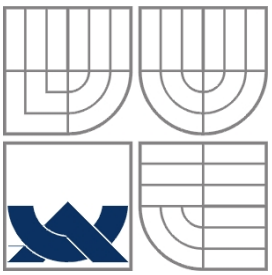
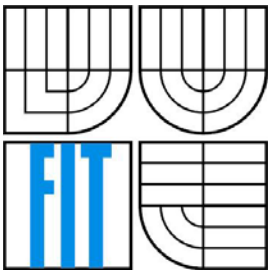


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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)

OPTIMIZING LAMP (LINUX/APACHE/MYSQL/PHP) PERFORMANCE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Igor Lamoš
AUTHOR

VEDOUCÍ PRÁCE — Ing. Adam Herout, Ph.D.
SUPERVISOR

BRNO 2009

Zde prosím vložit originál zadání bakalářské práce.

(Ve druhém výtisku nahradit originál zadání jeho kopií)

Abstrakt

Cílem práce bylo shrnout a ověřit nejzásadnější faktory ovlivňující výkon operačního systému Linux, webového serveru Apache, databázového serveru MySQL a PHP. Práce obsahuje důležité nastavení a jejich popis pro každou uvedenou součást webového serveru. Podle možnosti jsou u jednotlivých nastavení provedené testy na modelové aplikaci a zhodnocené jejich výsledky. Na základě výsledků testů byl sestaven seznam doporučujících nastavení. Jejich správnost byla ověřená aplikováním na reálné systémy. V obou případech došlo ke zvýšení výkonu aplikací a serverů.

Abstract

The goal of this work is to summarize and verify the fundamental factors affecting performance of Linux operating system, Apache web server and performance of MySQL database system and PHP. This work contains important settings for each part of webserver with corresponding description. Where possible, there are perfomance tests made on the model application and the results are reviewed. Based on the tests results, the recommended settings list was assembled. Accuracy of this recommendations was verified by applying the settings to real-world applications. In both cases, there was performance improvement.

Klíčová slova

Linux, Apache, PHP, MySQL, Debian, ApacheBench, ab, webový server, databázový server, výkon, zátěžové testy, ladění výkonu

Keywords

Linux, Apache, PHP, MySQL, Debian, ApacheBench, ab, webserver, database server, performance, load testing, performance tuning

Citace

Lamoš Igor: Možnosti optimalizace výkonu LAMP, bakalářská práce, Brno, FIT VUT v Brně, 2009

Možnosti optimalizace výkonu LAMP

Prohlášení

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

……
Igor Lamoš
20.5.2009

Poděkování

Za přínosné konzultace technické stránky práce bych rád poděkoval panu Ing. Jiřímu Vrbovi, Ph.D. ze společnosti QCM, s.r.o.

Obsah

1 Úvod

Apache je již od roku 1996 nejpoužívanější webový server. Podle statistiky společnosti Netcraft [7] z dubna 2009 má s počtem instalací přes 106 milionů v rámci celého Internetu podíl přes 45%. Statistika navíc uvádí, že z milionu nejvytíženějších webových serverů světa je na více než dvou třetinách nainstalován právě Apache.

Hlavním cílem této práce je popsat možnosti optimalizace výkonu webového serveru Apache ve spojení s PHP a databázovým serverem MySQL. Toto téma jsem si vybral, protože už několik let pracuji v oblasti informačních technologií jako vývojář webových aplikací a okrajově se zabývám konfigurací webových serverů na platformě Linux. Ke zvolení tohoto tématu také přispělo zakoupení mého vlastního webového serveru Dell PowerEdge R300, na kterém budou prováděny veškeré testy.

Hlavní kapitola bude věnována provádění zátěžových testů a rozdělena na jednotlivé části – Apache, MySQL, PHP a nastavení operačního systému. Tyto části budou obsahovat obecná doporučení pro vyladění výkonu a zátěžové testy s grafickým zobrazením výsledků. V rámci těchto kapitol bude také popsán vliv jednotlivých nastavení na výkon vůči výchozí konfiguraci. Na základě výsledků provedených testů se budu věnovat aplikaci vzniklých doporučení na reálné systémy.

Přestože se snažím uvádět nejlepší možné nastavení všech součástí, snažte se prosím pochopit, že obvykle není možné vytvořit jednoduché, rychlé a jednoznačné nastavení, které bude spolehlivě fungovat na všech serverech. Každá aplikace naprogramována v jazyku PHP běžící na webovém serveru Apache, databázovém serveru MySQL a operačním systému Linux je unikátní a s tímto vědomím je potřeba k této problematice přistupovat. Co funguje na jednom serveru nemusí spolehlivě a se stejnými účinky na výkon fungovat na serveru druhém. Skutečná nastavení se mohou lišit a měla by být otestována v konkrétním prostředí.

2 Testovací prostředí

Testování jsem prováděl na vlastním webovém serveru umístěném na páteři Internetu s konektivitou 100Mbps. Pokud není v práci uvedeno jinak, testy byly prováděny po Internetu na vyhrazené lince o rychlosti 10Mbps.

2.1 Hardware

Webový server Dell PowerEdge R300 byl v následující konfiguraci:

- **Procesor** Intel Xeon QuadCore 2.83GHz, 2x 6MB L2 Cache
- **Operační paměť** 12GB DDR2 667MHz (6x 2GB Dual Ranked DIMM)
- **Pevné disky** 2x 146GB SAS 15.000 ot./min., RAID1 (SAS6iR)
- **Síťová karta** Intel PRO 1000PT (Gigabit)

2.2 Software

Na serveru byl nainstalován virtualizační nástroj VMware ESX server verze 3.5.0. Virtualizaci jsem použil z důvodu budoucího víceúčelového použití serveru. Počas provádění testů však na serveru běžel pouze jeden virtuální počítač – testovaný server.

Operační systém distribuce Debian verze 4.0 s jádrem 2.6.18 byl nainstalován ve výchozí konfiguraci bez grafického prostředí, s minimální instalací balíčků a včetně následujícího softwaru:

- Apache 2.2.3
- PHP 5.2.0
- MySQL 5.0.32

2.3 Aplikace

Testování probíhalo na zvlášť připavené aplikaci naprogramované v PHP. Jednalo se o binární strom, který obsahoval 50 tisíc uzlů. U každého uzlu byl vygenerován náhodný text o velikosti v intervalu 2 – 3 kB. Celková velikost databáze byla 150 MB.

Stránka, na které byly prováděné zátežové testy obsahovala algoritmus, který si vybral náhodný uzel ve druhé polovině stromu. Počínaje tímto uzlem byly následně vybrány všechny podřízené uzly do úrovně 5 až 10. Textové informace uložené u všech specifikovaných uzlů byly následně vypsány.

Aplikace byla z hlediska PHP napsána objektově orientovaným stylem a její zdrojové kódy jsou přiloženy na CD.

Databáze a dotazy SQL byly navrženy s maximálním ohledem na výkon a tudíž nedocházelo ke zkreslení výkonu z důvodu špatného návrhu.

2.4 Nástroj pro testování

Samotné testování bylo prováděno nástrojem Apache Benchmark (ab), který je součástí webového serveru Apache. Tento nástroj podporuje všechny funkce, které jsou potřebné pro získání korektních výsledků.

Z důvodu snahy o odstranění vedlejších faktorů, které mohli mít potenciální vliv na měření výkonu, byl každý test, který je zde uveden, proveden několikrát. Použité byly ty nejlepší výsledky.

3 Faktory ovlivňující výkon

Moderní webové servery jsou provozované na řadě počítačů, kterých architektura se může značne lišit, no faktory ovlivňující výkon jsou ve všech případech prakticky stejné.

3.1 Procesor

V případě poskytování statického obsahu nehrá procesor serveru velkou roli. Webový server totiž v takovém případě vůbec nezajímá, co je obsahem souboru, který si klient vyžádal.

Naopak, v případě poskytování dynamického obsahu jde o důležitou komponentu. Každá aplikace zapojená do procesu vyřízení dynamického požadavku totiž potřebuje nějaký procesorový čas. V případě vysoké zátěže může dojít k vyčerpání zdrojů procesoru a výkon serveru začíná klesat.

3.2 Operační paměť

Nejhlavnější komponentou webového serveru je operační paměť RAM. Přístup do operační paměti je několikrát rychlejší než přístup k datům na disku. Pro optimální výkon je však potřeba vyladit konfiguraci serveru tak, aby nikdy nedošlo k použití virtuální paměti operačního systému – swap. V tom případě se totiž vše, co by bylo za normálních okolností uloženo do RAM, ukládá na pevný disk a výrazně se prodlužuje reakční doba serveru.

3.3 Pevné disky

Další hardwarovou částí přímo ovlivňující výkon je pevný disk. Webový server stráví značnou část tím, že přistupuje na disk. Protože jsou disky ještě stále velmi pomalé (ve srovnání např. s operační paměti), představují poměrně často hlavní příčinu špatného výkonu.

Můžeme říci, že pevné disky jsou všeobecným faktorem omezujícím výkonnost každého systému.

4 Testování a optimalizace výkonu

Zátěžové testy byly provedeny pro dlouhodobou i krátkodobou zátěž. Za krátkodobou zátěž považujme odeslání 1 000 požadavků, za dlouhodobou odeslání 10 000 požadavků na server. V obou případech bude použita úroveň konkurence od 25 do 400 klientů. Jednotlivé hodnoty můžeme vidět v horní části grafů. Barevné označení je ve všech grafech zachováno.

Jako vyřízený budeme považovat pouze takový požadavek, na který server dokázal odpovědět v intervalu 15 vteřin. V opačném případě nás čas odpovědi nezajímá a požadavek považujeme za nevyřízený.

4.1 Apache

Webový server Apache byl nainstalován z balíčků operačního systému. Nebyl tedy kompilován ze zdrojových kódů.

