

3.2.8 Nepoužívané moduly

HTTP server Apache je velice modulární. Existují pro něj spousty modulů přidávající funkčnosti, které mnohdy s vlastním posláním Apache ani nesouvisí. Je vhodné identifikovat moduly, které naše aplikace (respektive konfigurace Apache) vyžaduje ke svému běhu a ostatní vypnout. Ušetřená paměť a minimální zrychlení za to stojí.

3.3 MySQL

Pokud chceme z MySQL (a z databázových serverů obecně) získat maximální výkon, je třeba věnovat maximální pozornost už samotnému návrhu struktury tabulek a následně pak SQL dotazů nad nimi prováděnými. Pokud navrhneme strukturu tabulek nevhodným způsobem nebo zapomeneme vytvořit nad sloupci tabulky potřebné indexy, ani to nejlepší nastavení MySQL serveru nám příliš nepomůže. V případě složitých SQL dotazů alespoň můžeme doufat, že se s tím server nějakým způsbem popere a zoptimalizuje si je.

Možnosti optimalizace výkonu MySQL databázového serveru spočívají především ve správném nastavení velikosti bufferů a keší. V menší míře pak v nastavení míry agresivity při práci se soubory, tedy např. zda se mají změny okamžitě zapsat na disk, nebo stačí za několik vteřin při pravidelném flushi nebo až se to bude hodit (a spolu s tímto roste pravděpodobnost ztráty dat při pádu serveru). Tato nastavení se provádějí v konfiguračním souboru my.cnf. Většina z nich je nastavitelná pouze při spuštění, některé hodnoty však můžeme (pokud máme dostatečné oprávnění) měnit za běhu. Jedná se o tzv. Server System Variables [21].

U MySQL asi nejvíce ze všech součástí LAMP platformy platí, že při optimalizaci je nutné vycházet z konkrétních podmínek na serveru a dle aplikací, které server používají. Jen velice obtížně lze vydávat obecná doporučení. Při jednom použití vyhovují jedny parametry, při jiném úplně odlišné.

Nepoužíváme-li v našich aplikacích InnoDB tabulky, lze jejich podporu vypnout pomocí zadání parametru ***–skip-innodb*** při spuštění MySQL serveru.

„Systém MySQL obsahuje sadu interních vyrovnávacích pamětí (bufferů) a pamětí cache. U těchto lze řídit, kolik paměti se každé z nich přidělí. Dva nejdůležitější parametry z hlediska přidělování paměti jsou *key_buffer_size* a *table_cache*. Tyto parametry sdílí všechna vlákna běžící na serveru a mají na výkon serveru velký dopad. “[20]

3.3.1 Nastavení bufferů

„Klíč vyrovnávací paměti (key buffer) představuje místo v paměti, kam se ukládají indexy tabulek MyISAM. Jakmile dojde k použití bloku indexů, dojde rovněž k jejich uložení do vyrovnávací paměti. Při zadání dotazu se vždy, pokud je příslušný blok indexu ve vyrovnávací paměti načte přímo

z ní. V opačném případě se bude muset blok indexu načíst z disku do klíče, což je samozřejmě pomalejší. Obecně lze tedy říci, že čím je klíč vyrovnávací paměti větší, tím lépe. (...) Jeremy Zawodny, renomovaný expert společnosti Yahoo! na ladění MySQL, doporučuje nastavit tuto hodnotu zhruba na 20% - 50% celkové paměti určené pro server.“[20] Tato proměnná ***key_buffer_size*** se týká pouze MyISAM tabulek, pokud je využíván i typ innoDB je třeba se zajímat o parametr ***innodb_buffer_pool_size***.

„Vyrovnávací paměť pro čtení, jejíž velikost nastaví parametr ***read_buffer_size***, se používá pro ukládání dat tabulky při úplném prohledávání tabulky. Čím více dat tabulky lze uložit, tím méně je potřeba načítat z disku. Pokud však tento parametr nastavíte na příliš velkou hodnotu, sada čtecích vyrovnávacích pamětí může pro každé vlákno spotřebovat obrovské množství paměti.“[20]

„Vyrovnávací paměť pro řazení, nastavená v parametru ***sort_buffer_size***, se používá při provádění dotazů s klauzulemi ORDER BY a slouží tedy k seřazení dat. Parametr nastavte na větší hodnotu, jestliže třídíte velké množiny dat, nicméně i zde hrozí stejná rizika jako u vyrovnávací paměti záznamu. “[20]

Pro dotazy používající JOIN nad tabulkami je pak vhodné nastavit velikost ***join_buffer_size***, což zrychlí provedení těchto dotazů v případě, že nad těmito tabulkami nebylo možné vytvořit správně indexy.

Dále pokud používáme innoDB tabulky, je vhodné nastavit proměnnou ***innodb_additional_mem_pool_size*** určující velikost paměti na data o innoDB tabulkách. Nastavení hodnoty 16MB by mělo být dostatečné pro běžné použití, v případě příliš nízké hodnoty se objeví záznam v logu.

Velikosti Read, sort i join bufferu by ve většině případů mělo stačit zhruba o velikosti 8MB. Naopak *key_buffer_size* a *innodb_buffer_pool_size* bychom s dodržením výše zmíněných procentuálním omezením měli nastavit tak, aby se nám do nich vešly všechny data v otevřených tabulkách (umožňuje-li to velikost instalované operační paměti serveru) a nemuselo se příliš pracovat s pomalým diskem.

3.3.2 Nastavení cache

„Druhým opravdu důležitým parametrem je ***table_cache***. Tento parametr omezuje maximální počet souběžně otevřených tabulek. U tabulek typu MyISAM je každá tabulka a každý index samostatný soubor ve vašem operačním systému. Otevírání a zavírání souborů je pomalé, proto soubory zůstávají otevřené, pokud se výslovně nezavřou, pokud se nezastaví server nebo pokud celkový počet otevřených tabulek nepřekročí parametr table_cache. Zvětšení hodnoty bude užitečné, jestliže váš server obsahuje velký počet tabulek.“[20] Výchozí hodnotou bývá 64, což je i pro běžné použití na serveru s několika webovými aplikacemi velmi nízké číslo, které bude třeba zvýšit. Přizpůsobte ho počtu tabulek, které máte v databázi uloženy a přidejte nějakou rezervu pro případ budoucího růstu.

Kromě keše na tabulky je možné a doporučované zvýšit velikost keš na dotazy – ***query_cache_size***. Do ní se ukládají výsledky SQL dotazů. Pokud se nám tedy SQL dotazy často opakují, určitě zvýšení přinese zlepšení rychlosti odpovědi. Na její velikosti bychom tedy neměli příliš šetřit a místo výchozích 16MB nastavit alespoň 64MB.

3.4 PHP

Většina zde probíraných metod zrychlení běhu PHP skriptů jsou založené na tzv. (byte)**code cache**. „Code cache je termín pro proces, při němž se ukládají kompilované stránky PHP, takže pak PHP nemusí každou stránku znovu kompilovat pokaždé, když se požaduje.“[1] Tato technika nám umožňuje ve většině případů skokové zvýšení výkonu, neboť minimalizuje režii spojenou s kompilováním PHP skriptu a procesor(y) je pak volný pro vlastní vykonávání kódu aplikace. Rozšíření PHP, která využívají této techniky je cela řada. Pro účely této práce byly vybrány ty nejvýznamnější z nich – EAccelerator, APC a XCache. Většinou jediné co u nich lze nastavit je velikost cache pro zkompilované PHP skripty, pokud tedy na serveru provozujeme velké množství aplikaci nastavíme přiměřeně větší než defaultní velikost.

Kromě výše zmíněné skupiny rozšíření vedou ke zvýšení (snížení) rychlosti běhu PHP také některé konfigurační volby, o kterých se zmíní další část kapitoly.

3.4.1 EAccelerator

EAccelerator je PHP akcelerátor, optimalizér a cache dynamického obsahu. Ukládá si PHP skripty ve zkompilované podobě, čímž zrychluje jejich opakované provádění. Není totiž třeba znovu skript kompilovat, stačí ho načíst z cache. Autoři uvádějí, že zrychluje provádění PHP skriptů 1 až 10x. EAccelerator je založen na kódu z projektu Turck MMCache. Nastavení je dílem několika málo konfiguračních voleb, z nichž ty nejdůležitější jsou:

- ***eaccelerator.enable*** - zapínající toto rozšíření (1),
- ***eaccelerator.optimizer*** – zapíná provádění optimalizací kódu,
- ***eaccelerator.shm_max*** – maximální zabraná SHM paměť,
- ***eaccelerator.shm_ttl*** – čas, po kterém mohou být z paměti odstraněny nepoužívaná data,
- ***eaccelerator.shm_only*** – nastavení, zda používat na cache pouze operační paměť.

3.4.2 APC

APC je zkratkou Alternative PHP Cache. Jedná se o opensource produkt, který lze velice snadno zakomponovat jako modul do stávající instalace PHP. Konfiguračních možností není příliš mnoho

[6]. U většiny z nich je vhodné nechat výchozí hodnoty, pro základní nastavení nám bohatě stačí ověřit si, že tyto tři jsou nastaveny na vhodné hodnoty:

- ***apc.enabled*** – pro zapnutí tohoto rozšíření nastavíme hodnotu 1,
- ***apc.optimization*** – určuje, zda se má APC pokoušet provádět optimalizace kódu, hodnota 1 a výše tuto vlastnost zapíná a podle výše hodnoty provádí agresivnější optimalizace,
- ***apc.shm_size*** – udává maximální velikost SHM v MB, která bude použita jako cache – vhodná velikost je 30.

3.4.3 XCache

XCache je rozšíření PHP, které kešuje zkompilovanou podobu PHP skriptů do SHM paměti. Tím zvyšuje jejich výkon, neboť není třeba při každém spuštění znovu a znovu kompilovat – použije se jednoduše zkompilovaná verze z cache. Tímto dociluje až pětinásobného zrychlení generování stránek PHP skripty a snižuje zátěž serveru. Stejně jako v případě APC a EAccelerator je nastavení jednoduché:

- ***xcache.cacher*** – zapíná kešování zkompilovaných skriptů,
- ***xcache.size*** – nastavuje velikost cache,
- ***xcache.ttl*** – nastavuje expiraci obsahu cache,
- ***xcache.optimizer*** – zapíná optimalizátor.

3.4.4 Zend Optimizer

Zend Optimizer je komerční software od společnosti Zend Technologies Inc., nicméně je dostupný k bezplatnému použití. Ve své podstatě jde o aplikaci, která umožňuje spouštět PHP aplikace zakódované programem Zend Guard sloužícím k ochraně komerčních aplikací proti kopírování a zobrazení zdrojového kódu. Běh takto zakódovaných aplikací také do jisté míry zrychluje.

Mimo Zend Optimizeru vyvíjí stejná společnost produkt Zend Platform. Jde o drahý komerční produkt. Jde o webový aplikační server určený k mnohonásobnému zvýšení výkonu PHP aplikací, dle výrobce až 25x[22]. V rámci této práce nebude testován, nicméně mělo by se jednat o funkční možnost optimalizace výkonu PHP aplikace.

3.4.5 Konfigurační volby

Jak již bylo napsáno v úvodu této kapitoly, některé konfigurační volby v hlavím konfiguračním souboru PHP mají vliv na výkon:

- ***register_globals*** – volba ovládající chování proměnných. Pokud je zapnutá, všechny proměnné získané od klienta jsou dostupné v našem kódu, což může vést k problémům,

pokud si používané proměnné neinicializujeme. Navíc tato funkce vyžaduje jistý výkon - měla být vypnutá, pokud ji opravdu v naší aplikaci nepotřebujeme.

- ***expose_php*** – volba přidávající do odpovědi hlavičku oznamující, že je použito PHP. Opět je vhodné vypnout.
- ***magic_quotes_**** - skupina voleb, které zapínají funkci automatického escapování řetězců v různých případech. Vypnutím těchto funkcí je možné získat nějaký výkon navíc, je však nutné si být jist, zda jsou v naší aplikaci na všech potřebných místech řetězce escapovány abychom si nezpůsobili víc problémů než užitku.
- ***register_argc_argv*** – funkce řešící, zda budou v aplikaci přístupné proměnné argc a argv obsahující počet vstupních argumentů a vlastní argumenty. Vzhledem k tomu, že zcela jistě nebudeme provozovat PHP jako CGI můžeme tuto volbu vypnout.

3.4.6 Nepoužívaná rozšíření

Stejně jako u jiných součástí LAMP platformy i u PHP můžeme mít nainstalované a hlavně aktivovaná nepoužívaná rozšíření, neboť v linuxových distribucích bývá jako minimální závislosti uvedeny i moduly, které téměř nikdo nepoužívá a přijejich instalaci jsou samozřejmě aktivovány. Může se jednat o GD pro manipulaci s obrázky, mcrypt pro šifrování nebo o rozšíření pro připojení k PostgreSQL databázi. Vylepšení výkonu bude nejspíše neměřitelné, ale zbytečně zabírají paměť, takže je vypneme zakomentováním řádku např.:

extension=gd.so

nebo

extension=mcrypt.so

4 Test optimalizací

V předchozí kapitole jsme si přiblížili množství možností, které by měly vést ke zvýšení výkonu platformy LAMP. Nicméně zcela nepochybně není přínos všech popsaných úprav jednotlivých součástí platformy stejný a některé nemusejí dokonce vůbec fungovat a mohou situaci paradoxně ještě zhoršovat. Proto je potřeba provést testy různých konfigurací a zjistit, které stojí za to implementovat a které nepřinášejí žádné výhody. Celá tato kapitola je věnována popisu testovacího prostředí, testovacích metod, modelové aplikace a výsledkům provedených testů.

4.1 Testovací prostředí

Začněme pěkně od začátku – každý test vyžaduje prostředí, ve kterém bude vykonán a metodiku, podle které se bude jeho provádění řídit. V této podkapitole se budeme věnovat použitému software, hardware a použitým testovacím metodám.

4.1.1 Software

Ze softwarového hlediska bylo pro testy zvoleno prostředí linuxové distribuce Debian. Konkrétně verze 4.0 (kódové jméno Etch) včetně všech toho času dostupných aktualizací. V průběhu testování různých optimalizačních metod se objevila potřeba nainstalovat některé části systému z novější (dnes už také vydané jako stabilní) verze s kódovým názvem Lenny. V konečném důsledku byly na serveru použity následující verze důležitých součástí systému:

- jádro Linux 2.6.26,
- HTTP server Apache 2.2.9,
- PHP 5.2.6,
- MySQL server 5.0.51a,
- ApacheBench 2.3,
- Siege 2.66,

V průběhu testování pak byly prováděny testy na následujících verzích programů:

- EAccelerator 0.9.5.3,
- APC 3.1.2,
- XCache 1.2.2,
- Zend Optimizer 3.3,
- Lighttpd 1.4.19,
- Boa 0.94,
- Thttpd 2.25b,

- Nginx 0.6.32,
- Varnish 1.1.2,
- Squid Cache 2.7.STABLE3.

4.1.2 Hardware

Pro všechny popisované testy byl využit jako server počítač o následující konfiguraci hardware:

- dvoujádrový procesor AMD Athlon X2 5200+,
- 2GB DDR2 RAM paměti,
- 250GB SATA pevný disk Western Digital,
- onboard 1Gbit síťová karta Realtek.

Testovací programy byly spouštěny na notebooku Dell Latitude D505 o konfiguraci:

- procesor Intel Celeron M 1.3GHz,
- 512MB DDR paměti,
- 40GB ATA pevný disk IBM,
- 1Gbit síťová karta Intel.

Mezi danými počítači byla vytvořena 1Gbit/s Gigabit Ethernet počítačová sít pomocí kříženého UTP kabelu.

4.1.3 Metodika testování

Testování je zásadní pro celou práci a tak si v této kapitolce popíšeme, jaké testovací programy a postupy budeme používat k testování výkonu. A vlastně i dle toho si stanovíme, co vlastně budeme považovat za výkon.

Vzhledem ke snaze o maximalizaci vypovídací hodnoty námi provedených testů se budeme držet rad uvedených v [7] vytvořením co nejpodobnějších a nejspravedlivějších možných podmínek k testu:

- Samozřejmě použijeme pro všechny testy stejnou konfiguraci hardwaru a na straně počítače, kde jsou spouštěny testovací programy i stejné verze software včetně nastavení,
- použijeme stejně postavené počítačové sítě pro všechny testy,
- každý jednotlivý test provedeme 5krát a jako výsledek použijeme nejlepší dosaženou hodnotu,
- po každém provedeném testu provedeme reboot testovacího serveru, aby se vyprázdnily cache,

4.1.3.1 Výkon HTTP serveru

Z hlediska výkonu HTTP serveru se běžně vyskytují tři základní parametry [8]:

- Počet obsloužených požadavků za vteřinu,

- Latence odpovědi,
- Propustnost.

Počet obsloužených požadavků za vteřinu je poměrně jasné měřítko. Do jisté míry udává, kolik uživatelů naší aplikace zvládne server naráz obsloužit. Ovšem vždy záleží na vhodně nastaveném testu, který by se měl co nejvíce přiblížit skutečnému chování uživatelů (respektive jimi používaných webových prohlížečů). Počet obsloužených požadavků za vteřinu se velice liší podle povahy požadavku. Statických HTML souborů nebo obrázků zvládne většina serverů obsloužit několik stovek až tisíců za vteřinu. Pokud však dojde na složitější obsah, který se dynamicky mění a generuje až při příchodu požadavku od klienta k serveru, počet požadavků obsloužených serverem za vteřinu výrazně poklesne na řádově desítky až stovky. Pokud jde o velice náročný dynamický obsah generovaný použitím mnoha SQL dotazů a provádějících náročné výpočty, klidně se dostaneme na hodnoty v řádu jednotek dotazů / s.