4.1.1 Výchozí konfigurace

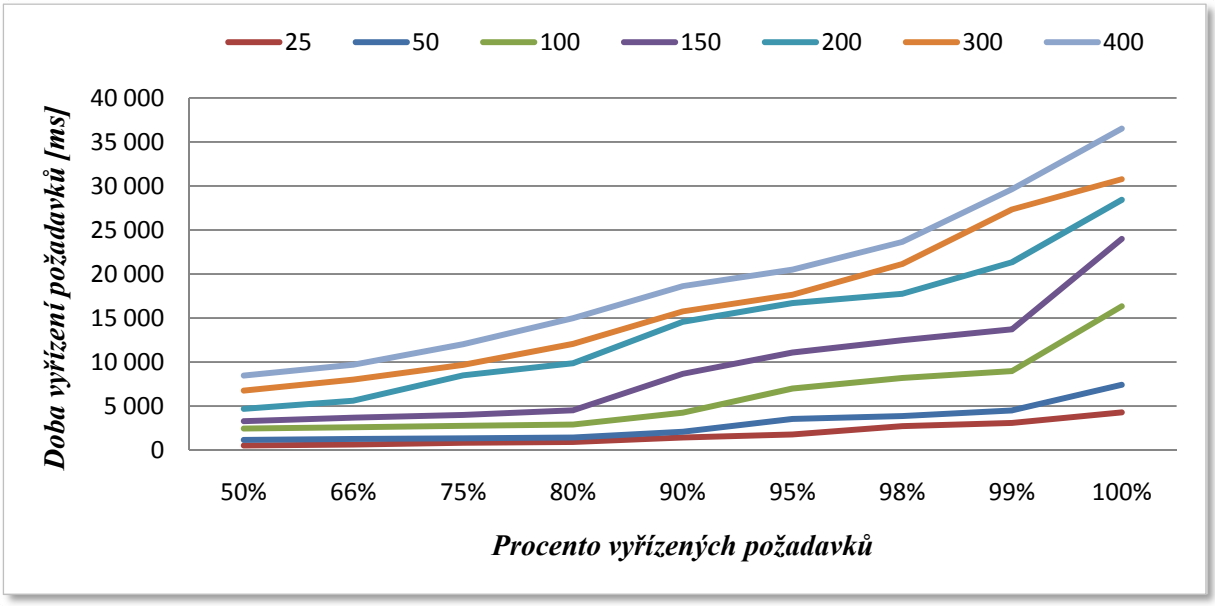
Za výchozí konfiguraci pro potřeby testů považuji aktuální nastavení hned po instalaci (výchozí nastavení Apache instalovaného z balíčků se můžou mezi některými distribucemi operačního systému lišit).

Výchozí hodnoty jednotlivých nastavení, které budu dál v této kapitole zkoumat a popisovat uvádím v následujícím seznamu:

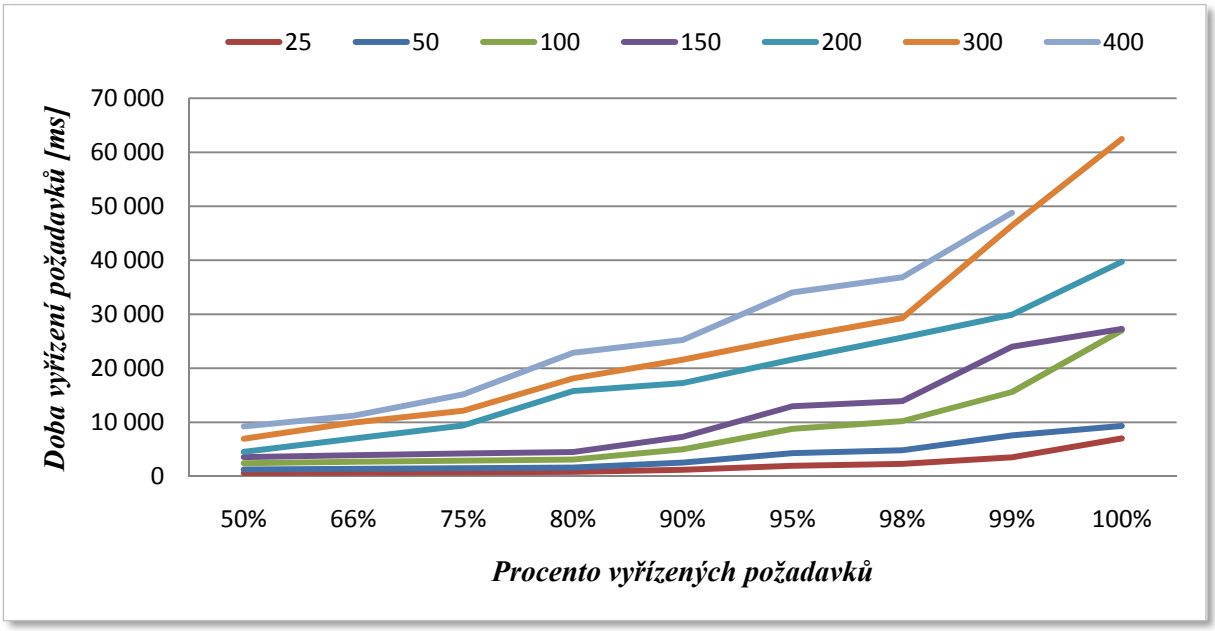
- *MaxClients* 100
- *MaxRequestsPerChild* 0
- *KeepAlive* Zapnuto
- *KeepAliveTimeout* 15 vteřin
- *MaxKeepAliveRequests* 100
- *HostnameLookups* Off
- Modul *mod_deflate* Nenačten
- Modul *mod_rewrite* Nenačten
- Direktiva *AllowOverride* None

Grafy na následující straně ukazují výkon webového serveru ve výchozí konfiguraci všech komponent, a to jak z hlediska krátkodobé, tak dlouhodobé zátěže. Výkon je zobrazen grafem závislosti procenta vyřízených požadavků na době, za kterou bylo konkrétní procento požadavků vyřízeno.

Z hlediska krátkodobé zátěže pozorujeme, že server do stanoveného limitu na odpověď 15 vteřin stihl odpovědět při středné konkurenci 200 klientů na zhruba 90% požadavků. V případě dlouhodobé zátěže počet vyřízených požadavků u konkurence 200 klientů nedosahuje ani 80%.



Graf č. 1 *– Krátkodobá zátěž při výchozím nastavení Apache*



Graf č. 2 *– Dlouhodobá zátěž při výchozím nastavení Apache*

4.1.2 MPM - Multi-Processing Module

Pro běžné UNIX-ové systémy lze Apache použít se dvěma základními moduly, které ovlivňují rychlost a další rozšiřitelnost serveru:

- *Worker* – používá několik procesů, každý proces má několik vláken a každé vlákno obsluhuje právě jeden požadavek.
- *Prefork* – používá několik procesů, každý proces obsluhuje právě jeden požadavek (má jedno vlákno). Tento systém obsluhy je podobný Apache ve verzi 1.3.

Podle některých zdrojů je Worker výkonnější než Prefork, podle jiných je výkonnost serveru srovnatelná. Modul Prefork je však náročnější na paměťové vybavení počítače ale spolehlivější a stabilnější než Worker.

Zde bych ještě rád popsal rozdíly a doporučení použití jednotlivých MPM ve spojení s PHP. Podle oficiální dokumentace PHP není doporučeno používat jej ve spojení s MPM Worker, které vrámci jednoho procesu používá několik vláken. PHP je balík, který spojuje někdy i několik desítek externích knihoven a pro svoji korektní funkčnost potřebuje fungující operační systém, webový server a vzpomínané externí knihovny. Jestli kterákoliv uvedená součást přestane pracovat, PHP potřebuje najít cestu k identifikaci problému a rychle ho vyřešit. Jestli uděláme základní framework, nad kterým PHP pracuje, velmi komplexní a tento framework nebude mít kompletně oddělené vlákna, segmenty paměti a fungující prostředí pro každý zpracovávaný požadavek, zvyšujeme možnost na výskyt problémů. Když se i přes toto varování vývojářů rozhodneme použít vláknový model (MPM Worker) jako základ webového serveru, se kterým bude PHP pracovat, je doporučeno podívat se na nastavení PHP pomocí FastCGI, kdy PHP běží ve vlastním paměťovém prostoru.

Vzhledem k omezením, o kterých jsem psal v předchozím odstavci, jsem se rozhodl zátěžové testy na modelové aplikaci provádět na konfiguraci Apache s MPM Prefork.

4.1.3 Vytváření procesů a nastavení MPM

V následujícím seznamu uvedu základní popis direktiv sloužících k nastavení MPM Prefork:

- *StartServers* počet procesů, které budou vytvořené po startu serveru
- *MinSpareServers* minimální počet nečinných procesů
- *MaxSpareServers* maximální počet nečinných procesů
- *MaxClients* maximální počet procesů (požadavků)
- *MaxRequestsPerChild* maximální počet požadavků na proces

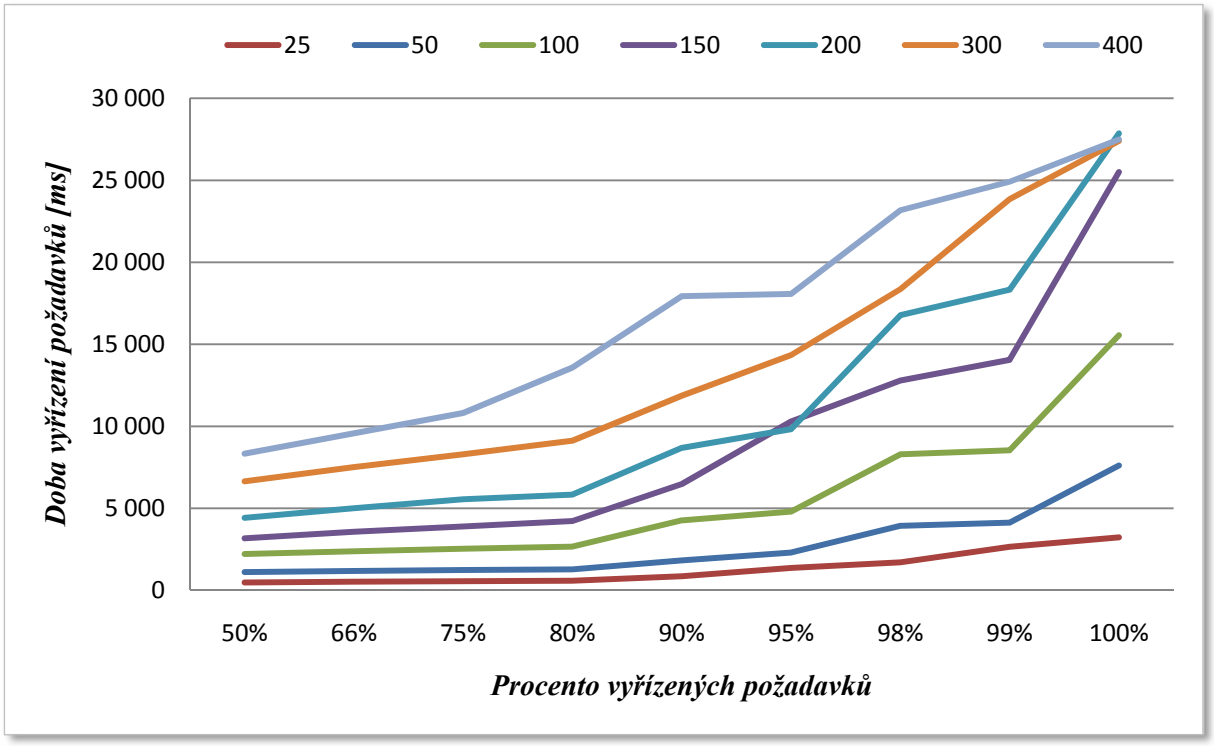
Nastavení těchto direktiv mělo v Apache serveru do verze 1.3 výrazný efekt na výkon. Apache potřeboval celkem dlouhou dobu na to, aby vytvořil dostatek procesů pro obsloužení všech příchozích požadavků. Po spuštění serveru a následném velkém počtu příchozích požadavků mohl být spuštěn maximálně jeden proces za vteřinu. Představme si situaci, kdy je počet příchozích požadavků v jeden okamžik 150 a direktiva StartServers je nastavena na hodnotu 5 (výchozí). Pro vytvoření dostatečného počtu procesů Apache potřeboval 145 vteřin. V tomto případě mělo význam určit správné nastavení direktiv StartServers, MinSpareServers a MaxSpareServers.

Od Apache verze 1.3 je však situace trochu odlišná a Apache vytváří procesy s exponenciálním růstem, přičemž začíná od jednoho. Tedy, v první vteřině vytvoří jeden proces, v další dva, pak čtyři a takto pokračuje až do okamžiku, kdy vytváří 32 procesů za vteřinu. Od této verze tedy lze konstatovat, že nastavování výše zmíněných direktiv nemá tak značný vliv na výkon systému jak tomu bylo v Apache verzi do 1.3. Jejích správným nastavením lze i tak výkon serveru vyladit.

V případě, že jsou za vteřinu vytvářeny více než 4 procesy, je do ErrorLog zaznamenána zpráva informující o této skutečnosti. Jestli se v ErrorLog objevuje mnoho těchto zpráv, pak je na místě změna nastevení direktiv MinSpareServers a MaxSpareServers.

Na záčátku jsem se zaměřil na změnu nastavení direktivy MaxClients. Tato direktiva nastavuje limit na počet současně vyřízených požadavků. Každý další požadavek na spojení bude zařazen do fronty a obsloužen až tehdy, když skončí obsluha dřivějšího požadavku a některý z existujících procesů se uvolní. Direktiva MaxClients v případě Prefork MPM určuje maximální počet procesů, který bude vytvořen pro obsluhu příchozích požadavků.

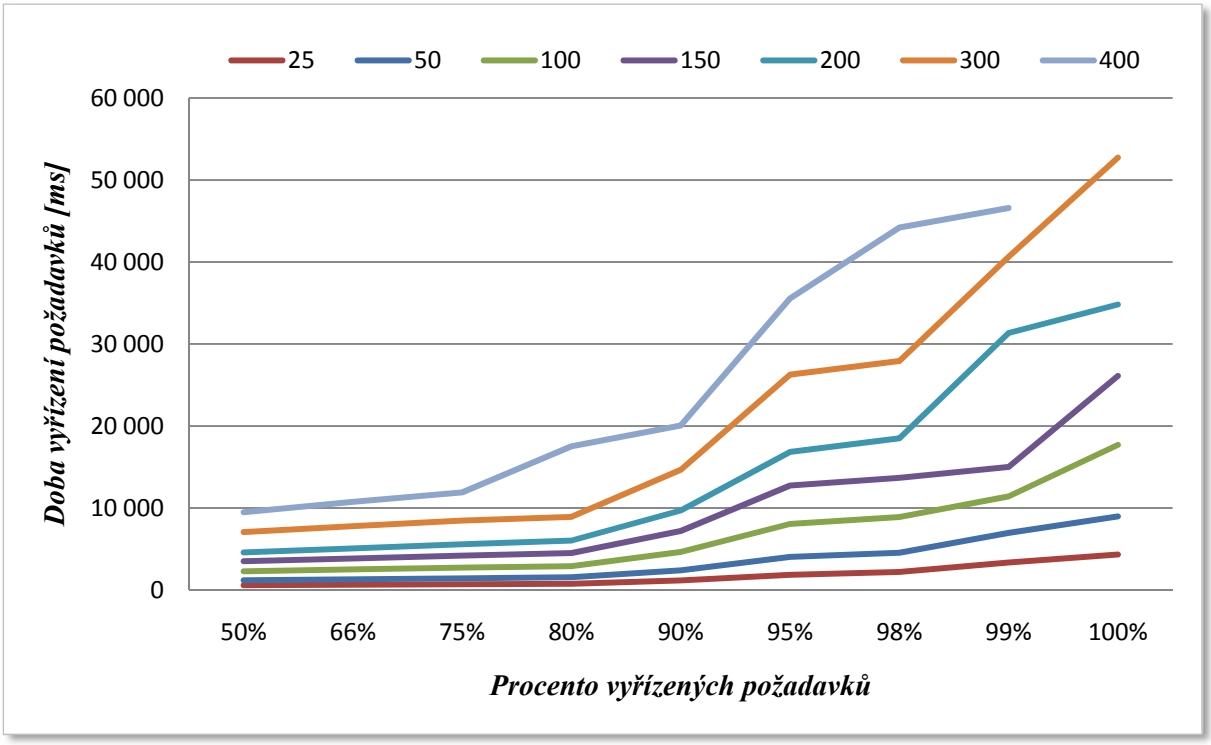
Výchozí hodnotou je 100, avšak u různých distribucí se tato hodnota může lišit. V prvním testu jsem zvýšil MaxClients na hodnotu 250. Výsledky ukazují následující grafy.



***Graf č. 3** – Apache: Krátkodobá zátěž při změně direktivy MaxClients z výchozí hodnoty na 250*

Test krátkodobé zátěže na grafu č. 3 u dříve sledované konkurence 200 klientů ukazuje, že server odpověděl ve stanovené době na téměř 97% požadavků, což je oproti výchozímu nastavení zlepšení o necelých 7%. U nejvyšší úrovně testované konkurence, 400 klientů, vyřizuje Apache 85% požadavků do 15 vteřin, což je zlepšení o 5% vůči výchozímu nastavení.

U dlouhodobé zátěže (graf č. 4 na následující straně) můžeme také sledovat zlepšení výkonu po provedené změně v konfiguraci – 4% u úrovně konkurence 200 klientů. Zlepšení výkonu pouze o 2% při dvojnásobné konkurenci je způsobené stále nízkým nastavením MaxClients v porovnání s počtem konkurentních klientů. Pokusíme se tedy zrychlit odezvu serveru i v případě vyšší konkurence a to dalším zvednutím hodnoty MaxClients.

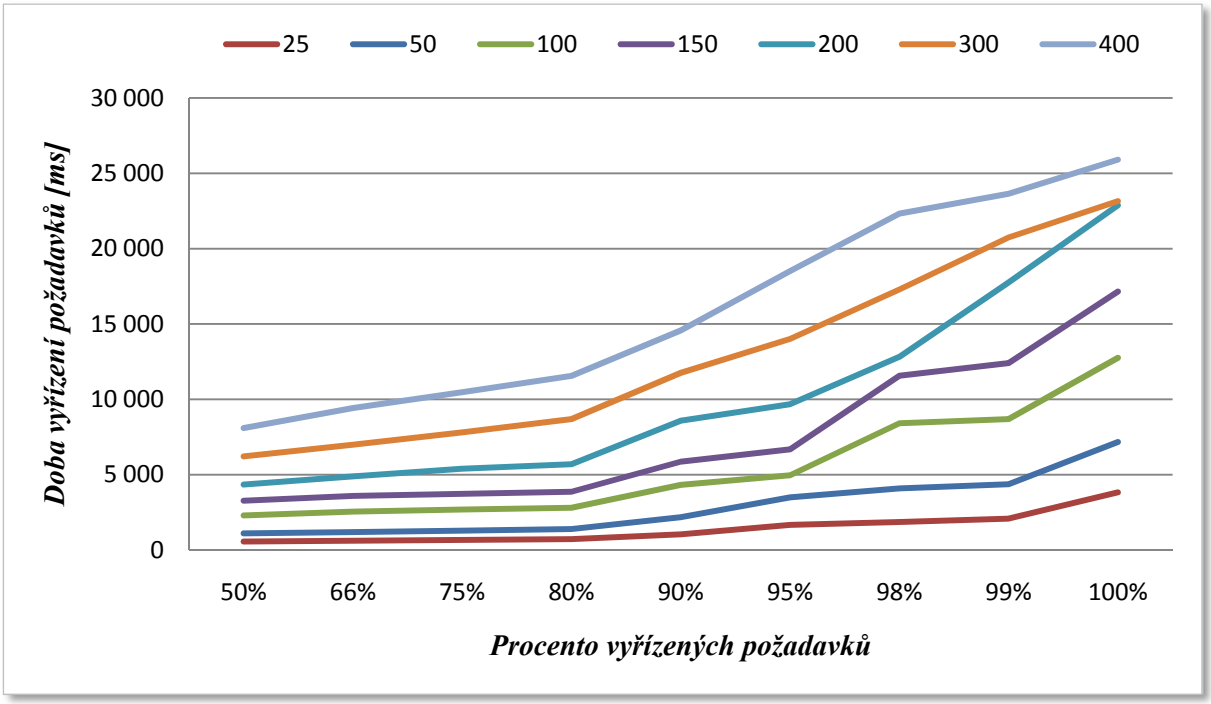


Graf č. 4 *– Apache: Dlouhodobá zátěž při změně direktivy MaxClients z výchozí hodnoty na 250.*

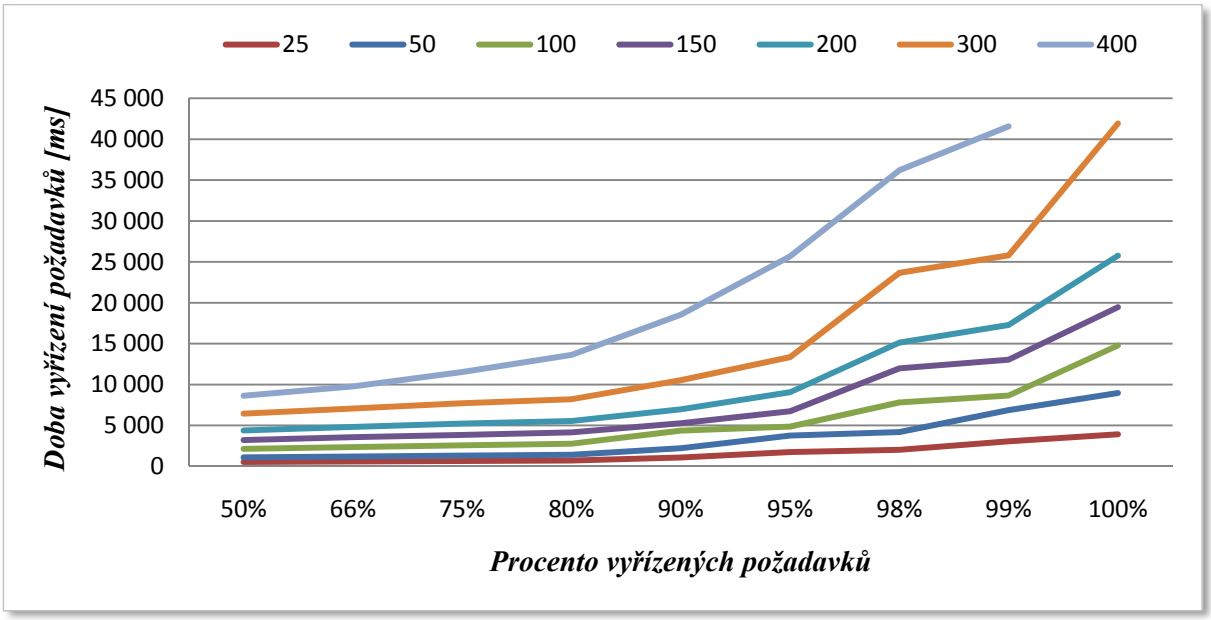
Hodnota direktivy MaxClients však nesmí přesáhnout hodnotu ServerLimit, která je ve výchozím stavu nastavena na 256. V dalším testu nastavíme MaxClients na dvojnásobek současné hodnoty a dosažené výsledky porovnáme jak s výchozí konfigurací, tak s předešlým testem. Pro nastavení MaxClients na 500 je však nutné upravit direktivu ServerLimit na hodnotu vyšší nebo rovnou novému nastavení MaxClients.