Pokud jde o latenci odpovědi, tak v tomto případě jde o několik hodnot, které se dají sledovat. Počínaje dobou, než HTTP server přijme příchozí spojení, pokračuje dobou, než obdrží první bajt odpovědi a konče úplným vyřízením požadavku a ukončením spojení. Zde nám nejde ani tak o konkrétní hodnoty, jako spíše o průměrné a extrémní hodnoty. Toto je především důležité proto, že náš server může být schopný obsloužit stovky dynamicky generovaných stránek za vteřinu, ale pokud bude průměrný čas na vyřízení třeba 5 nebo 10 vteřin (nehledě pak na maxima, která budou dosahovat 20s), tak návštěvníci daných stránek z toho moc nadšeni nebudou. Nicméně v této práci nás tato hodnota nebude až tak zajímat, pouze budeme sledovat, zda při některé použité metodě optimalizace nedosahuje nepřijatelně vysokých hodnot ukazujících na nějaký problém.

Poslední veličinou je propustnost serveru. Tímto máme na mysli počet bajtů odeslaných za vteřinu. Ideální stav je samozřejmě takový, že server dokáže zcela vytížit propustnost svých síťových karet 100Mbit/s (FastEthernet) či 1Gbit/s (GigabitEthernet). To zajisté není až tak velký problém, pokud jde o odesílání pouze statického obsahu v podobě pár velkých souborů, které se dostanou do cache. Pokud však dojde na menší statické soubory ve velkém množství, už bude propustnost klesat. A u dynamicky generovaného obsahu bude tato propustnost ještě menší. Ve skutečnosti však reálné zatížení HTTP serverů není zcela přesně jeden z těchto typů, ale jde o jejich kombinaci. Takže i propustnost bude ležet někde mezi.

4.1.3.2 Jednoduchá přehledová metoda

Tato metoda slouží pouze pro účely ukázání, zda se v některé ze součástí platformy LAMP stráví při běhu modelové webové aplikace více času než v jiných a určení, o které součásti se jedná. Původně zde byla popsána metoda složitější, u které bylo nutné provádět některé kroky ručně. Nakonec se však ukázalo, že její výsledky jsou téměř nepoužitelné a zavádějící. Proto bylo nutné provést modifikace této metody.

Pro provedení testu za využití této metody je nutné provést menší změny v kódu testované modelové webové aplikace. Jedná se o obklopení všech volání pracujících s MySQL databází kódem, který si zaznamená čas před a po skončení funkcí pracujících s databází. Následně pak z jejich rozdílu zjistí, kolik času se strávilo přístupy do databáze. Tím získáme první hodnotu. Už neupravená modelová aplikace obsahuje podobný kód na měření doby běhu celého PHP skriptu. Od takto získané hodnoty odečteme čas strávený prací s MySQL získaný jako první hodnotu. Takto jsme získali dvě časové hodnoty. Další získáme, pokud si vyžádáme daný PHP skript přes Apache z localhostu. K tomuto účelu se ukázal být velice vhodný testovací program ApacheBench nastavený na provedení jednineho požadavku s jediným konkurenčním vláknem. Opět od takto získané hodnoty odečteme minulé dvě získané, abychom dostali pouze čas strávený navíc oproti samotnému běhu PHP skriptu. Nakonec získáme poslední hodnotu stejným způsobem jako tu třetí, pouze místo lokálního počítače provedeme stažení PHP skriptem vygenerované stránky přes počítačovou síť. Od této hodnoty odečteme latenci použité sítě (tu zjistíme jako round trip time pomocí příkazu ping z testovacího počítače na testovací server), aby nám příliš nezkreslila výsledky. Takto získaný čas použijeme jako základ (100%) pro všechny následné výpočty. Od tohoto základu ještě odečteme součet prvních tří hodnot a získáme tak čas použitý síťovým subsystémem navíc oproti předchozímu vyřízení požadavku přes localhost. Celý tento postup provedeme alespoň 5x a hodnoty zprůměrujetme pro získání jisté vypovídací hodnoty údaje. Graf z takto získaných hodnot (Graf 1) je umístěn v úvodu kapitoly o optimalizacích.

4.1.3.3 Testování optimalizací MySQL

Pro potřeby otestování výkonu MySQL databáze bylo třeba napsat jednoduchý skript, který půjde použít pro zjištění, zda provedená úprava v nastavení databáze pomohla nebo ne. Ze zdrojového kódu modelové aplikace bylo nutné vybrat všechny používané SQL dotazy. Ty byly vloženy do kódu testovacího skriptu a ve smyčce mnohokrát opakovány pro dostatečně přesné zjištění doby jejich zpracování databází.

4.1.3.4 Programy pro testování výkonu HTTP serverů

Pro účely testování výkonu webových serverů bylo během let vytvořeno mnoho programů[8], ne všechny jsou však vhodné na testování, které chceme provést v této práci. Naším hlavním požadavkem je, aby takový program umožňoval definovat počet současných připojení a byl po skončení testu schopen vypsat přehlednou statistiku dosažených výsledků. Podle mnoha různých zdrojů [1] se jako vhodný a široce používaný kandidát ukázal být program ApacheBench. Pochází stejně jako samotný Apache HTTP server z dílny Apache Software Foundation, která ho vytvořila právě pro testování svého serveru. Umožňuje jak nastavení počtu konkurenčních vláken tak umí i zobrazit relativně podrobnou statistiku všech důležitých hodnot naměřených během proběhlého testu.

Bohužel v průběhu testování se ukázalo, že ApacheBench má jeden nedostatek a to, že během jednoho testu posílá požadavky stále na jednu a tu samou URL adresu. Toto je v případě testů zaměřených na statický obsah nežádoucí. Tento problém řeší použití jiného benchmarkového programu – Siege.

ApacheBench

Předešlá metoda byla pouze základní a přehledová. Potřebujeme však metodu, která bude dávat přesná čísla. A zde přichází na řadu program ApacheBench zmíněný v úvodu této podkapitoly. Aby měl test nějakou vypovídací hodnotu, provedeme sadu testů s různým nastavením.

Vlastní program ApacheBench umožňuje nastavení velkého počtu různých parametrů. Pro náš test jsou však důležité především tyto:

- **-c** – nastavující počet konkurenčních vláken Apache Bench, které zasílají testovanému serveru požadavky,
- **-t** – určuje čas, po který bude test trvat,
- **-n** – nastavuje kolik požadavků bude během testu uskutečněno,

Pro potřeby této práce bylo zvoleno pro každou testovanou konfiguraci provést sadu 5 testů s následujícími parametry:

- doba trvání 60s, 1 konkurenční vlákno,
- doba trvání 60s, 5 konkurenčních vláken,
- doba trvání 60s, 10 konkurenčních vláken,
- doba trvání 60s, 25 konkurenčních vláken,
- doba trvání 60s, 50 konkurenčních vláken,

Pro první zmíněný test z tohoto seznamu tedy vypadají kompletní parametry, se kterými je ApacheBench spouštěn následovně: „**ab -c 50 -t 60 http://10.11.1.2/skript.php**“.

Jak se později ukázalo v průběhu testování, zpracování některých PHP skriptů trvalo poměrně dlouho a během 60 vteřin nedošlo k uskutečnění subjektivně dostatečného počtu požadavků. Proto v těchto případech došlo k úpravě parametrů a místo 60s se test prováděl s nastavením aby uskutečnil 5000 požadavků, což zvýšilo vypovídací hodnotu testů.

Siege

Pro potřeby testování výkonu na statickém obsahu bylo nutné použít testovacího programu Siege. Ten umožňuje specifikovat seznam URL, ze kterého náhodně vybírá ty, které si v tomto spojení vyžádá. Stejně jako u předchozí metody založené na testování programem ApacheBench i zde použijeme sadu několika testů s různým nastavením. Nicméně jejich počet redukujeme na tři:

- doba trvání 60s, 1 konkurenční vlákno,
- doba trvání 60s, 2 konkurenční vlákna,
- doba trvání 60s, 3 konkurenční vlákena,
- doba trvání 60s, 4 konkurenční vlákena,
- doba trvání 60s, 5 konkurenčních vláken,
- doba trvání 60s, 10 konkurenčních vláken,
- doba trvání 60s, 25 konkurenčních vláken,
- doba trvání 60s, 50 konkurenčních vláken,
- doba trvání 60s, 100 konkurenčních vláken,
- doba trvání 60s, 150 konkurenčních vláken.

Vzhledem k dosahované propustnosti u statického obsahu zastoupeného sadou fotografií, který se vešel do cache (bez větších potíží vytíží i gigabitovou síť) provedeme test na stejném počítači, kde běží testovaný Apache (či jiný použitý HTTP server nebo reverzní proxy). Toto do jisté míry zkreslí výsledky (testovací program generuje zátěž), ale použití síťového prostředí by bylo ještě horší. Naopak u větších soubroSiege budeme spouštět s následujícími parametry: „**siege -c 100 -i -f ./urls.txt -b -t60S**“, kde v souboru urls.txt je seznam testovaného statického obsahu (viz následující kapitola).

4.2 Modelová aplikace

Modelová aplikace byla napsána v PHP a je rozdělena do několika částí. První částí je skript, který generuje obsah databáze, který se později použije pro testování v ostatních částech aplikace. Ty provádějí různé čtení obsahu z databáze. Tento důraz kladený na čtení z databáze je zcela záměrný - pokud se totiž nad tím zamyslíme, tak naprostá většina webových aplikací provádí v databázi především čtení (zobrazení článku, komentáře apod.), zatímco počet zápisů je velmi nízký (redaktor přidá článek jednou za den nebo i týden, komentáře k článkům napíše jeden ze několika stovek návštěvníků). Celá modelová aplikace byla koncipována jako jednoduchá. Z hlediska měření výkonu by bylo zbytečné se zabývat návrhem líbivého designu stránek a zasazování dat získaných z databáze do něj.

Abychom naší modelovou aplikaci co nejvíce přiblížili reálným webovým aplikacím, bylo nutné kromě dynamicky generovaného obsahu použít také obsah statický. V našem případě půjde o sadu 60 (30+30) fotografií ve formátu JPEG ve 2 standardních rozlišeních s průměrnou velikostí cca 323KB. Tyto budou představovat náhledy a fotografie jako je tomu u zpravodajských serverů. Kromě těchto fotografií bylo přidána i velká sada dalších, náhodně vygenerovaných, souborů o velikostech od 1 do 25MB, které zastupují soubory určené ke stažení (např. zdrojové kódy nebo instalační balíčky). Jejich celková velikost převyšuje velikost operační paměti a tak nebude možné je všechny vměstnat do cache. Právě na tomto statickém obsahu (fotky) by se měly ukázat výsledky optimalizace

cache Apache nebo jiných HTTP serverů či kešujících reverzních proxy, které použijeme na jeho místo. Kromě toho by se měly projevit úpravy provedené na úrovni OS. U sady náhodně generovaných souborů se projeví spíše celková (ne)náročnost serverů.

Pro uložení dat aplikace využívá databázový server MySQL. Jako typ použité tabulky byl zvolen typ InnoDB, neboť se jedná o modernější typ s mnoha vlastnostmi navíc oproti výchozímu MyISAM. InnoDB sice má být v některých ohledech pomalejší, ale pro účely této modelové aplikace to není relevantní.

4.2.1 Generování obsahu databáze

O generování obsahu databáze se stará PHP skript nazvaný ***fillDatabase.php***. Data jsou rozdělena do dvou tabulek. První obsahuje články (tabulka clanky) a druhá jejich autory (tabulka autori). Vygenerovaná data (články) uložená v databázi mají představovat neuspořádanou stromovou strukturu. Každý článek může mít až dva přímé následníky – v databázi článků na ně odkazují přes ID článku pole left a right. Každý článek pak má logicky jednoho přímého předka, kromě kořene stromu, který nemá žádného předka.

V prvním kroku skript vygeneruje záznamy pro 50 autorů. V dalším kroku jsou pak postupně vygenerovány jednotlivé články. Jako zdroj textu jsou používány slova z textu Lorem Ipsum. Generovací skript vždy je náhodně vybere přiměřený počet slov pro konkrétní pole v databázi. Pro titulek jde o 2 až 10 slov, u souhrnu 25 až 50 a u vlastního textu článku pak 500 až 1500 slov. Náhodně je také vybrán jeden z 50 dříve vygenerovaných autorů a je mu přiřazeno autorství daného článek. Podobně je vybrán i jeho přímý předek a jako čas vydání článku je použito aktuální datum a čas. Takto je postupně vygenerováno celkem 10000 článků.

Tento skript fillDatabese.php není v této práci předmětem žádného testování, jeho účelem je čistě vytvořit testovací data.

4.2.2 Čtení z databáze

Do databáze jsme si nechali PHP skriptem fillDatabase.php vygenerovat záznamy se články a jejich autory. Nyní se dostane ke slovu druhá část aplikace, která s nimi bude pracovat – číst je. Tato část aplikace umístěná ve skriptu ***showArticle.php*** je interně dále rozdělena podle konkrétně prováděné funkce:

1. První možná funkce provede jednoduše načtení posledních 50 titulků a souhrnů článků, dat publikování a jméno autora. Toto by mělo prověřit cachování výsledků dotazů v MySQL, neboť těchto 50 článků je vždy v průběhu testu naprosto stejných.
2. Druhou možností je výběr 5 náhodných článků a jejich zobrazení. Výběr jejich ID je řešen na straně PHP, neboť výběr provedený v SQL na MySQL byl tragicky pomalý. U

těchto výsledků se nedá příliš předpokládat kešování, protože se pokaždé ptáme na jiné články.

3. Třetí možností (ta je vlastně prováděna vždy) je prosté zobrazení počtu článků a autorů v databázi – opět by mělo být možné obsloužit z cache.

4. Doposud jsme na data v databázi nahlíželi jako na plochou tabulku. Jenže jak bylo vysvětleno v sekci o plnění databáze, má ta naše strukturu stromu. Proto abychom využili nabízených možností je poslední možností načítání práce s články jako se stromem. Jedná se o výběr podstromu – vybereme z databáze potomky vybraného článku do nastavené hloubky zanoření (2). Toto je provedeno rekurzí, aby byla zátěž jak na straně PHP tak i MySQL databáze.

Předmětem testování budou funkce uvedené v bodech 1, 2 a 4, které vlastně všechny zahrnují i použití zmíněné v bodu 3.

4.2.3 Ostatní části

Kromě výše zmíněných fillDatabase.php a showArticle.php tvoří modelovou aplikaci také index.php a lib.php. Index.php tvoří rozcestí mezi jednotlivými součástmi aplikace. V lib.php se vyskytují funkce pro připojení k databázovému serveru MySQL, samozřejmě včetně nastavení všech potřebných přístupových údajů. Lib.php je použit pomocí funkce include() v každého dalším skriptu aplikce.

Při testování bude využit skript index.php, jako typický příklad PHP skriptu, který vlastně téměř nic neprovádí a nepoužívá databázi, pouze je na něj spuštěno PHP.

4.3 Dosažené výsledky

V kapitole 3 bylo popsáno velké množství úprav, které lze provést za účelem zvýšení výkonu. V kapitole 4 je pak popsána modelová webová aplikace, testovací prostředí a metody testování. Na tomto základě bylo provedeno mnoho testů za účelem zhodnocení výkonostního přínosu jednotlivých úprav. Byly otestovány pouze ty úpravy, které jsou vzhledem k modelové aplikaci relevantní a dal se očekávat jejich měřitelný přínos.

V rámci této kapitoly jsou dosažené výsledky prezentovány ve formě grafů, v případě zájmu jsou v příloze k dispozici tabulky s naměřenými údaji.

4.3.1 Jednoduchá přehledová metoda

Jak již bylo uvedeno dříve, touto metodou byl na modelové aplikaci bez provedení jakýchkoliv optimalizací proveden test k získání přehledu, kde se stráví nejvíce času a výsledek byl znázorněn na grafu číslo 1. Jeho výsledky zachycuje následující tabulka:

Část	Naměřený čas	Strávený čas	Podíl na celkovém čase
MySQL	0,86	0,86	26,88%
PHP	1,54	0,68	21,25%
Apache	1,9	0,36	11,25%
Apache přes síť	3,2	1,3	40,63%

Tabulka 1: Výsledek jednoduchého přehledového testu (čas je uváděn v ms)

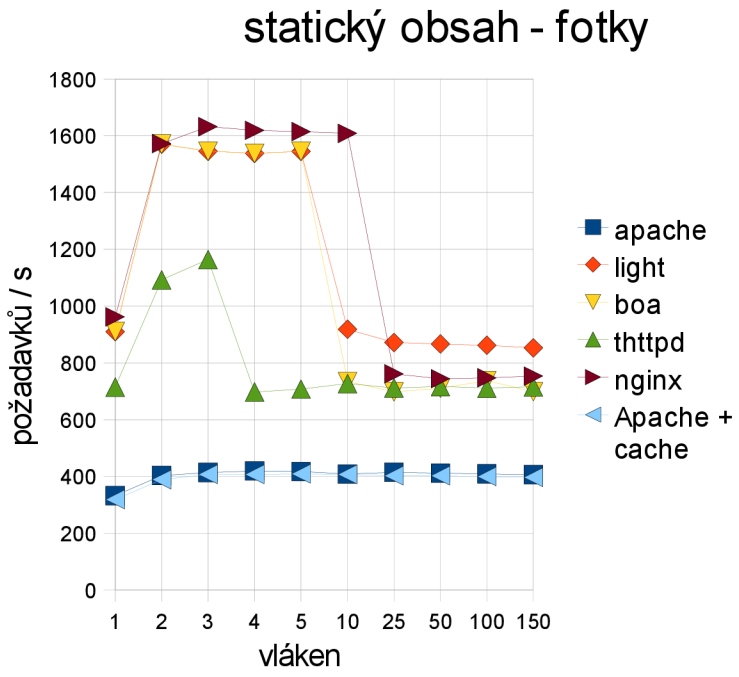
Největší podíl na čase zpracování tedy má odeslání odpovědi přes síť a vyhledání dat v MySQL databázi. Samotné zpracování Apachem je poměrně rychlé, o něco větší podíl pak má čas zpracování PHP skriptu..