Zde bych ještě rád upozornil, že direktivu MaxClients nelze zvedat donekonečna. Každý server má omezenou operační paměť, každý proces serveru Apache zabírá určitý prostor v RAM a je třeba si uvědomit, že maximální počet vytvořených procesů vynásobený velikostí procesu Apache nesmí přesáhnout dostupnou paměť po odečtením velikostí ostatních procesů. V opačném případě by server začal využívat virtuální paměť a výkon by rapidně klesl. Proto před každým navýšením direktivy MaxClients je potřeba zvážit nové nastavení. Jako nejlepší postup je zjištění průměrné velikosti procesu Apache a vydělení dostupné operační paměti touto hodnotou.

V dalším testu jsem zvedl nastavení MaxClients na hodnotu 500. Výsledek testu krátkodobé zátěže (graf č. 5) při konkurenci 200 klientů ukazuje nepatrné zlepšení o zhruba 0,5% oproti předchozímu testu, o více než 8% oproti výchozímu nastavení. U dvojnásobné konkurence došlo ke zlepšení o 5% vůči předchozímu testu, čímž se splnil můj předpoklad, že dalším navýšením MaxClients dosáhnu lepších výsledků pro vyšší konkurenci. Výsledky testu dlouhodobé zátěže na grafu č. 6 ukazují další zvýšení výkonu. U 200 současně připojených klientů stíhá server vyřídit 98% požadavků do stanoveného limitu, což je zlepšení o 3% vůči předchozímu testu. V případě 400 současně připojených klientů došlo ke zlepšení o 7% oproti výchozímu stavu – na současných 83%.



Graf č. 5 *– Apache: Krátkodobá zátěž při změně direktivy MaxClients na hodnotu 500.*



Graf č. 6 - *Apache: Dlouhodobá zátěž při změně direktivy MaxClients na hodnotu 500.*

4.1.4 Keepalive a trvalé TCP spojení protokolu HTTP/1.1

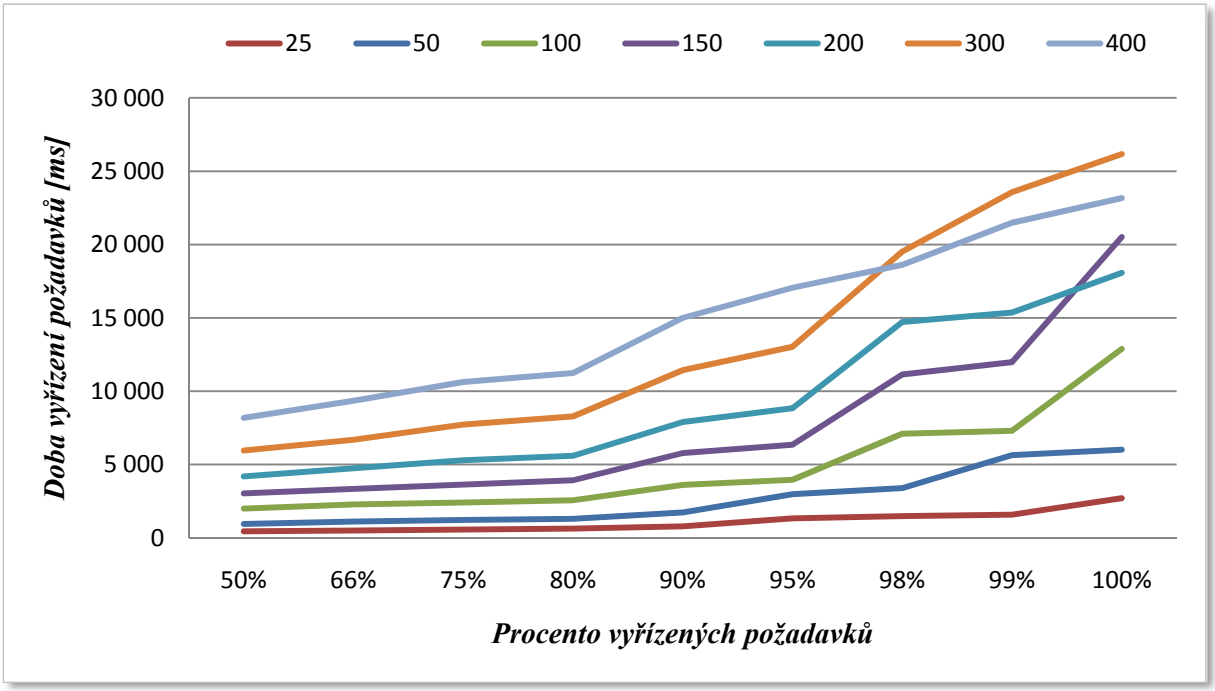
Direktiva KeepAlive spolu s trvalým spojením definovaným v HTTP/1.1 poskytuje dlouhodobé HTTP relace, kdy je přes jedno TCP spojení posíláno několik požadavků. V některých případech je možné pomocí KeepAlive snížit čas potřebný na vyřízení požadavků až o 50%.

Pro klienty používající HTTP/1.0 bude KeepAlive použito pouze v případě, kdy to klient výslovně vyžaduje. Naopak, pro klienty používající HTTP/1.1 jsou trvalé TCP spojení výchozí, pokud není specifikováno jinak.

S nastavením KeepAlive souvisí několik dalších direktiv v konfiguraci Apache. Jedná se především o direktivu KeepAliveTimeout, jejíž hodnota určuje počet sekund, kolik bude server čekat na další požadavky přes toto spojení. Když tento čas vyprší, spojení je ukončeno. Vysoká hodnota KeepAliveTimeout může způsobit problémy s výkonem na vytížených serverech. Čím je tato hodnota vyšší, tím víc procesů bude čekat na další požadavky, i když bude spojení nečinné. V kontextu virtuálních hostitelů nastavených na serveru se vždy bere v úvahu nastavení KeepAliveTimeout u prvního (výchozího) virtuálního hostitele. Ostatní nastavení KeepAliveTimeout jsou ignorovány. Na produkčních serverech bych z hlediska výkonu zatíženého serveru doporučil nastavení této hodnoty na úroveň 3 až 5 vteřin. Do tohoto intervalu se téměř vždy vyžádají všechny související soubory, tudíž pro získaní kompletního obsahu postačí jedno TCP spojení a po vyřízení všech požadavků proces včas zanikne a uvolní zabrané prostředky pro další procesy.

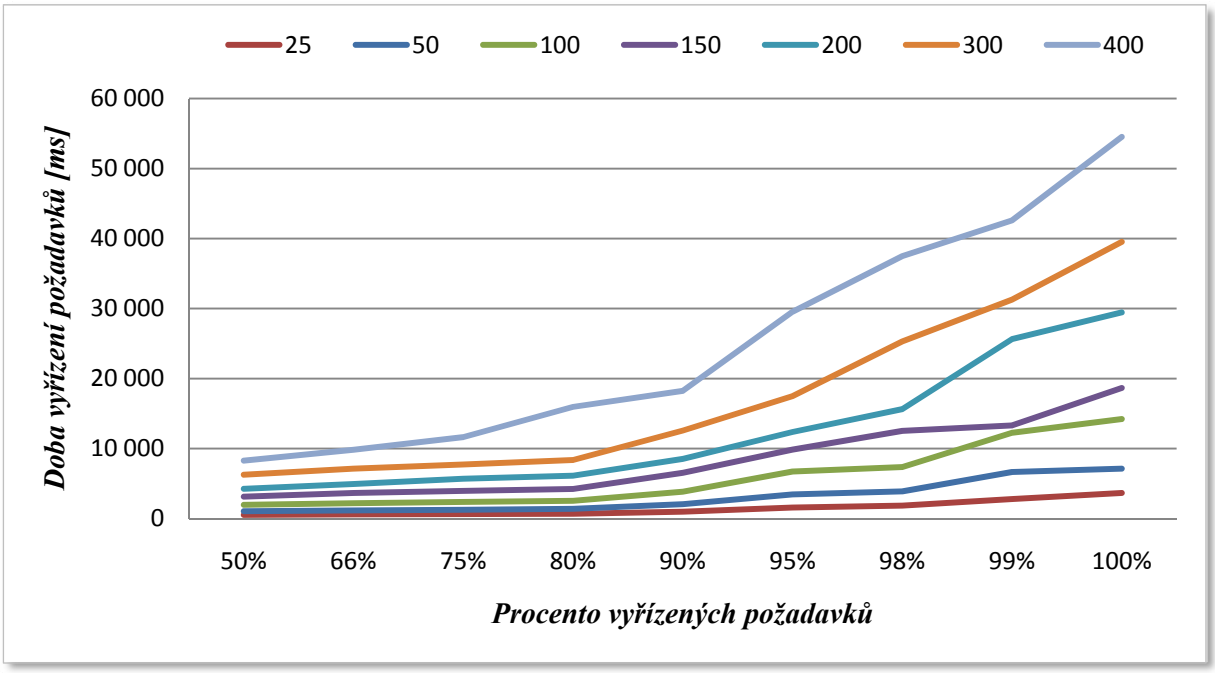
Za zmínku stojí ještě direktiva MaxKeepAliveRequests, která limituje počet požadavků vyřízených přes jedno spojení. Hodnota 0 znamená neomezený počet. Pro nejvyšší výkon je doporučené nastavit vysokou hodnotu, kolem 500 požadavků. Výchozí hodnota je 100.

V testu výchozí konfigurace, který je uveden v úvodu této kapitoly, jsem vypnul KeepAlive a to z důvodu, aby jsme viděli změnu ve výkonu, kterou tato funkce způsobí. Nyní si ukážeme test provedený pro krátkodobou i dlouhodobou zátěž ve výchozí konfiguraci Apache se zapnutým KeepAlive. KeepAliveTimeout byl v tomto případě nastaven na 5 vteřin, MaxKeepAliveRequests na hodnotu 500 požadavků. Zde jsou výsledky.



***Graf č. 7** – Apache: Krátkodobá zátěž po zapnutí KeepAlive*

Z výsledků testů je patrné značné zlepšení oproti konfiguraci bez zapnuté funkce KeepAlive. V případě konkurence 200 klientů je výkon z hlediska krátkodobé zátěže vyšší o téměř 7%, u dvojnásobné konkurence dokonce o 9%.



Graf č. 8 *– Apache: Dlouhodobá zátěž po zapnutí KeepAlive*

Testy dlouhodobé zátěže ukazují zlepšení o 8% u konkurence 200 klientů, u konkurence 400 klientů zhruba o 5%. Z testů je zřejmé, že zapnutí trvalých TCP spojení použitím KeepAlive zvýšilo výkon serveru Apache.

4.1.5 HostnameLookups a vše kolem DNS

Direktiva HostnameLookups povoluje použití DNS pro překlad IP adres za účelem logování názvů klientů a umístění tohoto názvu do REMOTE_HOST. Možné hodnoty nastavení jsou On, Off nebo Double. Hodnota Double značí provádění dvojitého reverzního DNS vyhledání. V praxi to znamená provedení dopředního vyhledání hned po tom, co je vrácen výsledek zpětného vyhledání, přičemž souhlasit s původní IP adresou musí minimálně jedna IP adresa vrácena dopředním vyhledáním.

Je důležité upozornit na to, že bez ohledu na nastavení direktivy HostnameLookups, je v případě, kdy je přístup kontrolován na základě hostitelského jména použitím modulu mod_access, prováděné dvojité reverzní vyhledání v DNS. Výsledek vyhledání však není globálně přístupný, pokud není direktiva HostnameLookups nastavena na hodnotu Double. V praxi to znamená, že když přijde požadavek na objekt, ke kterému je kontrolován přístup na základě hostitelského jména, tak bez ohledu na výsledek dvojitého reverzního vyhledání v DNS je do REMOTE_HOST umístěn výsledek dalšího zpětného vyhledání. Z toho plyne doporučení nepoužívat omezení přístupu pomocí hostitelských jmen, ale pomocí IP adres, kde nedochází ke zbytečnému překladu v DNS, což způsobuje spoždění.

Ve výchozím stavu je z důvodu rychlejšího vyřízení požadavků tato direktiva nastavena na Off, tedy vypnuto. U produkčních serverů toto nastavení důrazně doporučuji, protože vyhledání v DNS u

každého klienta výrazně prodlužuje dobu potřebnou pro vyřízení požadavků. Pro následný překlad IP adres v log souborech slouží nástroj *logresolve*.

Ve všech zde uvedených testech je direktiva HostnameLookups nastavena na Off. Modul mod_access používán není a konfigurace Apache neobsahuje ani jeden případ, kdy by byl kontrolován přístup pomocí hostitelského jména. Žádný z testů tedy není ovlivněn dobou potřebnou pro vyřízení dotazů na servery DNS.

Testy pro zjištění změny ve výkonu v případě zapnutí HostnameLookups jsem na testovací aplikaci neprováděl z důvodu, že všechny požadavky na server jsou směrovány z jednoho PC a proto by se snížení výkonu neprojevilo takovým způsobem, jako na produkčních serverech, kde jsou zdroje požadavků různorodé.

4.1.6 Direktiva AllowOverride a používání souborů .htaccess

Direktiva AllowOverride specifikuje, které nastavení, definované v konfiguraci Apache, můžou být přepsány nastaveními, které jsou definované v souborech .htaccess (soubory se můžou jmenovat i jinak, jejich název specifikuje direktiva AccessFileName). Direktiva AllowOverride může být specifikována pouze v kontextu <Directory>.

Nebudu zde popisovat všechny možné varianty, které lze direktivou AllowOverride dosáhnout, ale pouze dvě krajní – None (vypnuto), All (zapnuto) a jejich vliv na výkon. V případě nastavení AllowOverride na hodnotu None jsou soubory .htaccess kompletně ignorovány. Server v tomto případě ani nehledá tyto soubory, zda vůbec existují. Opačným případem je nastavení na hodnotu All, kdy obsahem .htaccess může být jákákoliv direktiva a tato přepíše dřivější nastavení serveru. V tomto případě bude Apache při každém požadavku na určitý soubor hledat .htaccess a zde je velmi důležité, v kontektu jakého adresáře byla direktiva AllowOverride specifikována. Platí pravidlo čím špecifičtější nastavení, tím méně pokusů o nalezení souboru .htaccess. Pro názornější vysvětlení uvedu příklad. Jestli je kořenovým adresářem virutálního hostitele adresář například /var/www/html/domena.cz/htdocs a direktiva AllowOverride je v konfiguraci nastavena v kontextu <Directory />, tedy globálně pro všechny adresáře, při požadavku na soubor index.html umístěný v kořenové složce hostitele se Apache pokusí otevřít soubory /.htaccess, /var/.htaccess, /var/www/.htaccess atd., to znamená několik zbytečných pokusů o otevření souborů, což se při vysoké zátěži (velkém počtu požadavků) projeví na výkonu negativním způsobem. Když je tedy nutné nastavit hodnotu direktivy AllowOverride na cokoliv jiného než None, vždy doporučuji toto nastavení provést na úrovni virtuálního hostitele, tedy v našem příkladu v kontextu <Directory /var/www/html/domena.cz/htdocs>. V takovém případě se Apache pokusí rovnou o otevření souboru .htaccess nacházejícího se v kořenové složce hostitele a dopad na výkon serveru je daleko menší.

4.1.7 Přepisování URL a použití modulu mod_rewrite

Přepisování URL je u dnešních webových serverů poskytujících dynamický obsah často používaná funkce. Umožňuje přepis URL přístupných pro vyhledávače do formy, jakou jsou navržené v konkrétní aplikaci. V praxi to znamená, že URL, které zadáva uživatel, nebo indexuje vyhledávač, vypadá například http://www.domena.cz/o-spolecnosti/ a toto URL je webovým

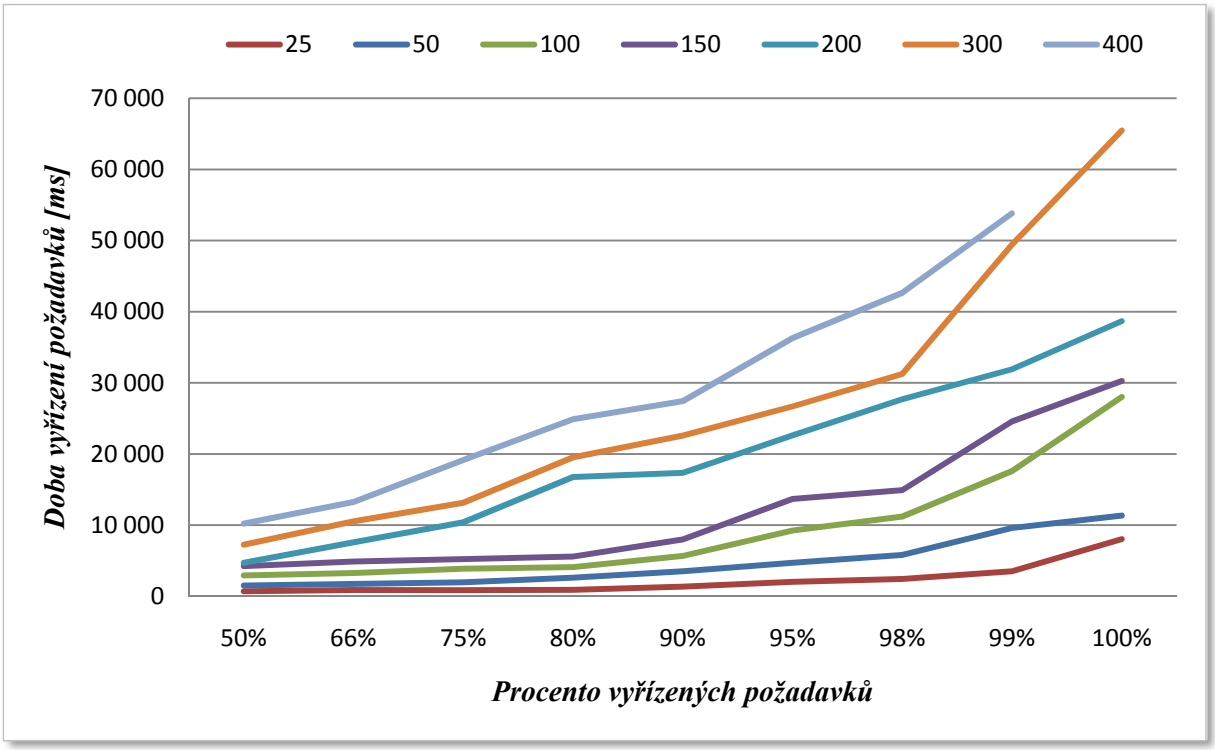
serverem přepsáno na http://www.domena.cz/obsah.php?stranka=o-spolecnosti. Z hlediska vyhledáváčů a jednoduchosti URL je lepší první varianta. Ze strany serveru a zpracování obsahu zase ta druhá.

Přepisování URL však způsobuje určitou zátěž na straně serveru a tudíž zpomalení výkonu. Úroveň zpomalení je dána složitostí a počtem jednotlivých pravidel pro vykonání konkrétního přepsání.

Modul mod_rewrite Apache serveru je mocným nástrojem pro tento účel. O tom, jak korektně vytvářet efektivní přepisovací pravidla by se dala napsat samostatná práce. To ale není mým cílem, a proto zde uvedu pouze dopad na výkon serveru, když je modul mod_rewrite povolen a použit. Pro účely testování si proto vytvoříme jednoduché pravidlo. Vyzkoušíme, jaký to bude mít dopad při dlouhodobé zátěži, kdy je větší pravděpodobnost, že se zpomalení projeví.

Modul mod_rewrite se nastavuje na úrovni virtuálního hostitele nebo v souborech .htaccess. První varianta je z hlediska výkonu serveru výhodnější, protože není potřeba povolit používání souborů .htaccess a tak zaniká negativní dopad jejich použití na výkon (direktiva AllowOverride popisována v předchozí kapitole). Řada webových serverů však poskytuje webhostingové služby pro různé uživatele, kteří nemají přístup k nastavování virtuálních hostitelů a chtějí používat modul mod_rewrite. V tomto případě je potřeba bezpodmínečně povolit používání souborů .htaccess.

Zde už uvádím výsledky provedených zátěžových testů a jejich popis.



***Graf č. 9** – Apache: Dlouhodobá zátěž a spomalení způsobené modulem mod_rewrite*

Z výsledků testu je patrný pokles výkonu oproti výchozí konfiguraci. Použitím jednoduchého pravidla pro přepisování URL klesl výkon u střední konkurence 200 klientů asi o 5%, u konkurence

400 klientů o téměř 9%. Je zřejmé, že použitím sady přepisovacích pravidel, které obsahují složité regulární výrazy může dojít ke značnému poklesu výkonu.

4.1.8 Komprese a použití modulu mod_deflate

S vytvořením protokolu HTTP /1.1 byla uvedena nová vlastnost požadavku nazývaná Accept-Encoding. Tato vlastnost umožňuje webovému serveru zjistit, jestli klientský software podporuje příjem komprimovaných dat.