4.3.2 Statický obsah

Pro testování bylo použito programu Siege. Došlo k otestování výkonu HTTP serverů a kešujících reverzních proxy. Jak bylo naznačeno v kapitole věnované metodám testování, byly prováděny 2 testy na různém statickém obsahu.

4.3.2.1 Fotky

V tomto případě tvoří statický obsah 60 fotek.

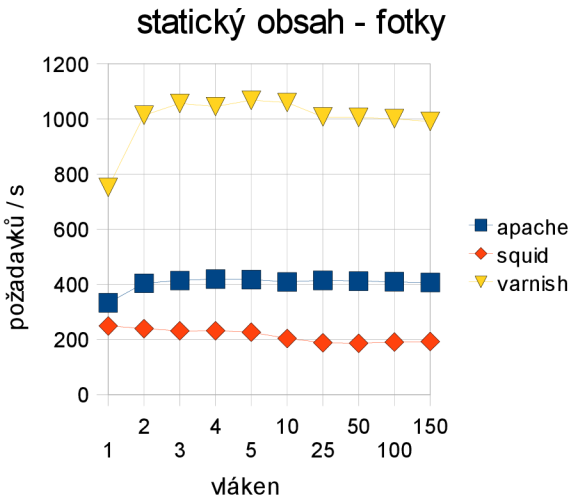


Graf 2: Výkon HTTP serverů na statickém obsahu (fotkách)

Graf 2 ukazuje, že použití jiného HTTP serveru může přinést výkonnostní zisk. V případě serveru nginx dosahuje počet obsloužených požadavků za vteřinu v průměru 2.97krát tolik, co zvládne obsloužit Apache, u lighttpd pak jenom o trochu méně, 2,84krát dosažený výkon Apache. Nasazení modulu mod_cache u Apache naopak způsobilo mírné zhoršení, pravděpodobně zapříčiněné kešováním na příliš mnoha místech (disková cache, cache Apache, mod_cache) a s tím spojenou vyšší výpočetní náročností.

Výsledky ostatních testovaných HTTP serverů nejsou také špatné, zvládnou obsloužit minimálně dvojnásobek toho co Apache.

Co se týče datové propustnosti serverů, tak výsledky zhruba kopírují trend zobrazený na grafu počtu požadavků za vteřinu a tak nepovažujeme za nutné zde graf s touto hodnotou zobrazovat. Nicméně je nutné poznamenat, že tuto sérii testů bylo nutné provést na stejném počítači jako běžely HTTP servery a to z důvodu příliš vysoké dosahované propustnosti, která překonala možnosti gigabitového ethernetu, který byl k dispozici v testovacích počítačích.



Graf 3: Apache vs kešující reverzní proxy - statický obsah (fotky)

Při testu reverzních proxy bylo prostředí sestaveno tak, že na portu 80 poslouchala reverzní proxy a na portu 8080 pak webový server Apache, na který měly proxy směřovat dotazy, které ještě neměly v keši. Před provedením testů kešujících reverzních proxy byly předpoklady, že jejich nasazením bude dosaženo velkého zrychlení. Jak ukazuje graf 3 zachycující výsledky tak částečně se to potvrdilo.

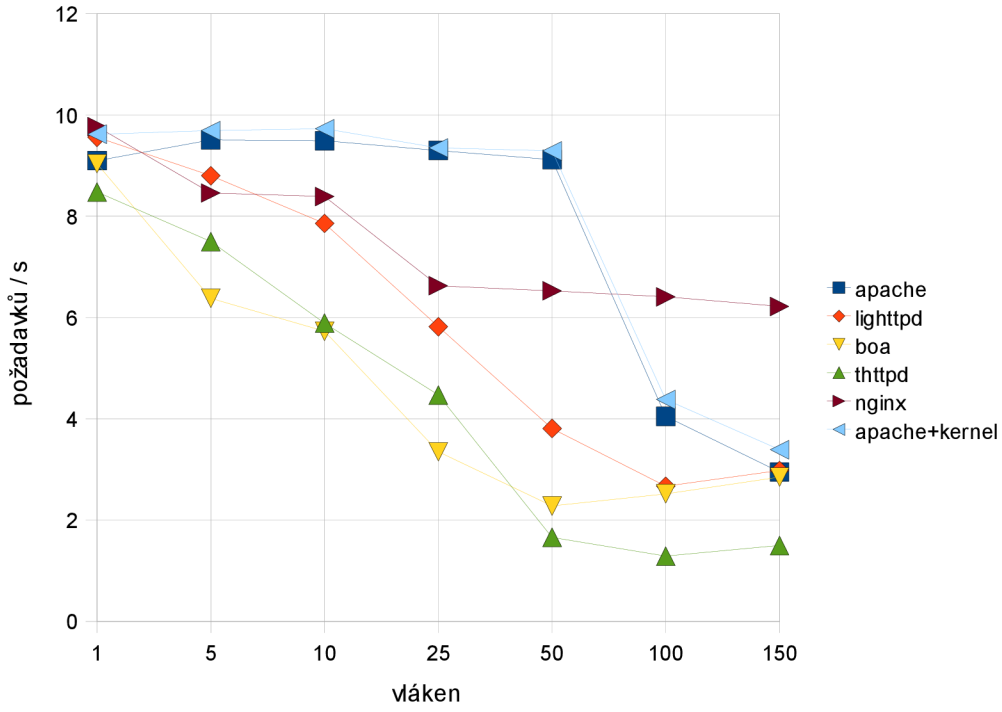
Kešující reverzní proxy Varnish dosáhla oproti serveru Apache téměř 2,5 násobného výkonu. Naopak pro proxy server Squid byly výsledky velkou katastrofou, dosahoval pouze okolo 53 % výkonu Apache! Nejdříve to vypadalo na chybu v konfiguraci, ale po mnoha opakováních testu a

zkoumání logu se ukázal být problém opravdu v samotném Squidu. Ten hlásil cache hit a u Apache se v logu žádné nové požadavky neobjevovaly.

Co se týče propustnosti tak ta i v tomto případě kopírovala průběh grafu 3.

4.3.2.2 Velké soubory

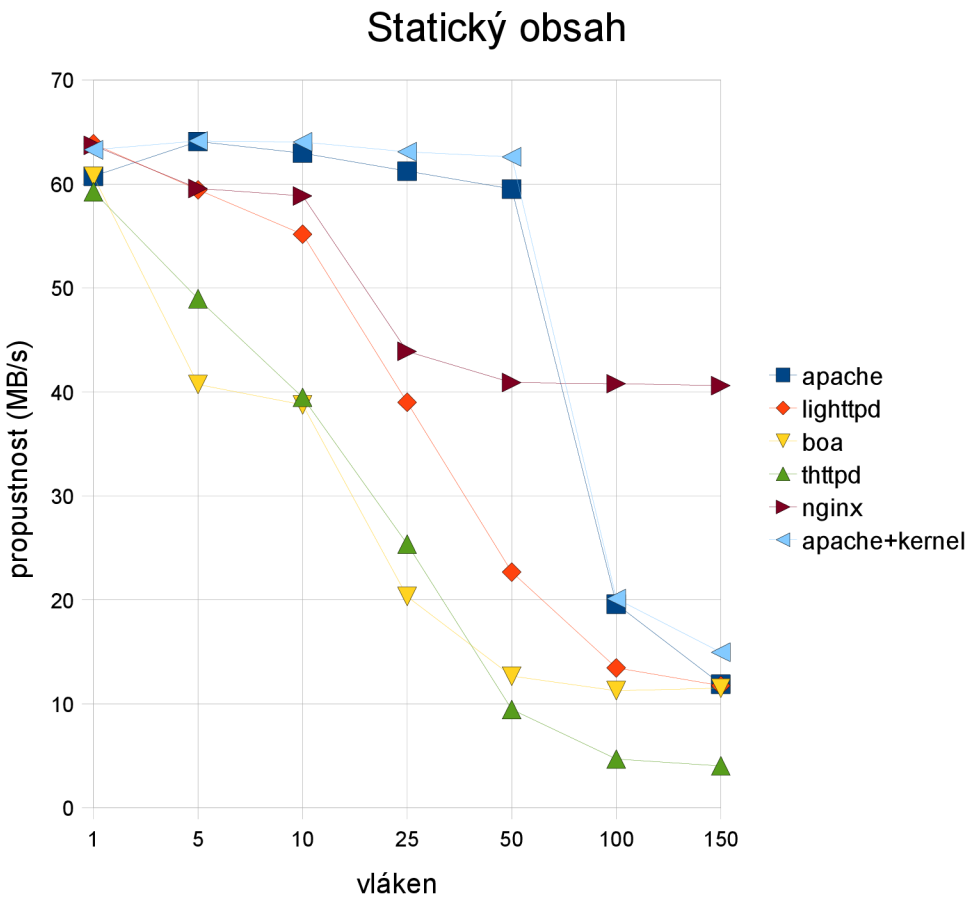
Minulý test provedený na malých fotkách ukázal některé zajímavé výsledky, nicméně pro úplnost jsme považovali za nutné provést další test, kdy se vygenerovalo přes 5GB souborů s náhodným obsahem o velikostech 25MB (60x), 10MB (250x), 5MB (250x) a 1MB (250x). V průměru tedy byla velikost testovacího souboru okolo 6,5MB. Jejich velikost tedy už převýšila velikost operační paměti a logicky nemohlo docházet k obsluze požadavků daty z cache (alespoň u větší části z nich) a tedy se dalo předpokládat, že počet obsloužených požadavků za vteřinu i počet odeslaných dat bude řádově menší.



Graf 4: Výkon HTTP serverů na statickém obsahu (velkých souborech)

Již v průběhu testu se ukázalo, že tento předpoklad je správný a vzhledem k dosahovaným rychlostem menším než datová propustnost gigabitového ethernetu bylo možné provést tento přes síť, čímž se odstranil další problém předešlého testu, kdy velký počet vláken testovacího programu mohl zkreslovat výsledky. Během testu bylo navíc zřetelně slyšet intenzivní využívání pevného disku. Výsledky takto změněného testu to zcela potvrzují, obě sledované hodnoty byly významně menší. Naměřené hodnoty ukazují, že pokud se příliš nevyužívá cache, nejsou rozdíly mezi jednotlivými

HTTP servery až tak velké - všechny je totiž brzdí pomalé disky. Nicméně se z nich dá vyčíst, že do zhruba 75 současných spojení si vede nejlépe právě Apache, v tomto bodě ho však předežene HTTP server nginx, u něhož dochází ze všech testovaných serverů k nejpomalejšímu poklesu datové propustnosti s přibývajícím počtem spojení. V průměru si vedl nginx o 14% lépe než Apache, jinak byly všechny servery horší v rozmezí 20 – 45%. Kromě jiných HTTP serverů bylo na tomto testu demonstrován přínos úprav parametrů jádra zmíněných v kapitole věnovaných optimalizaci jádra. V tomto případě byl Apache po provedení úprav schopen obsloužit v průměru o 5% více požadavků. Tedy pokud jsme v této situaci laděním software toho moc nezmůžeme a investice do rychlejšího diskového pole a více operační paměti je jedinou rozumnou možností jak výkon serveru znatelněji zvýšit.



Graf 5: Výkon HTTP serverů na statickém obsahu (velkých souborech)

Ano zde se u datové propustnotnosti serverů nekonalo žádné překvapení a trend je stejný jako u počtu požadavků za vteřinu. Nicméně závěr testu, kde všechny servery kromě nginx při 150 konkurenčních spojeních dodávají data rychlostí pouze do 15MB/s, stojí za zmínku výkon nginx. I při

tomto počtu klientů je schopen dodávat data pouze o třetinu pomaleji než při jednom klientovi, čímž ostatní několikanásobně převyšuje.

4.3.3 Dynamický obsah

V případě možností ke zlepšení výkonu při obsluze dynamického obsahu bylo provedeno mnoho testů a proto je tako podkapitola dále členěna podle toho, které části se úprava týkala.

Testy byly prováděny na celkem 4 různých skriptech, každý z nich prováděl trochu jinou a jinak náročnou činnost. Budeme je pro jednoduchost označovat 1 – 4, s tím, že:

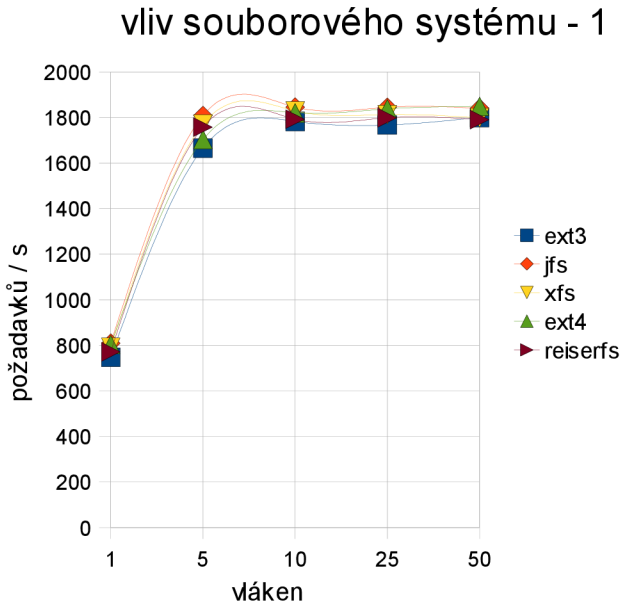
1. je test skriptu index.php,
2. showArticle.php?show=1
3. showArticle.php?id=1
4. showArticle.php?tree=1

4.3.3.1 Linux

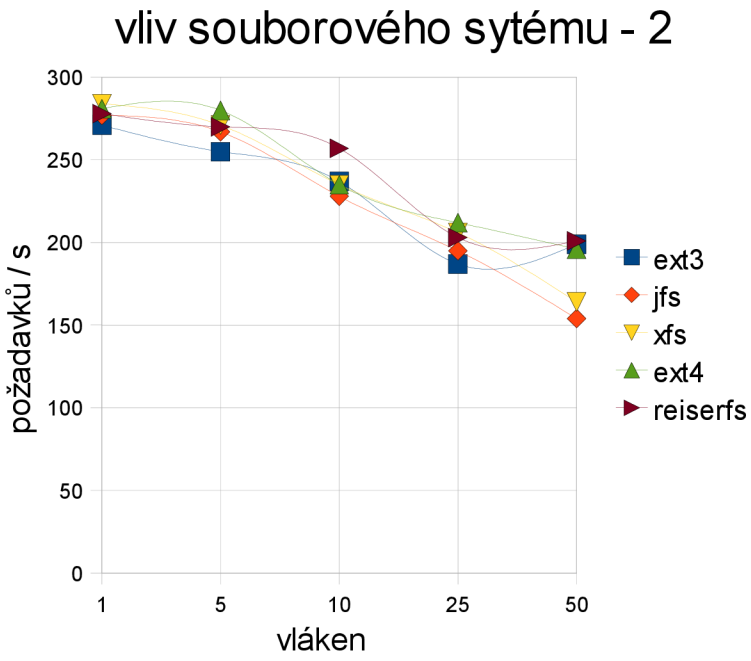
V případě jádra a OS obecně se pozornost ubírala především směrek k použitým filesystémům. Jejich nastavení testováno nebylo, neboť noatime má přínos neměřitelný a výchozí velikost bloků má být pro malé soubory, které se v modelové aplikaci vyskytují vhodná.

Souborový systém

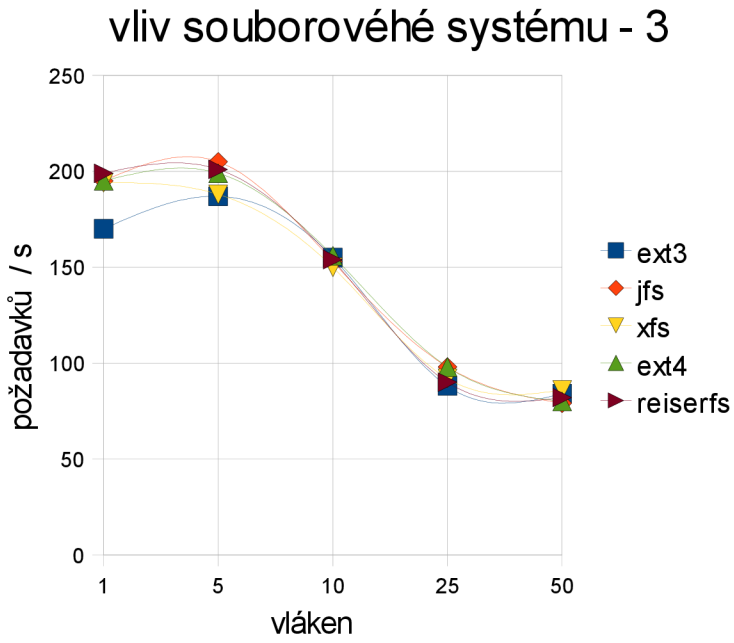
V prvním testu tedy bylo otestováno, zda bude mít použití jiného než v linuxu nejběžněji používaného EXT3 nějaký přínos. A testy potvrdily, že tomu tak opravdu je.