V předchozí verzi, HTTP/1.0, se data přenášela výhradně bez komprese. Když si ale uvědomíme, že hlavní úlohou webového serveru je poskytovat hypertextový obsah, hlavně HTML dokumenty, které obsahují často se opakující tagy, tak tento obsah může být snadno a velmi dobře zkomprimován běžnými algoritmy jako je například zip, bzip2 nebo gzip.

Modul mod_deflate v Apache verzi 2 tedy poskytuje kompresi HTTP/1.1 a odpovědi na příchozí požadavky jsou zpátky zasílané v komprimované formě (jestli klientský prohlížeč tuto vlastnost podporuje, jak již bylo zmíněno výše). Kompresí HTML dokumentů a ostatního textového obsahu poskytovaného serverem (kaskádové styly CSS, Javascript aj.) je možné dosáhnout 50 – 70% úsporu ve velikosti dat, která se projeví především v objemu přenesených dat a využití síťového rozhraní serveru.

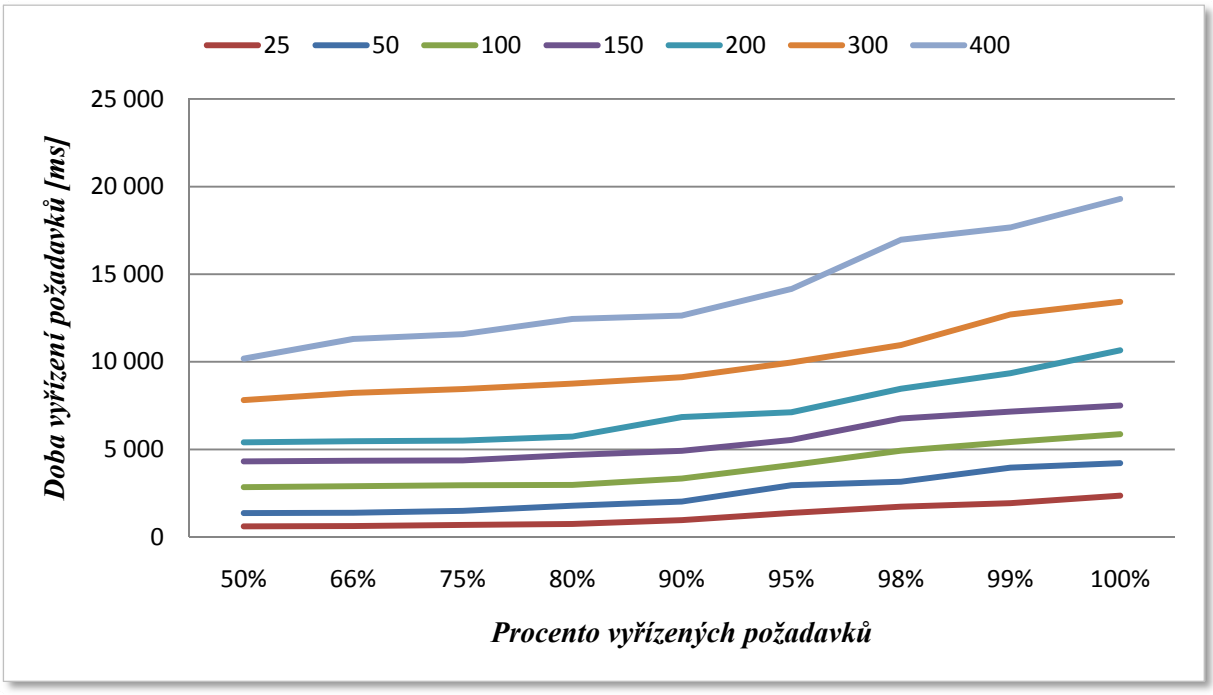
Zmenšením objemu přenesených dat nesnížíme pouze účet za serverhosting u poskytovatele služeb ale i využití síťového rozhraní. Menší účet za služby na výkoně nepřidá, no využití síťového rozhraní se na výkoně určitě projeví. Každé rozhraní má určitou propustnost, v dnešní době je to u serverů umístěných na pátěri internetu standardně 100Mbps. Když použitím modulu mod_deflate dokážeme snížit přenos dat alespoň na 50%, znamená to, že v jeden časový okamih můžeme na síťovou linku umístit dvakrát tolik dat, než bez použití komprese, a tím snižujeme oneskorení vyřízení příchozích požadavků.

Komprese však nepřináší jenom řadu pozitivních vlastností. Použitím komprese dochází k zabalení každé odpovědi na požadavek a to stojí procesorový čas. Před nakonfigurováním modulu mod_deflate je proto vhodným postupem sledovat zatížení procesoru, zejména při zvýšené zátěži webového serveru. Když i v tomto případě nejsou zdroje procesoru zcela vyčerpány, je na místě kompresi zapnout a provést potřebná měření, jestli má komprese pozitivní dopad na výkon. Většina výkonných webových serverů kompresi používá, protože běží na kvalitním hardwaru s rychlým procesorem, kde je limit síťové linky větším problémem než procesorový čas.

Vhodným nastavením je použití komprese pouze na textový obsah – HTML dokumenty, kaskádové styly CSS nebo javascript. Dalším obsahem, který obsluhuje webový server jsou obrázky, které jsou již zkomprimované (JPEG, GIF) a proto použití komprese na tento typ obsahu smysl nemá.

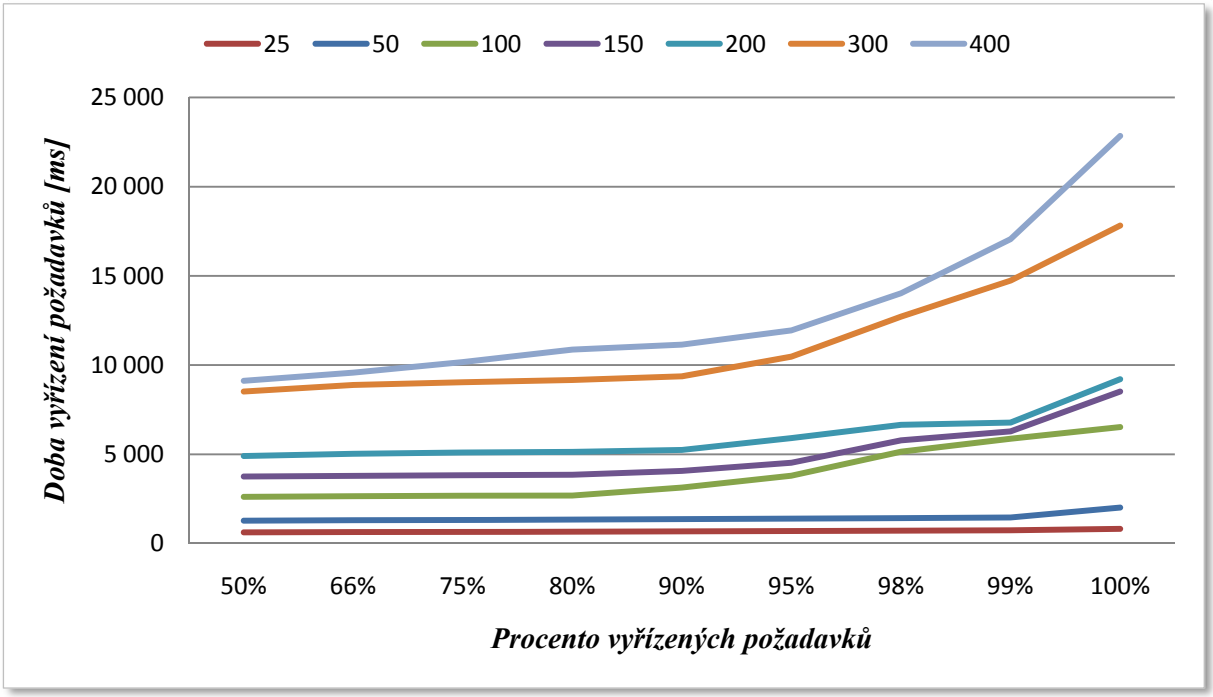
Pro test komprese na modelové aplikaci jsem modul mod_deflate nastavil přidáním direktivy *AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css application/x-javascript*.

Tato konfigurace nastavuje kompresi na výstupu z webového serveru (odpovědi na požadavky) a komprese se bude aplikovat pouze na uvedené typy dokumentů – HTML, čistý text, XML, CSS nebo javascript. Zde jsou výsledky měření.



Graf č. 10 *– Apache: Krátkodobá zátěž s nakonfigurovaným modulem mod_deflate*

Zapnutím a nakonfigurováním modulu mod_deflate jsme získali značné zlepšení výkonu serveru Apache. Pro porovnání s výchozím stavem se v případě konkurence 300 klientů vyřídí 100% požadavků s maximálním spožděním 13,5 vteřiny.



Graf č. 11 *– Apache: Dlouhodobá zátěž s nakonfigurovaným modulem mod_deflate*

Test dlouhodobé zátěže ukazuje pozitivní výsledky. Zatím co se nám ani v jenom z předchozích testů nepovedlo vyřídit při dlouhodobé zátěži a konkurenci 200 klientů 100% požadavků,

použitím komprese a modulu mod_deflate se to povedlo dokonce s maximálním spožděním do 10 vteřin. Křivky u konkurence 300 a 400 klientů sice překračují stanovený limit 15 vteřin, procentuální vyřízení požadavků je však ve srovnání s výchozím stavem zcela uspokojivé.

4.1.9 Doporučující nastavení

Nyní si zhrneme postup, jakým jsem prováděl testování výkonu webového serveru, zopakujeme si doporučení pro jednotlivé nastavení a ukážu výsledný test po vyladění výkonu a porovnání s výchozím stavem.

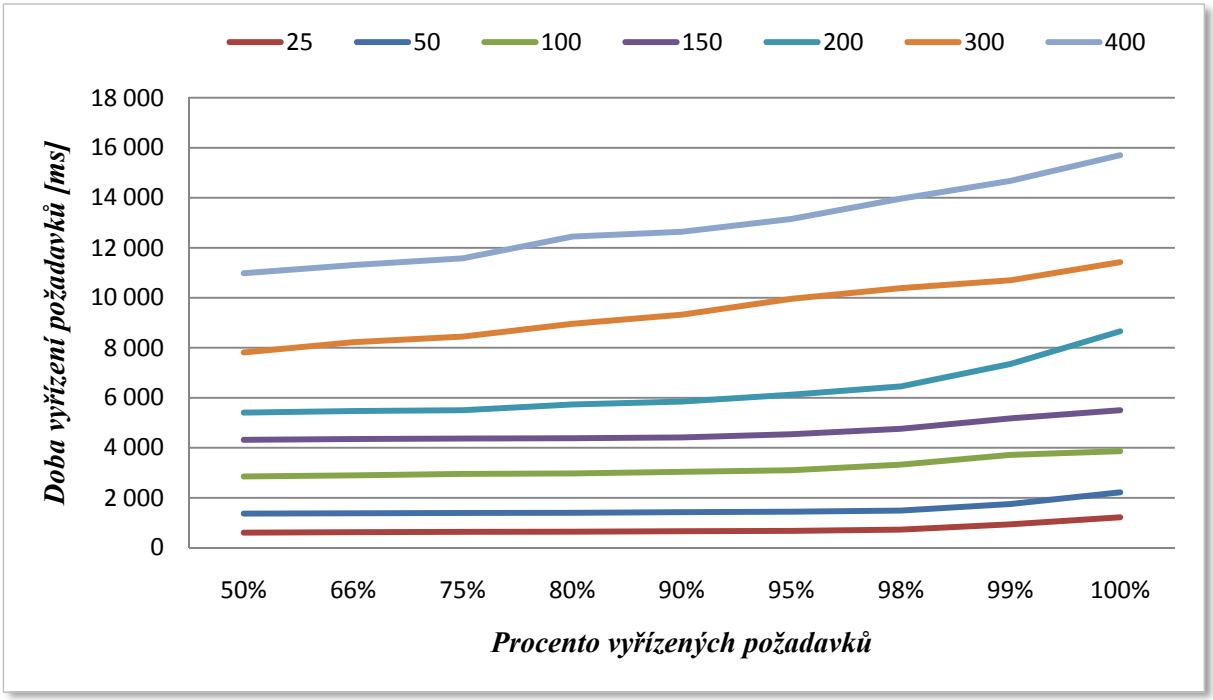
Jako první jsem měnil nastavení MPM Prefork, konkrétně direktivu MaxClients. Zde je důležité pamatovat na postup výpočtu hodnoty MaxClients. Příliš vysoká hodnota této direktivy totiž může způsobit snížení výkonu webového serveru v důsledku použití virtuální paměti nebo dokonce nestabilitu systému. Toto nastavení je proto potřeba měnit s přihlédnutím na tyto skutečnosti. V uvedených testech jsem změnil nastavení MaxClients z výchozí hodnoty až na 500.

Dále jsem sledoval změnu výkonu Apache po zapnutí KeepAlive – trvalých TCP spojení. Zde došlo ke zvýšení výkonu o 7 - 9% u krátkodobých testů a o 5 – 8% u dlouhodobých.

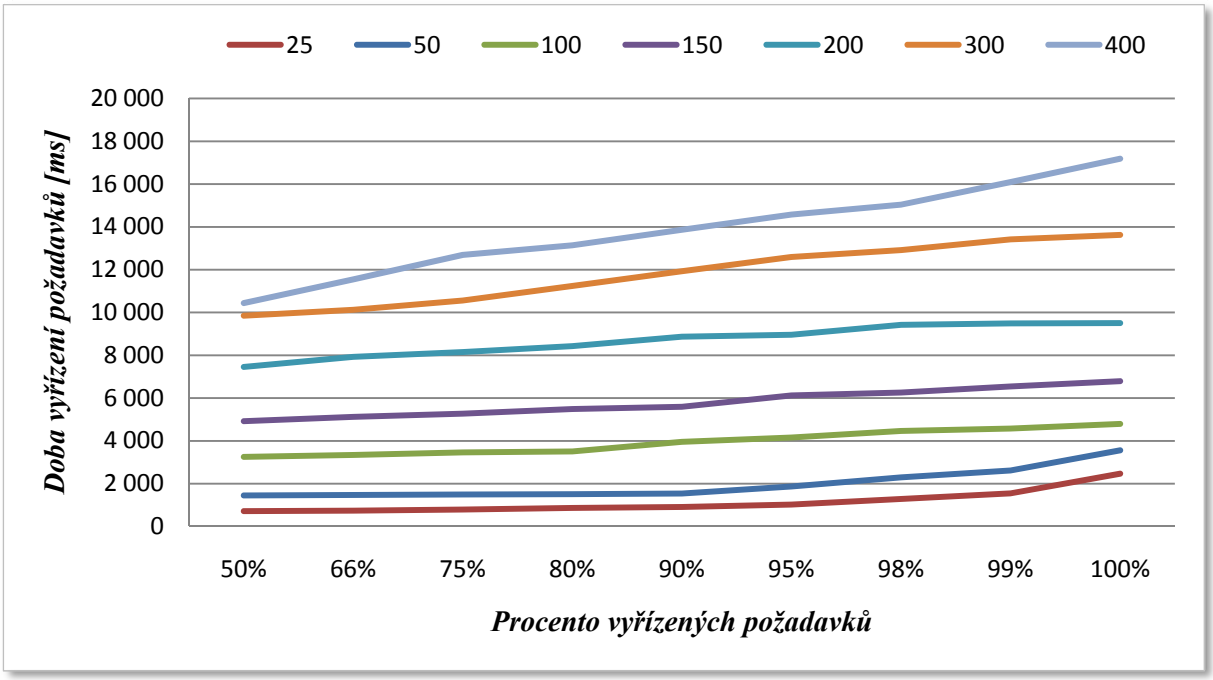
Nejlepší výsledky byly dosaženy při použití komprese a modulu mod_deflate. Obsah testovací aplikace je čistý text a tak úroveň dosažené komprese byla velmi vysoká – více než 70%. Ke zlepšení výkonu došlo hlavně z důvodu odlehčení síťového rozhraní, kdy byl server schopen na síť umístit několikanásobně víc paketů než tomu bylo bez použití komprese.

Test použití modulu mod_rewrite ukázal, že použití přepisování URL spomaluje výkon serveru. Použití jednoduchého pravidla pro přepsání URL způsobilo spomalení serveru zrhuba o 5%. Je tedy vhodné vyvarovat se masivnímu přepisování URL pomocí pravidel, které obsahují složité regulární výrazy. Navíc, nastavení modulu mod_rewrite doporučuji provést na úrovni virtuálního hostitele a nepovolovat soubory .htaccess použitím direktivy AllowOverride.

Zde jsou výsledky testů provedených na vyladěné konfiguraci Apache serveru.



Graf č. 12 *– Apache: Krátkodobá zátěž na vyladěném serveru*



Graf č. 13 *– Apache: Dlouhodobá zátěž na vyladěném serveru*

S přihlédnutím na výsledky testů výchozí konfigurace tyto grafy nepotřebují komentář.

4.2 MySQL

Databázový systém MySQL nabízí mnoho různých nastavení, pomocí kterých lze řídit všechny aspekty serveru. V této části popíšu nastavení ovlivňující výkon serveru globálně (bez ohledu na typ zvoleného úložiště) i nastavení ovlivňující výkon úložišť MyISAM a InnoDB.

Zde je důležité zdůraznit, že špatně navržená aplikace nebude i přes dobře vyladěný server fungovat optimálně. Nejjednodušším krokem ke zvýšení výkonu je proto optimalizace samotné aplikace.

4.2.1 Konektivita

Rychlost připojení je jednou z prvních věcí, která může ovlivnit výkon serveru. Důležitým nastavením v této oblasti je maximální velikost zpráv odesílaných serverem. Pro výchozí velikosti zprávy se používá nastavení net_buffer_length ve spolupráci s proměnnou max_allowed_packet. Nastavení proměnné max_allowed_packet na vyšší hodnotu se vyplácí v případě používání rozsáhlých textových hodnot. Nastavení net_buffer_length je vhodné udržovat nízké. Když je zpráva větší než net_buffer_length, MySQL automaticky zvedne tento limit až na úroveň max_allowed_packet.

Nesmíme zapomenout na proměnnou max_connections určující maximální počet souběžných připojení k databázovému serveru. V případě nastavení na nízkou hodnotu se v době vytížení uživatelé nemusí k serveru připojit.

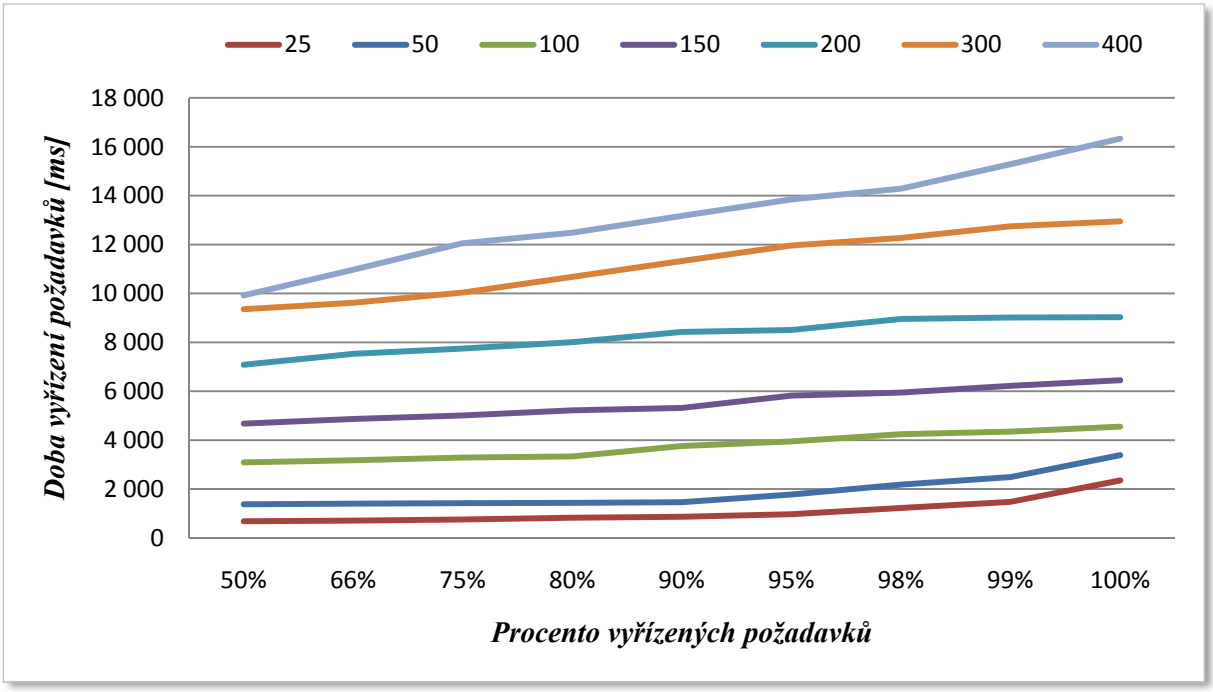
Každé existující spojení však zabíra prostředky serveru i v případě, kdy je neaktivní. Spojení je po určité době nečinnosti potřeba ukončit. Toto zabezpečuje nastavení wait_timeout. Hodnota wait_timeout však nesmí být příliš nízká, jinak by klient, který se pokusí o práci po ukončení relace nebyl schopen vykonat žádný SQL dotaz a obdržel by zprávu „Lost connection to MySQL during query“, což znamená ztrátu spojení s MySQL počas provádění dotazu.

4.2.2 Paměť

Mezipamět dotazů

Mezipaměť dotazů (anglicky query cache) je funkce, která byla uvedena v MySQL ve verzi 4.0. Jde o prostor, který slouží pro ukládání výsledků dotazů. Lze takto ukládat provedené dotazy a když by měl být v budoucnu spuštěn stejný dotaz, lze použít uložené výsledky a není znova nutné prohledávat všechny potřebné tabulky. Vzhledem k tomu, že přístup do paměti je vždy rychlejší než přístup k disku, nalezení výsledků dotazů v cache znamená značné zvýšení výkonu. Tato funkce je velmi užitečná v prostředí, kde jsou tabulky, které se často nemění a pro které server přijímá stejné SQL dotazy. Vyrovnávací paměť pro výsledky dotazů nikdy nevrací neaktuální data, protože při změně tabulky jsou relevantní data ve vyrovnávací paměti zrušena.