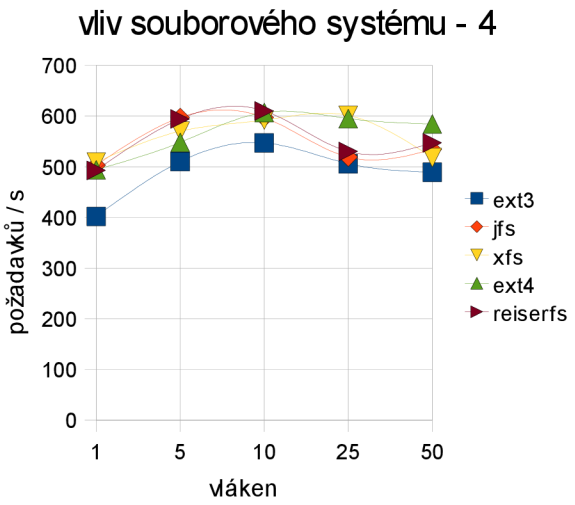
Graf 6: Vliv použitého filesystému



Graf 7: Vliv použitého filesystému



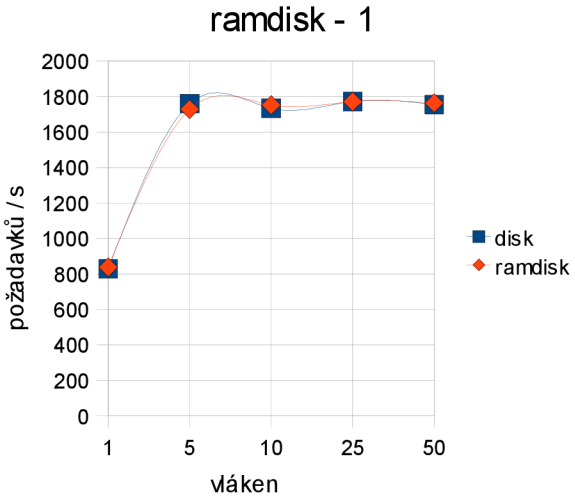
Graf 8: Vliv použitého filesystému



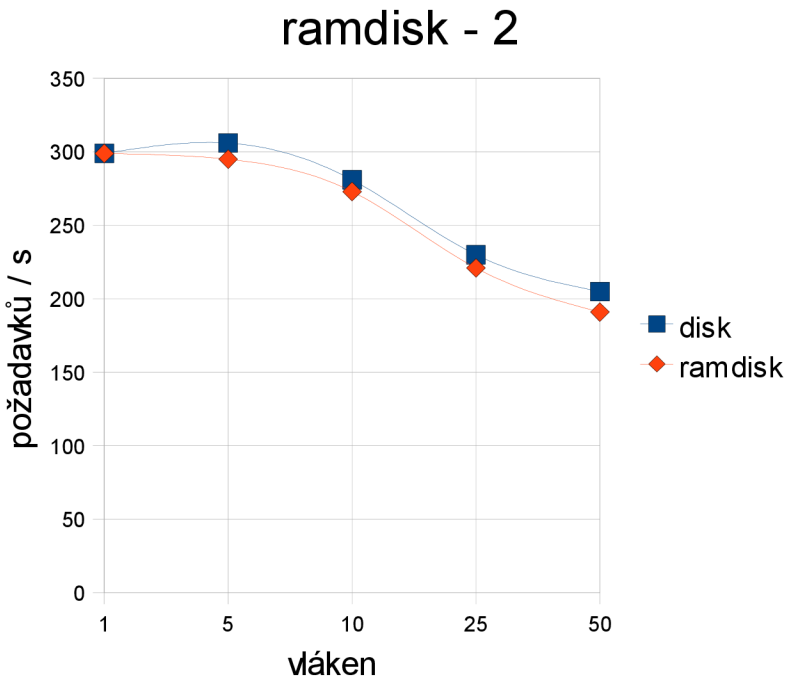
Graf 9: Vliv použitého filesystému

Výsledky testů ukazují, že nejhůře si vedl standardní EXT3, všechny další testované filesystémy prokázaly lepší výsledky. Nejlepším výkonem na modelové aplikaci se prokázaly EXT4 a ReiserFS. Oproti EXT3 s jejich použitím mohlo být obslouženo v průměru o 7% více požadavků, v konkrétních případech dokonce 15%. Tedy vliv souborového systému použitého na serveru opravdu existuje, byť rozdíly mezi výsledky testů za použití různých filesystémů jsou relativně malé.

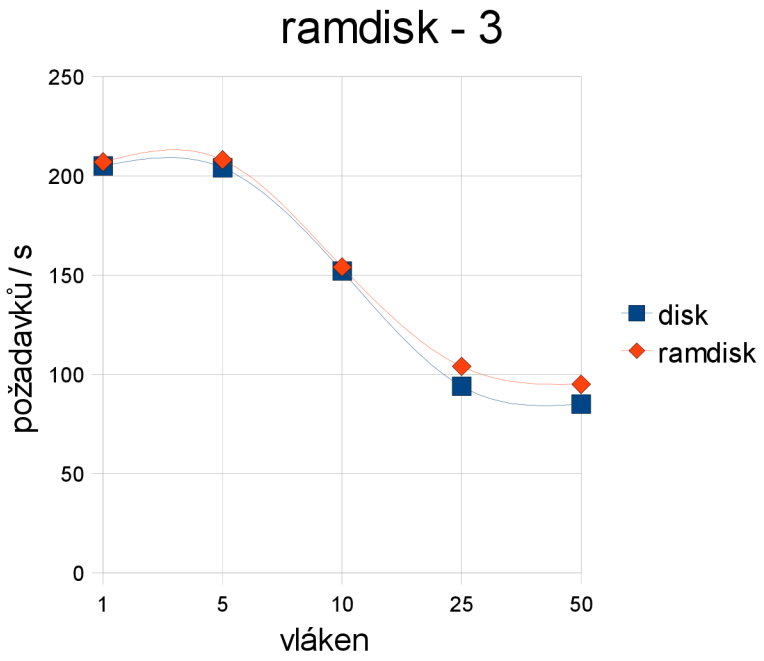
Vzhledem k souvislostem následovalo otestování použití ramdisku pro běh PHP skriptů přímo z operační paměti.



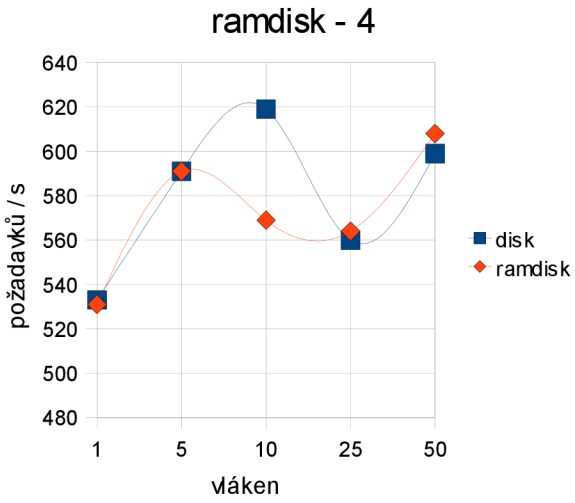
Graf 10: Vliv ramdisku



Graf 11: Vliv ramdisku



Graf 12: Vliv ramdisku

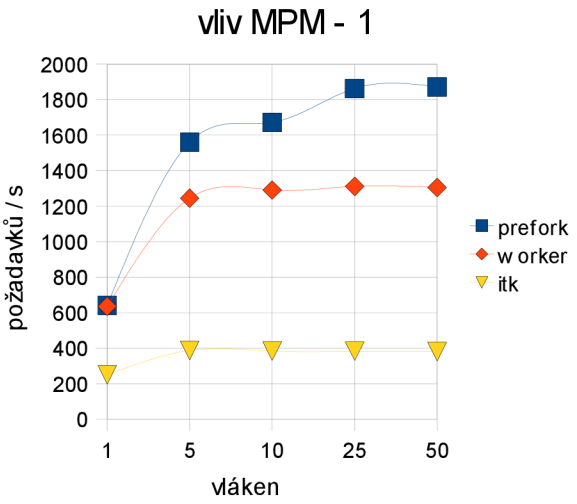


Graf 13: Vliv ramdisku

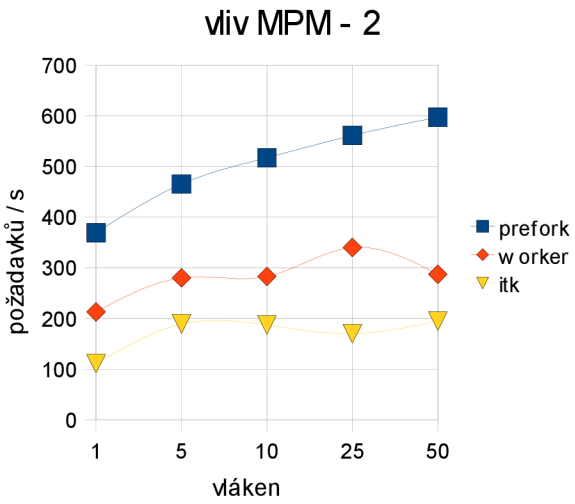
Z výsledků testů plyne, že ramdisk výkonu nepomohl, kromě testu v případě 4 v podstatě kopíruje případ, kdy není použit. V průměru přinesl zhoršení o 8%. Potvrdil se tedy předpoklad, že kešování na více místech užitek nepřinese. Dle výsledků tedy použití ramdisku nic neopodstatňuje, nepřináší nic co by vyvážilo jeho problematičtější udržování v aktuálním stavu a nutnost častého zálohování jeho obsahu pro případ výpadku napájení.

4.3.3.2 Apache

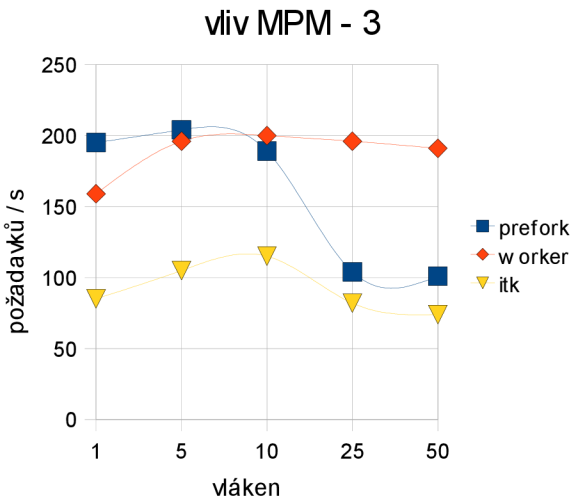
V případě Apache byl prvním provedeným testem výběr vhodného MPM modulu. Spolu s PHP jako modulem bylo možné použít pouze moduly PreFork a ITK, modul Worker pak s PHP běžícím přes FastCGI.



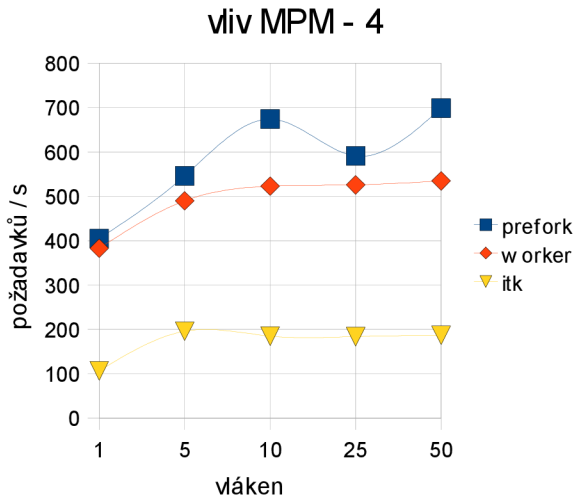
Graf 14: výkon MPM modulů - 1



Graf 15: výkon MPM modulů - 2



Graf 16: výkon MPM modulů - 3

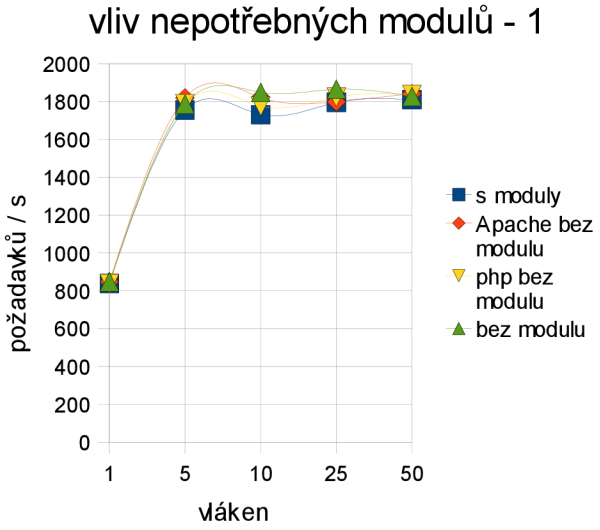


Graf 17: výkon MPM modulů - 4

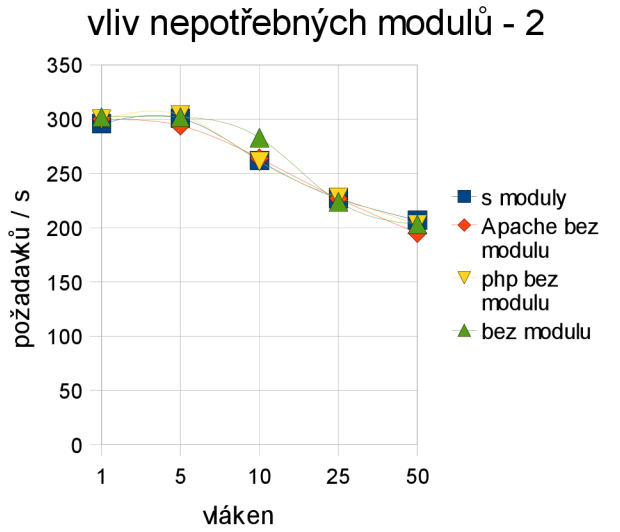
Výsledky ukazují, že nejlépe si vedl výchozí MPM prefork. Worker bývá považován za nejvýkonnější MPM pro Apache, nicméně v kombinaci s FastCGI ukázal jistou převahu pouze ve třetím testu.V ostatních případech ztrácel až 50%. MPM ITK bývá považován za pomalý a testy to potvrzují.

Vypnutí nepotřebných modulů

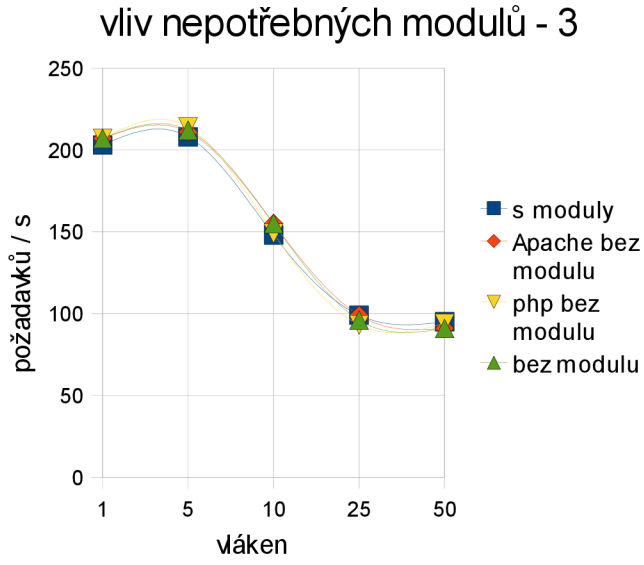
Jak bylo zmiňováno v kapitole o možnostech optimalizace, je vhodné vypnout nepotřebné a nepoužívané moduly. V této podkapitole jsou popsány výsledky po vypnutí těchto modulů v server Apache a PHP. Test je uveden zde pro obě komponenty, aby se zde nevyskytoval 2x téměř stejný test.



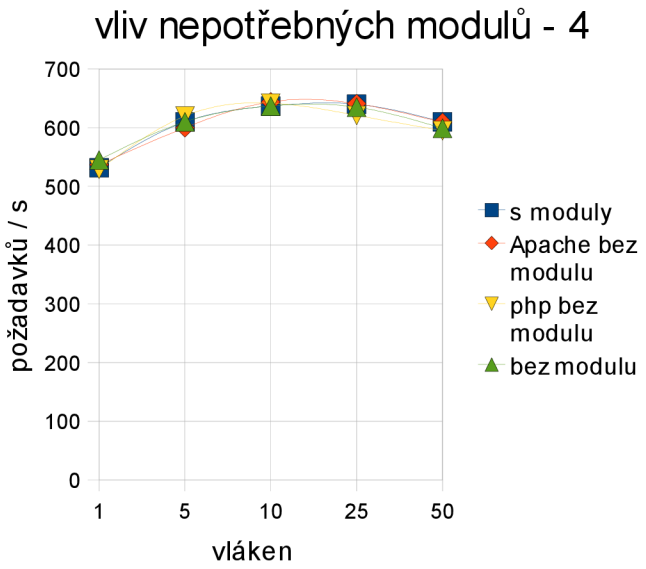
Graf 18: Vliv nepotřebných modulů na výkon - 1



Graf 19: Vliv nepotřebných modulů na výkon - 2



Graf 20: Vliv nepotřebných modulů na výkon - 3



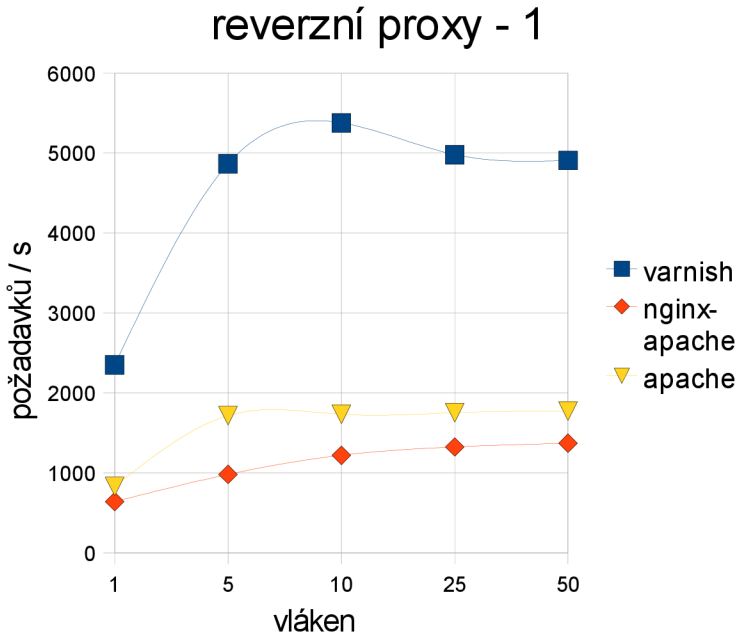
Graf 21: Vliv nepotřebných modulů na výkon - 4

Potrvrdil se dříve učiněný předpoklad, že vypnutí nepotřebných modulů nebude mít žádný zásadní vliv na výkon. V průměru totiž vyšlo, že se oproti původnímu stavu s aktivními moduly nic nezměnilo.

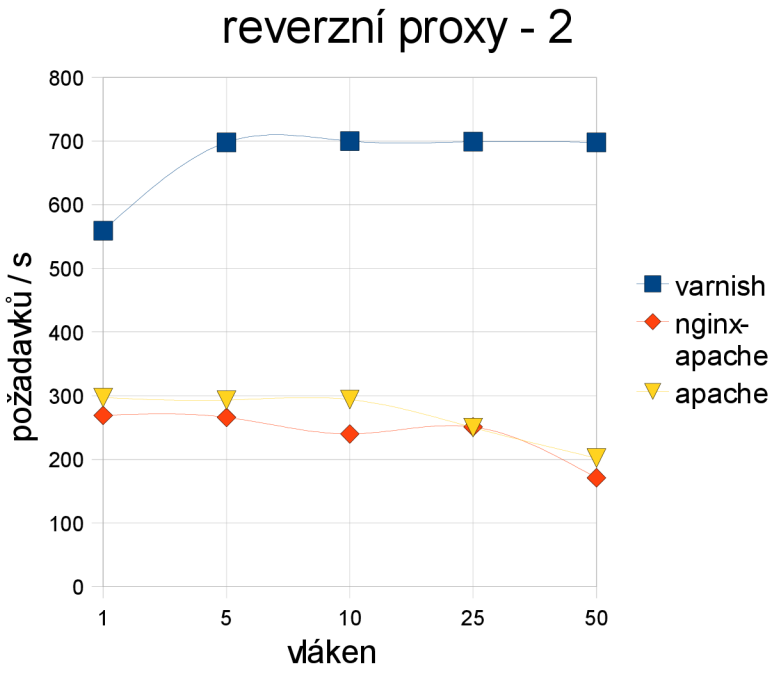
Reverzní proxy

Nasazení reverzních proxy na dynamický obsah má smysl pouze v případě, pokud tímto způsobem odkláníme obluhu statického obsahu na jiný rychlejší server, nebo proxy kešuje i dynamicky generované stránky.

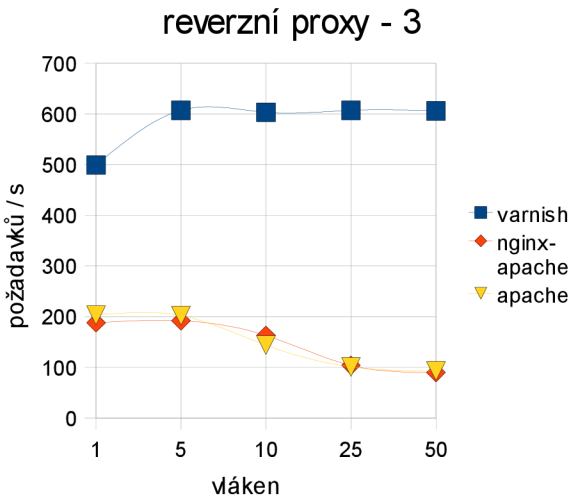
Testovaná byla reverzní proxy varnish, nginx je zde jako ukázka toho, že pouhé přeposílání požadavků dále stojí určitý procesorový čas. Původně bylo v plánu otestovat i mod_proxy + mod_cache z Apache, nicméně tato varianta se ukázala být nepoužitelná, neboť nedocházelo ke kešování výsledků. Dalším kandidátem pak byl Squid, ale ani nebyl otestován po zjištění výsledků při testu se statickým obsahem.



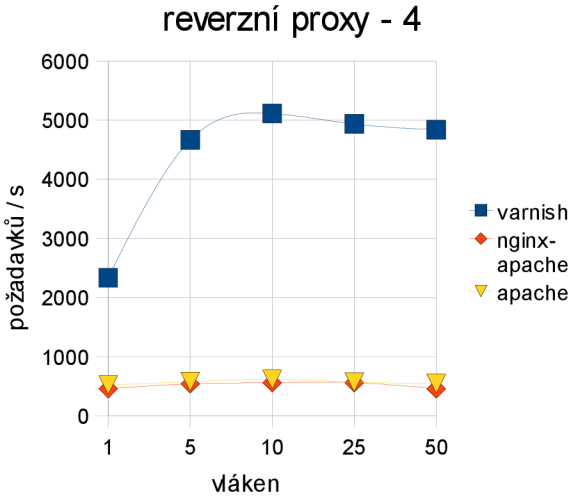
Graf 22: Výkon reverzních proxy - 1



Graf 23: Výkon reverzních proxy - 2



Graf 24: Výkon reverzních proxy - 3



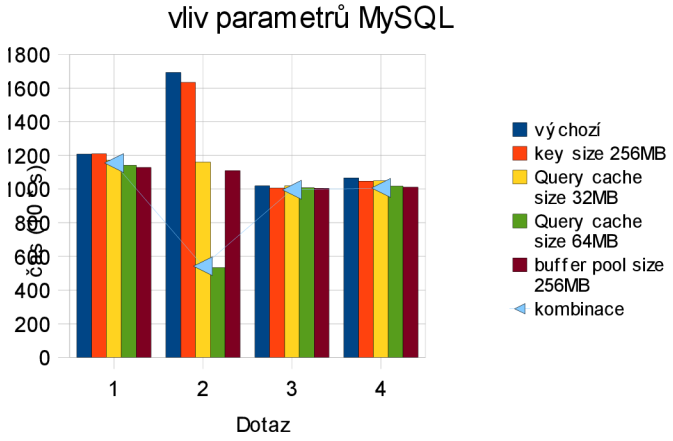
Graf 25: Výkon reverzních proxy - 4

Výsledky testů dopadly zajímavě. Režie s přeposíláním požadavků u nginx se potvrdila, díky ní zvládlo být v průměru obslouženo o 12% méně požadavků. Jediná skutečně kešující reverzní proxy – varnish se uvedla skvělým výsledkem, v průměru obsloužila 4,4 tolik požadavků jako samotný Apache, u posledního testovaného skriptu to pak bylo více než 8x tolik. Tyto čísla vypadají skvěle, na druhou stranu si je třeba uvědomit skutečnost, že ne všechny dynamicky generované stránky je možné kešovat. V případě, že je obsah stránky po každém načtení vygenerován uplně jiný není možné takovouto stránku do cache uložit bez následků na funkcionalitu webové aplikace (tedy této její konkrétní části). V tomto případě bude nutné buď upravit tuto stránku aby správně generovala

HTTP hlavičky týkající se možnosti jejího kešování a nebo konfiguraci reverzní proxy, aby tuto stránku vynechala z ukládání do cache.

4.3.3.3 MySQL

V případě MySQL byl výkon měřen pomocí upraveného PHP skriptu (viz popis v metodice). Testovaly se 4 SQL dotazy, opakované 50000x.



Graf 26: Čas potřebný ke zpracování SQL dotazů

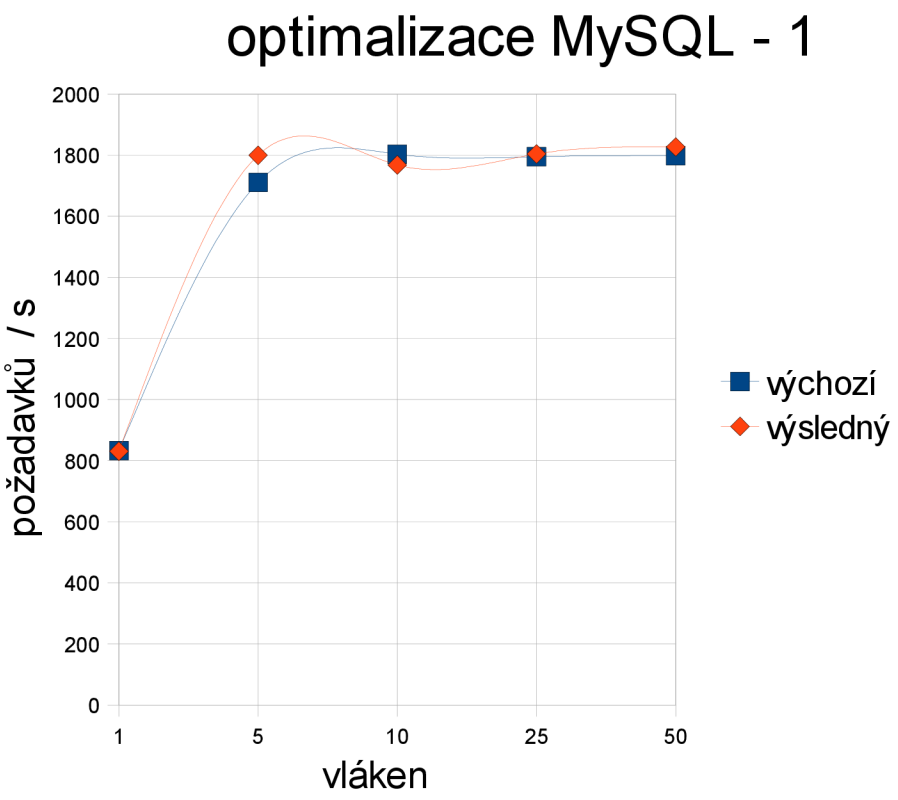
Předchozí graf zachycuje, jak se vyvíjel čas nutný pro zpracování dotazů po změně jednotlivých parametrů MySQL serveru. Jelikož jde o čas, menší hodnota je lepší.

Postupně byly vyzkoušeny tyto nastavení:

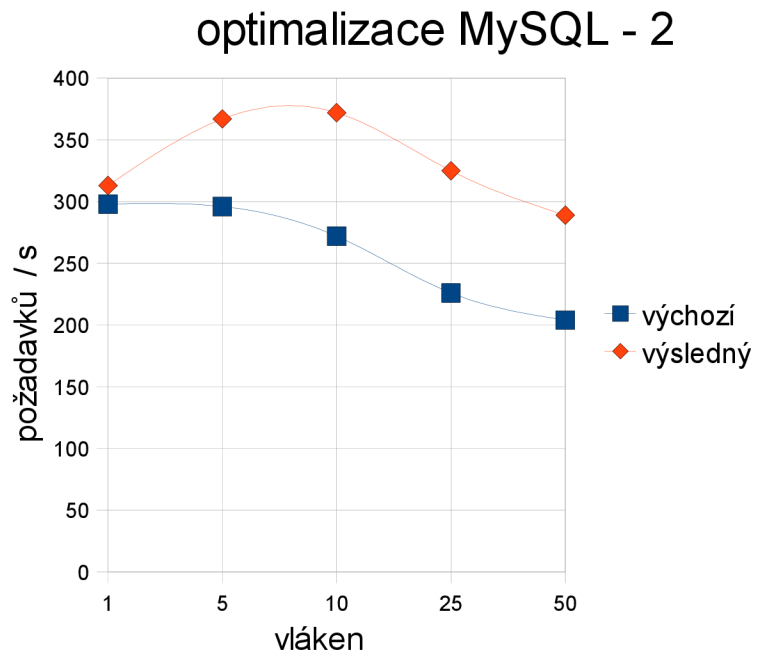
- key size 256MB
  - zvýšení key_size z 16MB na 256MB, tak, aby se do této paměti vešel celý obsah databáze, která čítá zhruba 50MB dat.
- Query cache size 32MB
  - zvýšení query_cache_size z 16 na 32MB pro uložení výsledků dotazů,
  - zvýšení query_cache_limit z 1 na 2MB.
- Query cache size 64MB
  - zvýšení query_cache_size z 16 na 64MB pro uložení výsledků dotazů,
  - zvýšení query_cache_limit z 1 na 4MB.
- Buffer pool size 256MB
  - zvýšení innodb_buffer_pool_size z 8 na 256MB,
  - zvýšení innodb_additional_mem_pool z 1 na 8MB.

- Kombinace
  - zvýšení innodb_buffer_pool_size z 8 na 256MB,
  - zvýšení innodb_additional_mem_pool z 1 na 8MB,
  - zvýšení query_cache_size z 16 na 64MB pro uložení výsledků dotazů,
  - zvýšení query_cache_limit z 1 na 4MB.

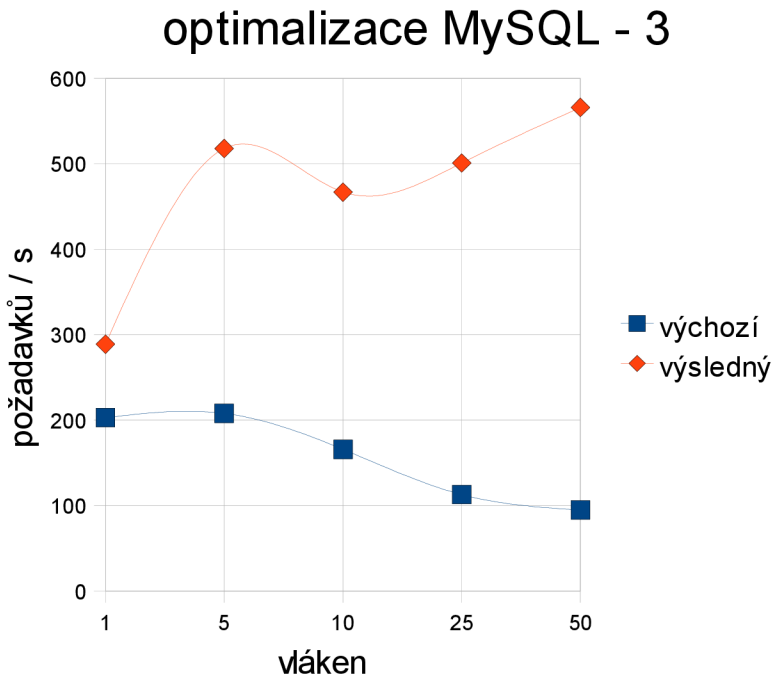
Po zhodnocení a aplikaci nejlepších hodnot proměnných vyšly následující výsledky:



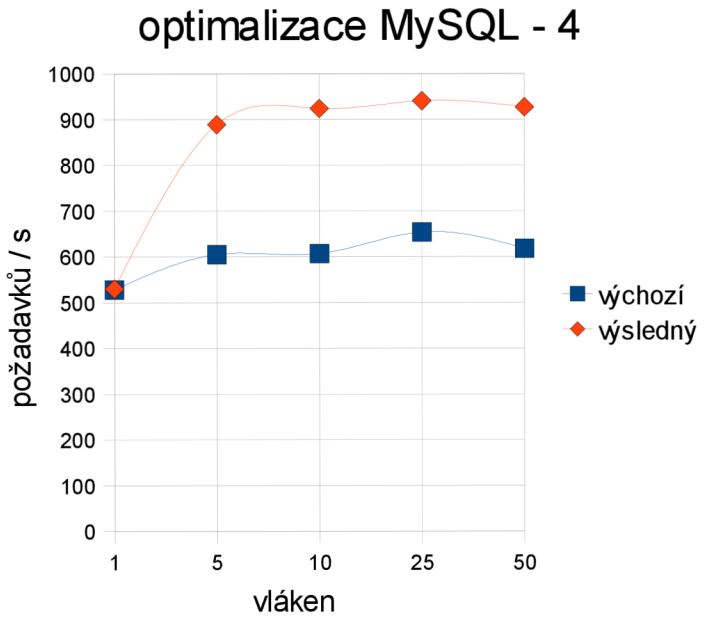
Graf 27: Výkon před a po optimalizaci MySQL - 1



Graf 28: Výkon před a po optimalizaci MySQL - 2



Graf 29: Výkon před a po optimalizaci MySQL - 3



Graf 30: Výkon před a po optimalizaci MySQL - 4

Kromě testu zaměřeného na první skript je ve všech případech vidět slušné zvýšení počtu obsloužených požadavků za vteřinu. V průměru dosahuje 68%! V případě testu třetího skriptu dochází k největšímu zlepšení, kdy oproti výchozímu nastavení zvládne server obsloužit šestinásobek požadavků.

4.3.3.4 PHP

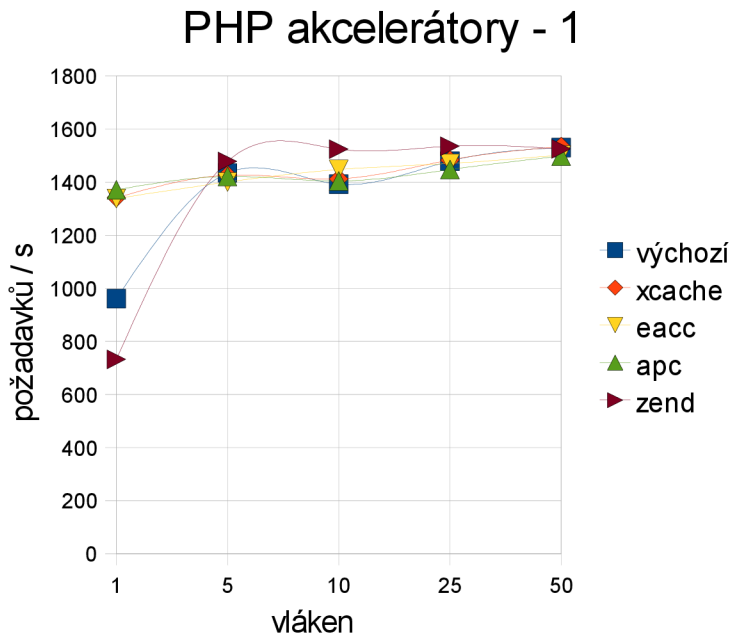
V případě PHP je nejzajímavější možností použití tzv. bytecode cache neboli PHP akcelerátory. Při testování proto začneme jimi.

PHP akcelerátory

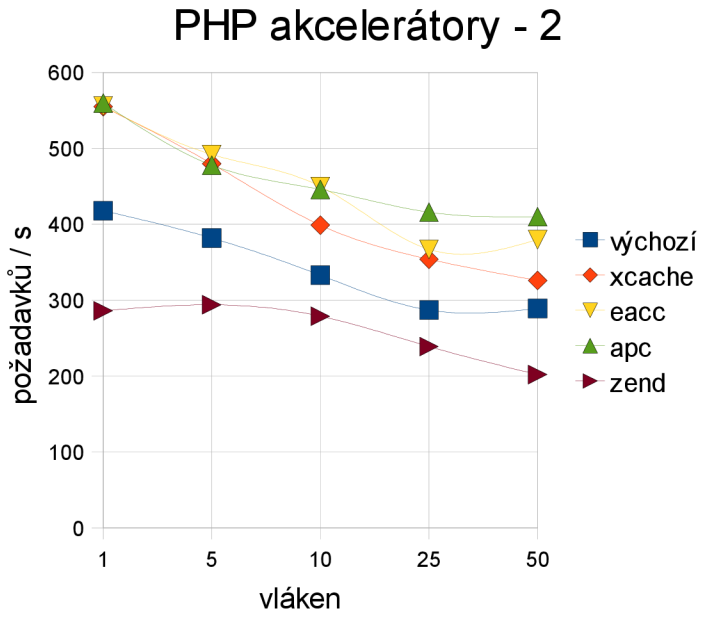
V testu jsme použili 3:

1. XCache,
2. EAccelerator,
3. APC.

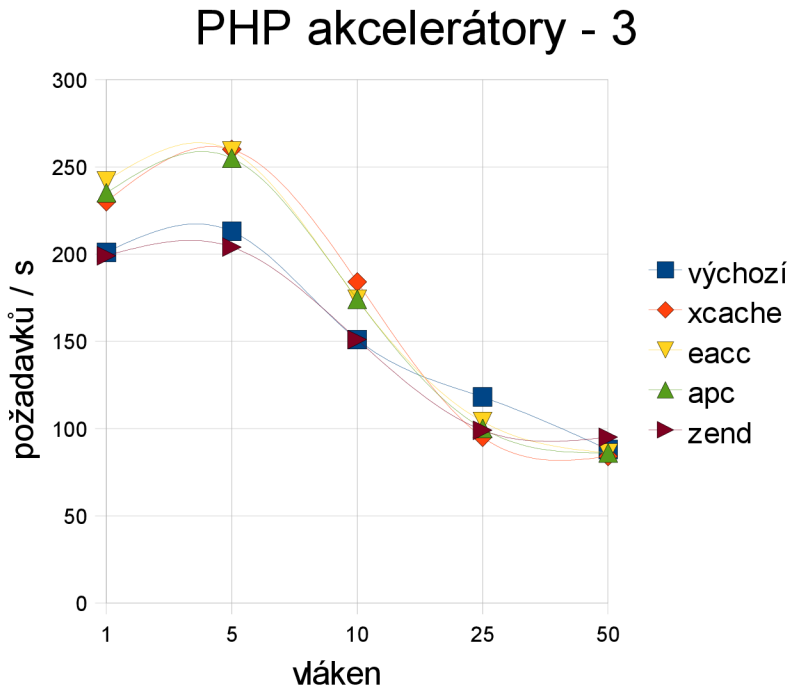
Plus specifický Zend Optimizer, který do této kategorie také jistým způsobem zapadá.



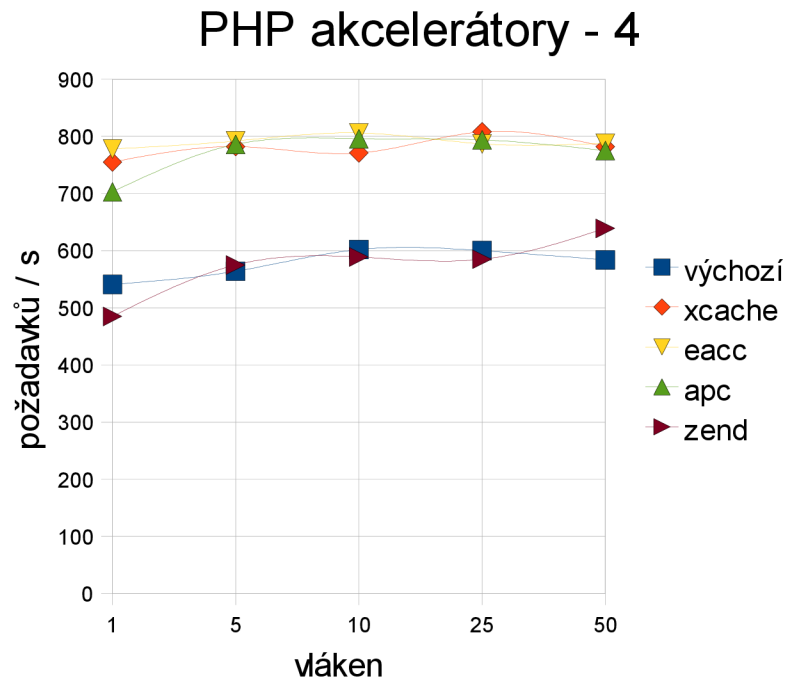
Graf 31: Výkon při použití PHP akcelerátorů - 1



Graf 32: Výkon při použití PHP akcelerátorů - 2



Graf 33: Výkon při použití PHP akcelerátorů - 3

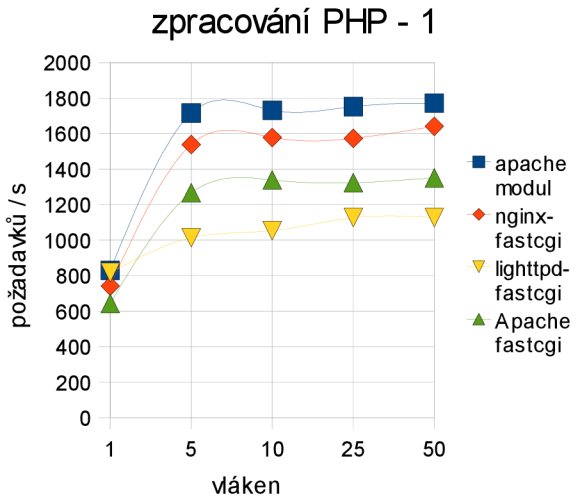


Graf 34: Výkon při použití PHP akcelerátorů - 4

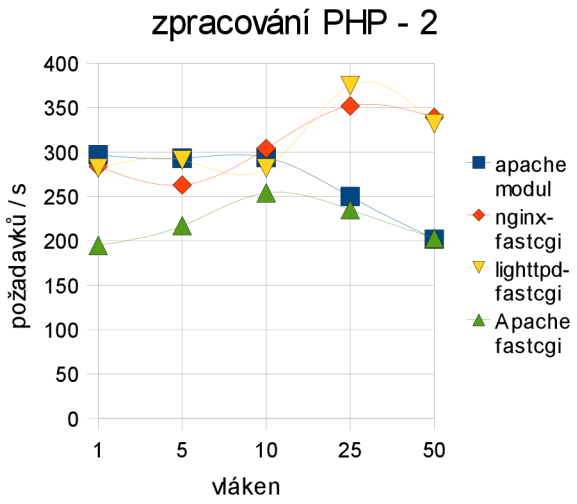
Nasazení PHP akcelerátorů je dle výsledků těchto testů naprostou nutností. Ty nejlepší z nich vylepšily výkon modelové aplikace o 36 respektive 34%, což je slušný výkon. Jednalo se o EAccelerator a APC, nicméně ani XCache si se svými 28% výkonu navíc nevedla špatně. Naopak Zend Optimizer se ukázal ve špatném světle a spíše uškodil než pomohl (pouze v testu 3. skriptu).

Způsob zpracování PHP skriptů

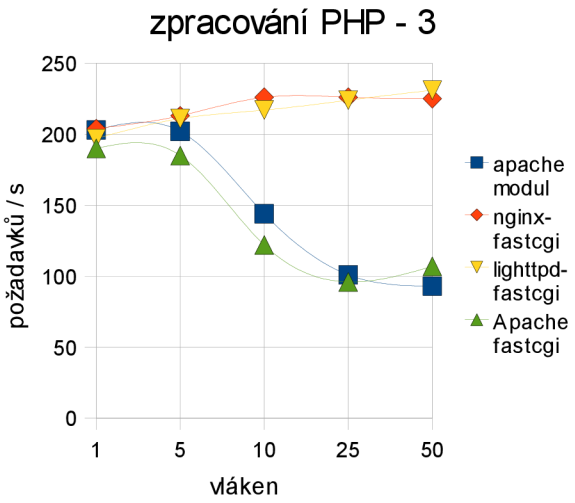
Následuje kapitola zhodnocující možné způsoby zpracování PHP skriptů na serveru. Jako možnosti byly otestovány Apache s modulem, Apache + FastCGI, nginx + FastCGI a lighttpd + FastCGI.



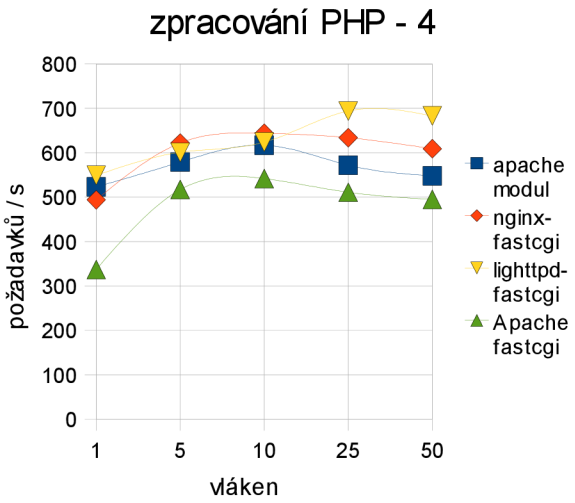
Graf 35: Způsob zpracování PHP vs výkon - 1



Graf 36: Způsob zpracování PHP vs výkon - 2



Graf 37: Způsob zpracování PHP vs výkon - 3



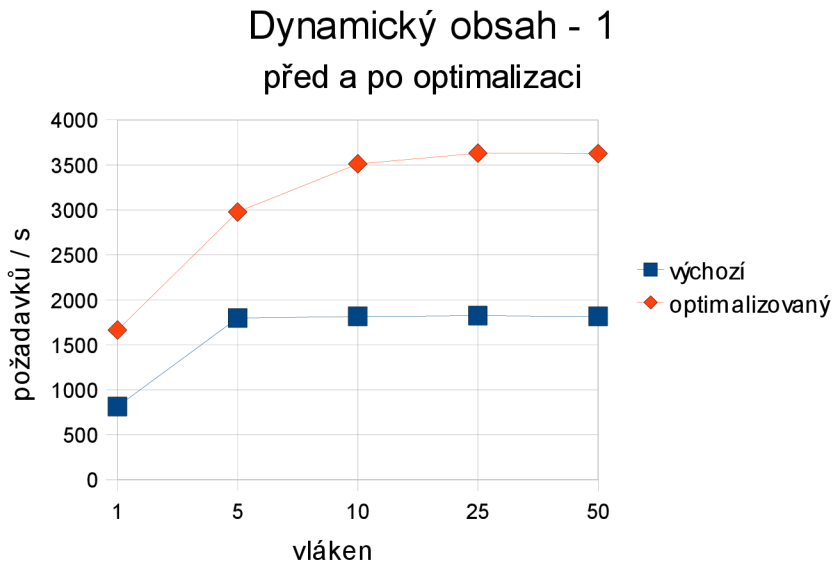
Graf 38: Způsob zpracování PHP vs výkon - 4

Z testu vychází jasně nejhůře varianta provuzu PHP přes FastCGI v rámci Apache. Naopak nejlépe dopadla implementace FastCGI v kombinaci s nginx, která výchozí Apache s PHP jako modulem převálcovala o 20%. Nicméně i Lighttpd se ziskem 16% výkonu navíc není k zahození.

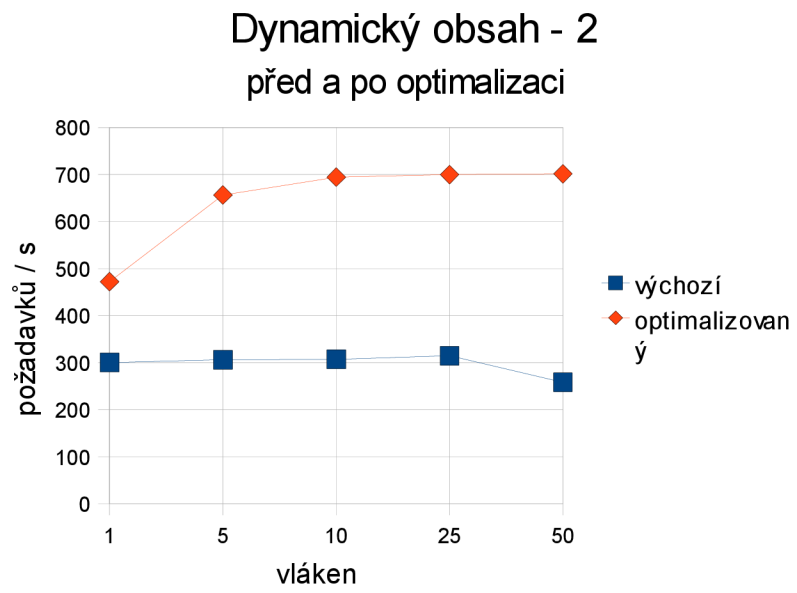
4.4 Optimalizace modelové aplikace

Po provedení všech testů uvedených v minulé kapitole přišlo na řadu nasazení kombinace těch nejslibnějších optimalizačních možností na modelové aplikaci, aby se ukázalo, nakolik přinesou zvýšení výkonu při jejich nakombinování dohromady.

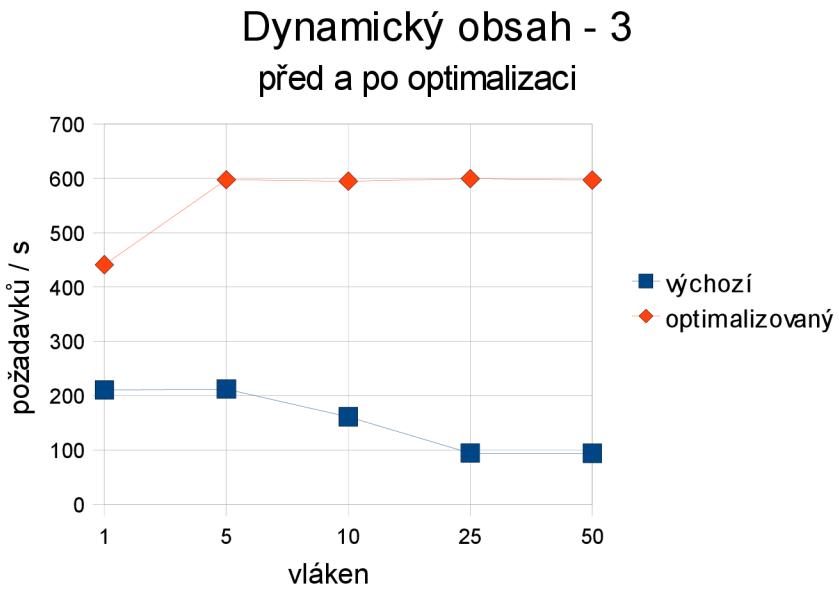
Bylo použito nginx jako reverzní proxy a zároveň HTTP server pro obsluhu statického obsahu. Pokud se vyskytly požadavky na dynamicky generovaný obsah provedl nginx jeho předání kešující reverzní proxy Varnish. Ta, pokud nedokázala obsloužit požadavek ze své keše, předala tento požadavek dále serveru Apache, který se staral o běh PHP skriptů. Na něm samozřejmě nechyběl PHP akcelerátor APC. U MySQL databáze pak došlo k nastavení keší a bufferů na stejné hodnoty jako v průběhu testu v předchozí kapitole. A toto celé se odehrávalo na diskovém oddílu naformátovaném pro souborový systém EXT4.



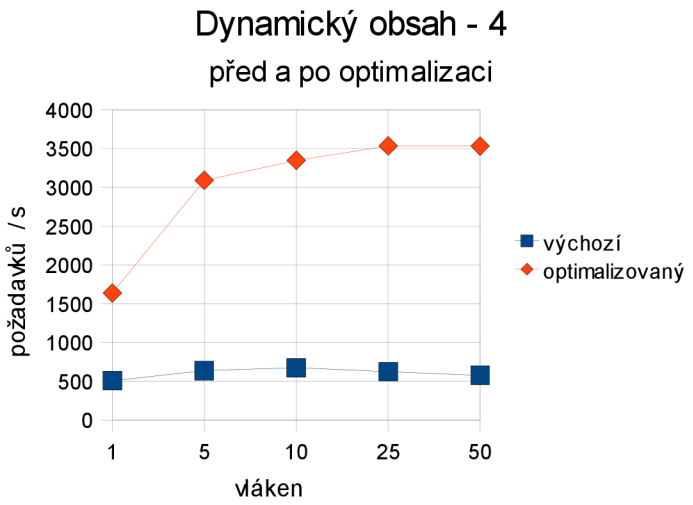
Graf 39: Před a po optimalizaci - dynamický obsah - 1



Graf 40: Před a po optimalizaci - dynamický obsah - 2

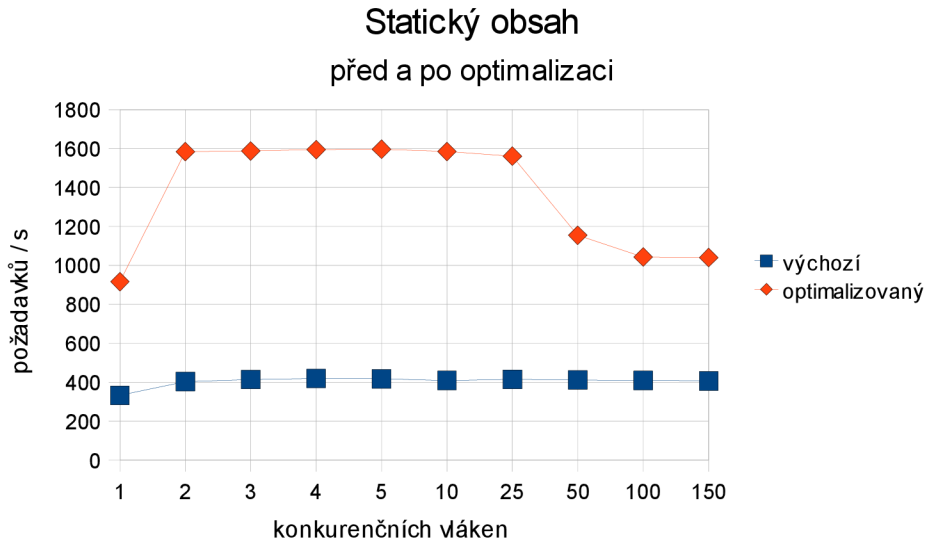


Graf 41: Před a po optimalizaci - dynamický obsah - 3



Graf 42: Před a po optimalizaci - dynamický obsah - 4

Výsledky na sebe nenechaly dlouho čekat a jak lze vidět z tohoto grafu rychlosti obsluhy dynamického obsahu, zrychlení stojí za to. Dosahuje v průměru 3,4 násobku původních hodnot naměřených s neoptimalizovaným Apachem i zbytekem systému.



Graf 43: Před a po optimalizaci - statický obsah

V poli statického obsahu (na sadě testovacích fotografií) se také dostavilo solidní zlepšení, v průměru zvládl server po optimalizaci obsloužit 3,4krát více požadavků za vteřinu.

Je tedy vidět, že tyto optimalizační možnosti fungují i dohromady, byť jejich kombinací člověk nezíská tolik výkonu navíc, kolik by vycházelo pouhým součtem zjištěných zlepšení v jednotlivých testech.

4.5 Sada doporučení

Na základě výsledků provedených testů byla vypracována sada doporučených optimalizací pro LAMP platformu. Opět optimalizace jsou rozděleny podle jednotlivých součástí LAMP.

4.5.1 Linux

4.5.1.1 Výběr souborového systému

Standardně na serveru budeme mít souborový systém EXT3, testy však ukázaly, že jeho výkon je ze všech běžně používaných nejhorší. Na základě výsledků testů doporučujeme místo něj používat **EXT4** nebo **ReiserFS**. Pokud máme rádi novinky tak zvolíme EXT4. Naopak pro konzervativnější uživatele je vhodný ReiserFS

4.5.1.2 Nastavení parametrů jádra

Nastavením parametrů jádra uvedených v kapitole o jádře se dle výsledků měření získalo navíc pár procent výkonu navíc. Vzhledem k tomu, že tyto volby nepřenastavují nic přímo kritického, můžeme jedině doporučit jejich nastavení.

4.5.2 Apache

4.5.2.1 Odklonění obsluhy statického obsahu

Z testů jasně plyne, že oddělení obsluhy statického obsahu z Apache na jiný HTTP server přináší výkon navíc. Dle výsledků na modelové aplikace je dopuručeno použít HTTP serveru **nginx** nebo **lighttpd.**

4.5.2.2 Použití kešující reverzní proxy

Před Apache doporučujeme předřadit reverzní proxy, která bude provádět kešování dynamicky generovaného obsahu. V testech se ukázal být jako použitelný pouze **Varnish**, který je navíc velice flexibilní z důvodu svého vlastního VCL konfiguračního jazyka. Tím má uživatel plnou kontrolu nad tím, který obsah se do keše uloží a který ne. Dále je třeba u všech skriptů, jimiž vygenerovaný obsah je žádoucí uložit do keše, zkontrolovat, zda neposílají HTTP hlavičky, které kešování brání, např.:

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

jinak by varnish pouze předával požadavky backend serveru, který by musel obsah znovu a znovu generovat. V takovém případě by došlo naopak ke zpomalení. V případě, kdy většina generovaného obsahu nebude kešovatelná pak ztrácí použití reverzní proxy význam a toto doporučení vynechejte.

4.5.2.3 Výměna Apache za server s FastCGI

V testech se ukázalo, že existují rychlejší možnosti z hlediska generování dynamického obsahu než PHP jako modul Apache a to HTTP servery využívající FastCGI ke zpracování PHP. Dle testů si nejlépe vedl **nginx** (navíc ho doporučujeme i pro statický obsah). Nicméně zde je třeba vzít do úvahy (a otestovat), zda je naše aplikace schopná na FastCGI běžet bez problémů. Nicméně pak už bychom mohli těžko tomuto serveru říkat, že je postavený na LAMPu.

4.5.2.4 MPM modul Prefork

Pokud budeme používat Apache pro zpracování PHP skriptů, rozhodně použijeme Prefork MPM modul. Worker je sice údajně rychlejší, ale testy to v kombinaci Worker + FastCGI neprokázaly. Navíc Prefork je sázkou na jistotu, je nejlépe otestovaný.

4.5.3 MySQL

4.5.3.1 Nastavení bufferů a cache

U MySQL se ukázalo, že je hlavně třeba správně nastavit velikosti proměnných key_size, query_cache_size, query_limit_size, innodb_buffer_pool_size a innodb_additional_mem_pool. Doporučujeme nastavit je na hodnoty dle možností vašeho systému a způsobu práce aplikací s databází. Nelze doporučit žádné obecně fungující hodnoty, musí se vyzkoušet co pomůže na konkrétním systému.

4.5.4 PHP

4.5.4.1 Využití PHP akcelerátorů

Jednoduchou instalací PHP akcelerátorů dosáhneme poměrně slušného zisku výkonu. Navíc není třeba vůbec zasahovat do zdrojových kódů webových aplikací. Vzhledem k tomuto je jejich použití silně doporučeno, dle výsledků jsou dobré **APC** a **EAccelerator.**

5 Reálné nasazení

V předchozí části práce byly prozkoumány možnosti zvýšení výkonu webového serveru postaveného na platformě LAMP. Následně byly přínosy jednotlivých opatření otestovány na modelové aplikaci a na základě toho byla vytvořena sada doporučení. V této kapitole budeme tato doporučení aplikovat na reálnou aplikaci a ověříme si, zda i v tomto případě dojde k očekávanému vylepšení výkonu.

5.1 Popis prostředí

Testování bude probíhat na serveru hostujícím několik málo virtuálních hostů (domén). Ze strany software jde o čerstvě nainstalovanou distribuci Debian Etch, čili parametry souhlasí s testovacím serverem použitým u modelové aplikace. Kromě funkce webového serveru zastává i funkci mailserveru, VPNky, samba serveru (sdílení souborů a tiskový server), což nás v tomto případě nemusí příliš zajímat, neboť všechny tyto ostatní služby je nutné na serveru provozovat a nelze tedy aplikovat doporučení o vypnutí nepotřebných služeb. Z hlediska hardware jsou parametry následující:

- procesor Intel Celeron 2,13GHz,
- 1GB DDR RAM paměti,
- 640GB SATA pevný disk Western Digital (připojený na SATAI řadič),
- onboard Fast Ethernet síťová karta Realtek.

5.2 Popis aplikace

Jak již bylo dříve zmíněno, na serveru je hostováno několik webů. Nejvýznamnější a nejnavštěvovanější z nich je úložiště.com (http://www.uloziste.com/). Jedná se vlastně o relativně malý internetový upload server. Relativně, neboť měsíční objem dat z něj odeslaných překračuje 1TB a stále narůstá. Samotné jeho fungování je jednoduché:

1. Dynamický obsah, tedy PHP skripty starající se o upload souborů na server, vytváření nových účtů apod. jsou umístěny na hlavní doméně *www*. Nejvíce náročným skriptem, a tedy kandidátem na provedení zátěžového testu, je skript *register2.php*, který nahrává uploadované soubory na správné místo a uživatelům vypisuje jejich dříve nahrané soubory. Kromě tohoto skriptu provedeme testování i na *monitoring-serveru.php,* který při každém spuštění provádí dotazy do relativně velké MySQL databáze.
2. Veškerý statický obsah je pak odsunut na subdoménu *files.uloziste.com*. PHP je na této doméně zcela vypnuté. V jednotlivých adresářích jsou uloženy uživateli nahrané soubory. Zde bude možné otestovat optimalizace zaměřené na statický obsah. K tomuto účelu bude ze statistik stahovaných dat vybráno 30 nejvíce stahovaných souborů.

5.3 Aplikovaná doporučení

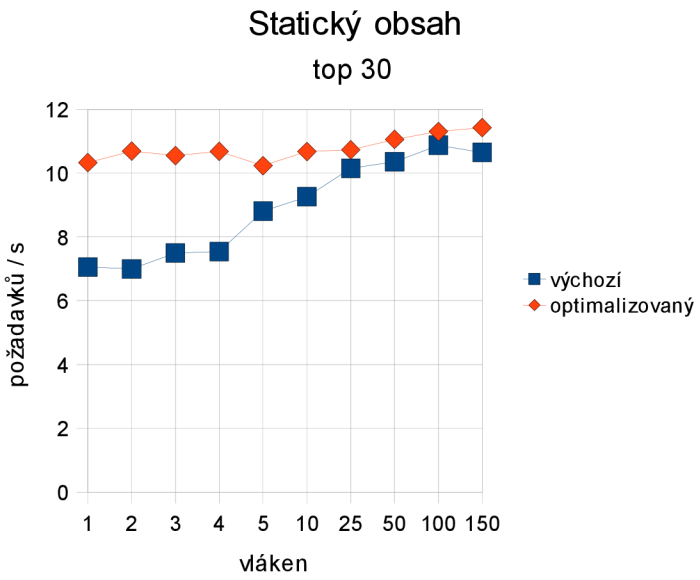
Vyjdeme ze sady doporučení sestavených v minulé kapitole. Jelikož byla aplikace navržena od začátku s úmyslem mít oddělený statický a dynamický obsah, můžeme jednoduše nasadit nginx jako HTTP server pro obsluhu statického obsahu. Před Apache, který se bude starat o zpracování PHP skriptů pak bude předřazena kešující reverzní proxy Varnish. U samotného Apache použijeme místo MPM modul Prefork. Dále na PHP použijeme PHP akcelerátor APC pro kešování zkompilovaných PHP skriptů. K tomu provedeme u MySQL nastavení bufferů a keší dle nastavení uvedeného v kapitole o testování (data v DB jsou menší než v testovaném případě, ale z důvodu budoucího růstu budou takto nadhodnocené hodnoty vhodné). Dále pak dojde k přesunu dat na souborový systém EXT4. A nakonec ještě nastavíme parametry jádra.

Při aplikaci těchto doporučení se ukázalo, že se mohou vyskytnout problémy. V tomto případě se jednalo především o problém s nginx a statickým obsahem – aplikace využívala možnosti poskytnuté serverem Apache ohledně customizace výpisů obsahu adresářů. U verze nginx dostupné jako balíček v Debianu 4.0 byl tento problém neřešitelný a bylo nutné stáhnout aktuální stabilní verzi a příslušný modul, který tuto funkcionalitu přidává. Hlavní část problému tím byla vyřešena, nicméně kosmetické rozdíly oproti původnímu stavu existují a budou snad vyřešeny v další verzi nginx.

5.4 Dosažené výsledky

5.4.1 Statický obsah

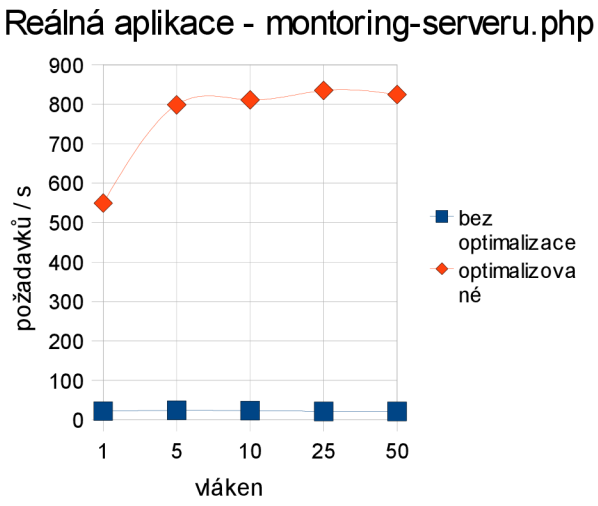
Výsledek testu dopadl poněkud hůře, než jsme očekávali. Průměrný výkonnostní zisk v tomto případě dosáhl „pouze“ výše 24%. Je to stejně jako v případě testu provedeného s velkými soubory v předešlé kapitole způsobeno jejich velikostí - průměrná velikost souboru je přes 20MB, celkem přes 650MB. Není tak možné je všechny mít v keši (tento server má pouze 1GB paměti a její poměrně velká část je stále obsazena jinými službami) a dochází k intenzivnímu využívání pevného disku.



Graf 44: Statický obsah - optimalizovaný vs výchozí stav

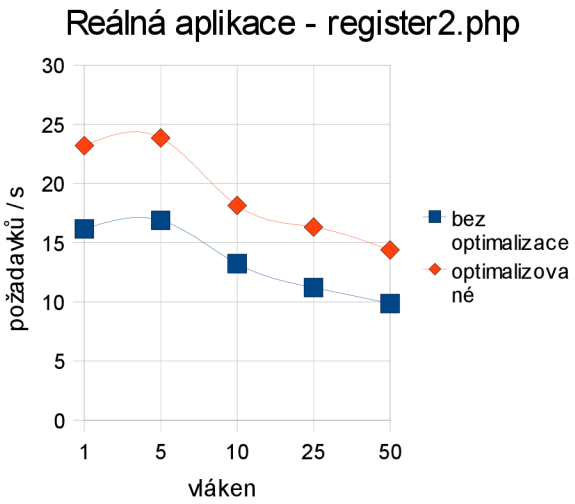
5.4.2 Dynamický obsah

Situace u dynamického obsahu bylo čekání na výsledek napínavější, neboť se mohlo prokázat zlepšení způsobené mj. nastavením MySQL databáze a použití PHP akcelerátoru, na druhou stranu první testovaná adresa byla ponechána s nastavením hlaviček jako nekešovatelná, tudíž varnish zde působil jako prostředník, což se na výkonu dle předchozích testů podepsalo. U skriptu monitoring-serveru.php naopak bylo žádoucí kešování, protože obsah se generuje na základě dat měnících se v intervalu 10minut.



Graf 45: Vliv aplikovaných doporučení na dynamický obsah

U skriptu monitorig-serveru.php se tento očekávaný výsledek opravdu dostavil, zrychlení díky kešující reverzní proxy bylo v průměru téměř 34 násobné.



Graf 46: Vliv aplikovaných doporučení na dynamický obsah

Naopak u měření provedeného na skriptu register2.php došlo ke zrychlení pouze v průměru o 43%.

5.4.3 Zhodnocení

Celkově toto zrychlení způsobené pouhým vyladěním nastavení a drobnou obměnou použitého software je příjemným překvapením. Ikdyž se vyskytly drobné problémy při jejich aplikaci, výsledek stál za to.

6 Závěr

6.1 Dosažené výsledky

První část této práce stručně přibližuje tematiku serverové platformy LAMP – její základní součásti, důvody proč se často používá apod. Poté následuje kapitola diskutující různé možnosti a nápady, jak jednotlivé části zrychlit.

Jako součást této diplomové práce bylo naprogramována modelová webová aplikace ve skriptovacím jazyce PHP s využitím MySQL pro uložení dat. Právě tato aplikace byla použita při provádění testů. Ty tvořily poměrně značnou práce, neboť bylo potřeba zjistit, zda dříve popsané možnosti přinášejí žádané zlepšení nebo naopak zhoršení. Pro tento účel tak bylo vytvořeno a popsáno několik testovacích metod. K jejich provedení pak bylo vytvořeno několik jednoduchých skriptů..

Po zhodnocení výsledků provedených testů byla vypracována sada doporučení říkající, co je vhodné na LAMP serveru změnit a nastavit, aby bylo dosaženo zvýšení jeho výkonu. Jde zejména o použití PHP akcelerátoru, jiného HTTP serveru pro obsluhu statického obsahu, nasazení kešující reverzní proxy a v neposlední řadě správné nastavení velikosti keše a bufferů MySQL databáze.

Při jejich použití vzrostl výkon serveru při obsluze modelové aplikace zhruba na 3,4 násobek původní hodnoty jak u statického tak dynamického obsahu. Po aplikaci na reálné aplikaci však byl nárust pouze 24% u statického obsahu díky jeho specifikům - velikosti souborů. U dynamického obsahu pak 43% v případě, kdy nebyla stránka kešována a při kešované stránce dosáhl zoptimalizovaný server 34krát vyššího počtu obsloužených požadavků.

Tedy úvodní předpoklad, že výkon webového serveru postaveného na platformě LAMP lze zvýšit i vhodným nastavením software a použitím jiných technlogií se jasně potvrdil. Byť zlepšení mohlo být ještě lepší, není tato získaná možnost obsloužit více klientů se stejným hardware vůbec špatná.

6.2 Přínos práce

Možnostmi optimalizací výkonu webových serverů se zabývá poměrně velký počet publikací, nicméně zaměřují se většinou pouze na dílčí problémy či řeší pouze jednotlivé součásti – HTTP nebo databázové servery. Chybí v nich celkový pohled na všechny součásti tvořící danou serverovou platformu. Tato diplomová práce takovýto ucelený pohled přináší a to na populární platformu LAMP.

Na základě mnoha provedených sad testů, jak na statický tak i dynamický obsah, byly vybrány nejslibnější optimalizační metody. Na jejich základě byla vypracována sada doporučení pro

zoptimalizování výkonu LAMP serveru. Tyto jsou aplikovatelná na širokou škálu různých typů webových aplikací provozovaných na těchto serverech..

Aplikací sady doporučení na server, na kterém běží reálná aplikace, došlo ke zvýšení počtu požadavků, které je schopen obsloužit za vteřinu, o 24% v případě statického obsahu. V případě dynamicky generovaného obsahu pak šlo o zlepšení o 43%

6.3 Možnosti dalšího rozvoje

Tato práce nahlíží na problematiku výkonu webového serveru pouze ze softwarového pohledu. Nicméně neméně důležitý je i pohled z hlediska hardware. Proto by bylo žádoucí se v rámci budoucí práce zaměřit na tuto hardwarovou část problému. Nepochybně by bylo zajímavé zjistit, zda je z hlediska výkonostního přínosu vhodnější používat ve webových serverech několik vícejádrových procesorů (tj. jeden drahý server) nebo méně (třeba i pomalejších) procesorů/jader ve více samostatných (levnějších) serverech spojených do clusteru, zda stojí za to investice to SAS disků místo levnějších SATA, použití lépe taktovaných RAM atp.

Literatura

[1] Sklar, D. *PHP 5 moduly, rozšíření, akcelerátory* / Vyd. 1. Brno : Zoner Press, 2005. 341 s. ISBN 80-86815-19-6

[2] Castagnetto, J. *Programujeme PHP profesionálně* / 2. oprav. a aktualiz. vyd. Brno : Computer Press, 2004. xxiv, 656 s. ISBN 80-7226-310-2

[3] Wikipedia: *LAMP (software bundle)*. Wikipedia, the free encyclopedia. 2009, [online; navštíveno 5.1.2009].

URL http://en.wikipedia.org/wiki/LAMP_(software_bundle)

[4] Netcraft Ltd: April *2009 Web Server Survey*. 2009, [online; navštíveno 18.5.2009].

URL http://news.netcraft.com/archives/web_server_survey.html

[5] The Apache Software Foundation: *Apache HTTP Server Project - Related Projects*. 2008, [online; navštíveno 5.1.2009].

URL http://httpd.apache.org/related_projects.html

[6] The PHP Group: *APC Runtime Configuration Manual*. 2009, [online; navštíveno 5.1.2009].

URL http://cz.php.net/manual/en/apc.configuration.php

[7] VN L.A.M.P.: *Howto: Performance Benchmarks a Web server.* 2009, [online; navštíveno 5.1.2009].

URL http://forum.vnlamp.com/showthread.php?t=144

[8] Wikipedia: *Web server benchmarking.* Wikipedia, the free encyclopedia. 2008, [online; navštíveno 5.1.2009].

URL http://en.wikipedia.org/wiki/Web_server_benchmarking

[9] Milde, D. *PHP jako FastCGI pod Apachem*. 2008, [online; navštíveno 5.1.2009].

URL http://blog.milde.cz/linux/PHP-jako-FastCGI-pod-Apachem/

[10] Milde, D. *Možnosti spolupráce PHP a Apache. 2008,* [online; navštíveno 5.1.2009].

URL http://blog.milde.cz/programovani/Moznosti-spoluprace-PHP-a-Apache/

[11] Lee, J. *Open source :vývoj webových aplikací Linux, Apache, MySQL, Perl a PHP* / Vyd. 1. Brno : Mobil Media, 2003. 448 s. ISBN 80-86593-43-6

[12] The Apache Software Foundation: Apache Performance Tuning. 2008, [online; navštíveno 5.1.2009].

URL http://httpd.apache.org/docs/2.2/misc/perf-tuning.html

[13] Suchý, M. *Reverzní proxy.* Internet Info, s.r.o. 2004, ISSN 1212-8309, [online; navštíveno 5.1.2009].

URL http://www.root.cz/clanky/reverzni-proxy/

[14] Jelsoft Enterprises Ltd.: *Reduce Apache's Load With lighttpd On Debian Etch.* 2008, [online; navštíveno 5.1.2009].

URL http://www.howtoforge.com/reduce-apache-load-with-lighttpd-debian-etch

[15] Jelsoft Enterprises Ltd.: *How To Set Up A Caching Reverse Proxy With Squid 2.6 On Debian Etch.* 2008, [online; navštíveno 5.1.2009].

URL http://www.howtoforge.com/how-to-set-up-a-caching-reverse-proxy-with-squid-2.6-on-debian-etch

[16] Wikipedia: *Reverse proxy.* Wikipedia, the free encyclopedia. 2009, [online; navštíveno 5.1.2009].

URL http://en.wikipedia.org/wiki/Reverse_proxy

[17] Bowen, R. *Caching Dynamic Content with Apache httpd.* O'Reilly Media, Inc.2006, [online; navštíveno 5.1.2009].

URL http://www.onlamp.com/pub/a/onlamp/2006/11/16/apache-mod_cache.html

[18] Likins, A. *System Tuning Info for Linux Servers*. Red Hat, Inc. 2001, [online; navštíveno 5.1.2009].

URL http://people.redhat.com/alikins/system_tuning.html

[19] Marti, D. *Apache and Firewall Performance Tips from the Xenu.net Masters.* Linux Journal. *2002,* [online; navštíveno 5.1.2009].

URL http://www.linuxjournal.com/article/6041

[20] Welling, L. *MySQL :průvodce základy databázového systému* / Vyd. 1. Brno : Computer Press, 2005. 255 s. ISBN 80-251-0671-3

[21] Sun Microsystems, Inc.: *MySQL 5.0 Reference Manual.* 2009, [online; navštíveno 23.5.2009]

URL http://dev.mysql.com/doc/refman/5.0/en/index.html

[22] Zend Technologies Ltd.: *Zend Guard*. 2009, [online; navštíveno 22.5.2009]

URL http://www.zend.com/en/products/guard/

Seznam příloh

Příloha 1. Tabulky s výsledky testů

Příloha 2. DVD se zdrojovými kódy modelové aplikace, testovacími skripty, textem této práce a plakátkem.

Příloha 1. Tabulky s výsledky testů

V této příloze jsou tabulky zachycující údaje změřené v průběhu testování. Není-li uvedeno jinak, jsou hodnoty počty požadavků za vteřinu.

Tabulka 1 – statický obsah (fotky), různé HTTP servery (Graf 2)

vláken	apache	lighttpd	boa	thttpd	nginx	Apache – cach
1	332,46	910,13	901,15	715,14	961,96	318,8
2	403,57	1570,88	1565,01	1092,47	1572,6	388,71
3	413,84	1545,66	1526,11	1164,14	1632,37	407,5
4	419,31	1538,16	1527,2	697,85	1619,18	407,45
5	417,24	1545,22	1527,05	707,57	1614,76	409,26
10	408,79	918,6	734	727	1608,56	407,39
25	414,62	872,03	698,12	712	761,68	402,3
50	411,58	867,04	710,93	717	744,39	402,67
100	409,04	862,16	737,56	711	747,7	399,64
150	406,42	853,11	698,41	716	753,42	397,04

Tabulka 2 – statický obsah (fotky), různé reverzní proxy (Graf 3)

vláken	apache	squid	varnish
1	332,46	248,91	751,36
2	403,57	239,38	1012,72
3	413,84	231,07	1056,27
4	419,31	231,29	1044,74
5	417,24	226,45	1067,31
10	408,79	203,38	1059,15
25	414,62	188,1	1007,32
50	411,58	186,18	1005,27
100	409,04	190,48	1000,52
150	406,42	192,38	990,71

Tabulka 4 – statický obsah (velké soubory), různé HTTP servery (Graf 4)

vláken	apache	lighttpd	boa	thttpd	nginx	apache+kernel
1	9,1	9,56	9,04	8,48	9,77	9,62
5	9,51	8,8	6,38	7,5	8,46	9,69
10	9,49	7,86	5,73	5,89	8,39	9,73
25	9,3	5,82	3,35	4,47	6,63	9,35
50	9,12	3,81	2,28	1,66	6,53	9,29
100	4,05	2,67	2,52	1,29	6,41	4,38
150	2,95	2,98	2,85	1,5	6,22	3,39

Tabulka 5 – statický obsah (velké soubory), různé HTTP servery (Graf 5), propustnost v MB/s

vláken	apache	lighttpd	boa	thttpd	nginx	apache+kernel
1	60,75	63,88	60,74	59,27	63,72	63,29
5	64,1	59,44	40,72	48,97	59,57	64,15
10	62,97	55,18	38,76	39,49	58,85	64,05
25	61,24	39,01	20,32	25,36	43,91	63,1
50	59,52	22,67	12,66	9,46	40,91	62,61
100	19,59	13,46	11,27	4,69	40,8	20,12
150	11,89	11,76	11,51	4,03	40,61	14,97

Tabulka 6 – dynamický obsah, vliv různých souborových systémů (Graf 6-9)

test	vlaken	ext3	jfs	xfs	ext4
1	1	747	809	791	803
	5	1666	1809	1771	1703
	10	1783	1846	1829	1821
	25	1767	1845	1813	1838
	50	1801	1841	1799	1849
2	1	271	277	284	281
	5	255	267	271	280
	10	237	228	235	235
	25	187	195	206	212
	50	199	154	164	196
3	1	170	195	194	195
	5	187	205	188	199
	10	155	153	150	156
	25	88	98	92	98
	50	84	79	86	80
4	1	402	503	509	494
	5	510	597	570	548
	10	547	597	592	607
	25	506	519	601	595

Tabulka 7 – dynamický obsah, vliv použití ramdisku (Graf 10-13)

test	vláken	bez ramdisku	s ramdiskem
1	1	829	838
	5	1760	1728
	10	1733	1752
	25	1772	1771
	50	1754	1763
2	1	299	299
	5	306	289
	10	281	273
	25	198	221
	50	213	191
3	1	205	207
	5	204	208
	10	152	154
	25	94	104
	50	85	95
4	1	533	531
	5	591	591
	10	619	569
	25	543	564

Tabulka 8 – dynamický obsah, vliv použitého MPM (Graf 14-17)

test	vláken	prefork	worker	itk
1	1	641	644	250.34
	5	1560	1267	387.47
	10	1672	1339	384.39
	25	1863	1324	383.76
	50	1872	1352	381.19
2	1	369	195	111.24
	5	465	217	189.11
	10	517	254	187.97
	25	561	235	170.62
	50	597	203	195.00
3	1	77	8	85.35
	5	204	185	105.44
	10	189	122	115.19
	25	104	96	82.17
	50	101	107	74.49
4	1	405	337	107.46
	5	546	517	196.25
	10	674	542	185.79
	25	591	511	184.74
	50	699	495	187.22

Tabulka 9 – dynamický obsah, vliv nepotřebných modulů (Graf 18-21)

test	vláken	s moduly	bez apache	bez php	oboji
1	1	839	848	838	848
	5	1756	1819	1788	1787
	10	1731	1822	1776	1849
	25	1796	1799	1823	1865
	50	1810	1842	1838	1828
2	1	296	300	300	302
	5	301	294	304	302
	10	262	264	261	283
	25	227	227	266	224
	50	247	175	203	203
3	1	203	207	207	207
	5	208	211	214	212
	10	148	155	149	155
	25	99	99	93	96
	50	95	91	94	91
4	1	532	537	529	545
	5	610	600	620	610
	10	637	644	642	632
	25	640	641	621	551
	50	610	609	595	599

Tabulka 10 – dynamický obsah, vliv reverzních proxy (Graf 22-25)

test	vláken	varnish	Nginx + apach	apache
1	1	2349	641	830
	5	4865	981	1716
	10	5377	1220	1732
	25	4980	1324	1753
	50	4907	1372	1772
2	1	559	269	297
	5	698	266	293
	10	700	240	294
	25	699	251	250
	50	698	171	202
3	1	499	188	203
	5	607	192	202
	10	603	163	144
	25	607	104	101
	50	606	90	93
4	1	2336	459	523
	5	4663	542	579
	10	5113	561	617
	25	4934	561	572
	50	4837	458	548

Tabulka 11 – dynamický obsah, vliv mysql optimalizací (Graf 27-30)

test	vláken	výchozí
1	1	833
	5	1711
	10	1803
	25	1795
	50	1799
2	1	298
	5	296
	10	272
	25	226
	50	204
3	1	203
	5	208
	10	166
	25	113
	50	95
4	1	528
	5	605
	10	607
	25	654

Tabulka 12 – dynamický obsah, vliv PHP akcelerátorů (Graf 31-34)

test	vláken	výchozí	xcache	eacc	apc	zend
1	1	961	1341	1338	1371	732
	5	1434	1426	1400	1421	1478
	10	1392	1412	1447	1404	1524
	25	1479	1483	1470	1447	1535
	50	1530	1534	1502	1498	1525
2	1	418	555	556	560	286
	5	382	480	492	478	294
	10	333	399	502	446	279
	25	287	454	367	316	239
	50	289	326	380	470	202
3	1	60	230	242	235	199
	5	213	260	259	255	204
	10	151	184	174	174	151
	25	118	95	104	100	99
	50	88	84	86	86	95
4	1	541	755	778	703	485
	5	564	782	792	786	575
	10	602	771	806	796	589
	25	600	808	787	794	585

Tabulka 13 – dynamický obsah, vliv způsobu zpracování PHP (Graf 35-38)

test	vláken	modul	nginx	lighttpd	apache fastcgi
1	1	830	742	818	644
	5	1716	1539	1014	1267
	10	1732	1579	1054	1339
	25	1753	1574	1127	1324
	50	1772	1642	1128	1352
2	1	297	284	282	195
	5	293	263	291	217
	10	294	304	282	254
	25	250	352	375	235
	50	202	339	332	203
3	1	203	204	197	190
	5	202	213	211	185
	10	144	226	217	122
	25	101	226	224	96
	50	93	225	231	107
4	1	523	494	549	337
	5	579	622	601	517
	10	617	644	624	542
	25	572	634	694	511
	50	548	609	683	495

Tabulka 14 – optimalizovaná modelová aplikace – dynamický obsah (Graf 39-42)

Test	vlaken	výchozí	optimalizovaný
1	1	813,52	1664,26
	5	1797,3	2976,93
	10	1815,68	3512,12
	25	1823,53	3630,1
	50	1815,23	3627,1
2	1	300,4	472,16
	5	306,3	656,43
	10	307,26	694,46
	25	314,77	699,83
	50	258,53	701,47
3	1	210,86	441,1
	5	212,58	597,47
	10	161,37	594,66
	25	94,7	599,33
	50	94,34	596,83
4	1	508,9	1637,93
	5	637,87	3090,1
	10	676,08	3348,65
	25	625,16	3533,4

Tabulka 15 – optimalizovaná modelová aplikace – statický obsah (fotografie) (Graf 43)

vláken	výchozí	optimalizovaný
1	332,46	916,08
2	403,57	1584,06
3	413,84	1587,6
4	419,31	1594,63
5	417,24	1596,62
10	408,79	1585,12
25	414,62	1561,03
50	411,58	1155,15
100	409,04	1043,29
150	406,42	1040,43

Tabulka 16 – aplikace doporučení na reálném serveru na statickém obsahu (Graf 44)

vláken	výchozí	optimalizovaný
1	7,06	10,33
2	7	10,69
3	7,5	10,55
4	7,54	10,68
5	8,81	10,24
10	9,26	10,68
25	10,15	10,73
50	10,36	11,06
100	10,88	11,31
150	10,65	11,43

Tabulka 17 – aplikace doporučení na reálném serveru na dynamickém obsahu (Graf 45-46)

testovaný skript	vláken	bez optimaliza	optimalizované
register2.php	1	16,16	23,21
	5	16,9	23,85
	10	13,23	18,15
	25	11,21	16,32
	50	9,86	14,39
monitoring-serveru.php	1	22,3	549,39
	5	24,12	798,63
	10	23,5	811,02
	25	21,61	835,06
	50	21,31	824,54