Ve výchozím nastavení vyrovnávací paměť funguje v režimu, kdy ukládá všechny dotazy. Pomocí parametru query_cache_type nastaveného na hodnotu DEMAND lze upravit režim tak, aby ukládal pouze výsledky dotazů, ve kterých bude uvedeno SQL_CACHE. Naopak, ve výchozím režimu lze do dotazu přidat SQL_NO_CACHE čímž dosáhneme to, že výsledek dotazu nebude do vyrovnávací paměti uložen. Velikost mezipaměti dotazů se nastavuje parametrem query_cache_size. Velikost jednotlivých záznamů (výsledků dotazů), které mají být ukládány do cache lze limitovat nastavením proměnné query_cache_limit.



Graf č. 14 *– Porovnání výkonu po aktivaci mezipaměti dotazů a jejím použití*

V závěrečném výkonnostním testu serveru Apache se mezipaměť dotazů nepoužívala. Pro stejné nastevení Apache jsem povolil její použití a výsledky testu demonstruje graf č. 14 – došlo ke zlepšení o zhruba 4%. Protože zde již dochází k obsloužení téměř všech požadavků, procentuální zlepšení výkonu jsem vyjádřil na základě porovnání časů, za které server odpověděl.

Počet případů, kdy byla MySQL schopna číst z mezipaměti dotazů nalezneme ve stavové proměnné qcache_hits. V uvedeném testu bylo MySQL schopné najít relevantní výsledky v cache ve 20% případů. Z uvedeného zlepšení výkonu lze analogicky odvodit vliv tohoto parametru na tabulky, nad kterými jsou často prováděny stejné SQL dotazy.

Počet otevřených tabulek

Dalším parametrem, který má výrazný vliv na výkon serveru je počet otevřených tabulek. Parametr table_cache umožňuje nastavit, kolik tabulek může být najednou otevřeno pro všechny klienty, kteří pracují s MySQL. Příliš nízké nastavení znamená časté provádění operací pro otevření a zavření tabulky, příliš vysoká hodnota znamena mrhání prostředky operačního systému.

MySQL obsahuje užitečný nástroj pro korektní nastavení tohoto parametru. Tím je stavová proměnná opened_tables. Vysoký počet znamená, že MySQL neustále provádí náročné operace s tabulkami. Vysokou hodnotou mám zde na mysli hodnotu považovanou za vysokou vzhledem k době běhu serveru. Nejlepším doporučením pro nastavení této hodnoty je její pomalé zvyšování a sledování stavové proměnné opened_tables.

Test vlivu tohoto parametru na výkon nejsem schopen provést na modelové aplikaci z důvodu, že v databázi je pouze jedna tabulka. Výsledky takového testu by nemohly být považovány za korektní.

4.2.3 Úložiště MyISAM

Parametrem značně ovlivňujícím výkon MyISAM tabulek je cache klíčů, což je paměť uchovávající informace o indexech tabulek. Hlavní jednotkou tohoto úložiště je blok, jenž může obsahovat podrobné informace o indexech jedné nebo í více tabulek. Při práci s indexovanými datami v konkrétní tabulce se MySQL nejdříve podíva do cache. Pokud cache obsahuje relevantní informace, MySQL provede extrémně rychlé čtení z (nebo zápis do) cache klíčů. Jestli tedy dojde například ke změně určité informace v indexovaném sloupci, je aktualizován blok cache klíčů. Až bude nutné tento aktualizovaný blok z cache odstranit, například za účelem uvolnění místa pro nový požadavek, zapíše se aktualizovaný blok zpátky na disk a tím se informace trvale uloží. Pro zjištění, který blok se má v případě nedostatku místa pro nový požadavek z cache klíčů odstranit slouží časová informace uložena u každého bloku. Ta umožňuje MySQL zjistit, které bloky jsou v cache klíčů nejdéle a které má odstranit za účelem získání volného místa pro nové bloky.

Nejdůležitejším nastavením ovlivňujícím cache klíčů je key_buffer_size, tedy velikost samotné cache. Nastavení na příliš nízkou hodnotu znamená, že nezískáte plnou výhodu cache klíčů. Naopak, příliš vysoké hodnoty plýtvají drahocennými prostředky serveru. Pro začátek se doporučuje nastavení na úrovni 5 – 10% dostupné paměti počítače, v závislosti na velikost požadavků InnoDB na paměť

lze přidělit i více. Toto nastavení však neberte jako obecné doporučení. Jak bylo již řečeno, každý systém je unikátní a stanovit stejnou hodnotu pro každý z nich nemůže přinést stejné výsledky. Dobrým postupem je začínat s nízkými hodnotami a ty postupně zvyšovat podle zatížení serveru.

4.2.4 Úložiště InnoDB

Důležitou roli při zvyšování výkonu databáze hrají různé typy vyrovnávacích pamětí a algoritmy pro práci s nimi. Jinak tomu není ani u úložiště InnoDB. Interními strukturami těchto bufferů určujících, které informace jsou do nich ukládany a jak jsou synchronizovány s diskem se zde zabývat nebudu. Pro nás je především důležité znát jejich vliv na výkon.

InnoDB například nastavuje malou část bufferů stranou. Do této části bufferů pak vkládá určité informace o změnách za účelem snížení počtu diskových operací. Tyto změny jsou pak hromadně zapsány na disk, což je mnohem efektivnější.

Dalším faktorem ovlivňujícím výkon je čtení napřed. InnoDB při zjištění konzistentního sekvenčného čtení z určité tabulky začne načítat dopředu další data do prostoru bufferů a očekává, že tyto data budou brzo požadovány a požadavek na tyto data bude možné rychleji obsloužit.

Před nastavením hodnoty proměnné innodb_buffer_pool_size určující velikost paměti alokované pro prostor bufferů je důležité zvážit několik okolností. Vyhrazené MySQL servery mají většinou alokované více paměti než servery používané na různé úlohy, no i v tomto případě může vysoká hodnota způsobit zpomalení celého systému. Databázový server, který provádí více zápisů než čtení ma zcela jiné nároky na velikost a nastavení bufferů než u serverů, kde je poměr čtení a zápisu opačný.

InnoDB používá na rozdíl od MyISAM cache pro indexy i data současně. V této situaci je potřeba buffery nastavit tak, aby pravidelně docházelo k sychronizaci s diskem. Jinak budou data v případě neočekávaného pádu systému ztracena.

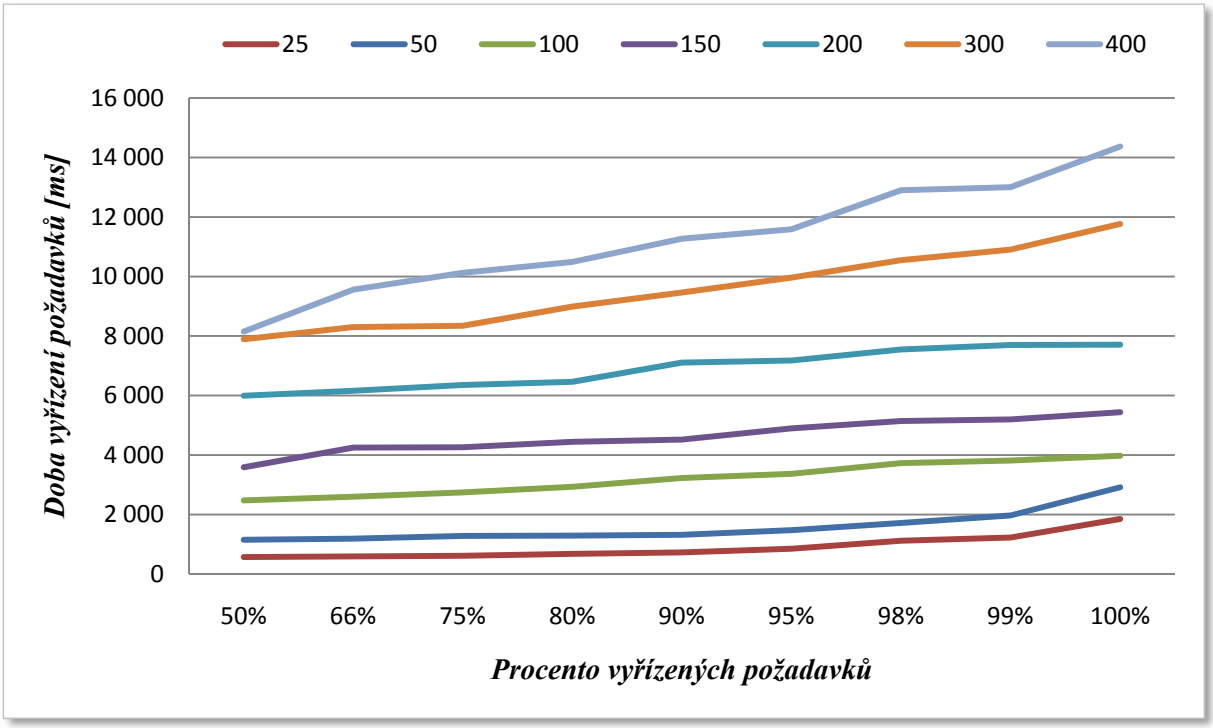
4.3 PHP

4.3.1 Cache PHP

Nejlepší možností pro zvýšení výkonu PHP je použití cache. Výchozí instalace PHP vyžaduje před provedením určitého skriptu režii spojenou s přístupem, načtením, analýzou a zkompilováním. Tyto kroky jsou prováděny především kvůli možnosti změny tohoto skriptu, avšak k tomu ve většině případů nedochází tak často.

Použití cache v PHP řeší řada produktů, komerčních i open-source. Použitím cache lze výkon serveru zvýšit až o 40%.

Nyní ukážu test provedený na vyladěné konfiguraci Apache s použitím APC (Alternative PHP Cache).



Graf č. 15 *– Porovnání výkonu PHP bez použití cache a s použitím APC*

Výsledky testu s použitím APC ukazují zlepšení o 15% - 17%. Menší zvýšení výkonu je způsobeno jednoduchostí aplikace, kterou jsme ukládali do cache. Jednoduchá aplikace totiž nezpůsobuje takové zpoždění, aby byl rozdíl ve výkonu takový, jak by možná někdo očekával.

4.4 Nastavení OS

Výkon Apache, MySQL i PHP je na linuxových systémech závislý na funkcích jádra operačního systému. Absence funkcí, které jsou dostupné v nejnovějších verzích, může mít negativní dopad na výkon. Mezi vylepšení jádra patří například:

- podpora více vláken (nemá vliv na výkon Apache v konfiguraci MPM Prefork)
- symetrický multiprocessing (nelze testovat při použití uvedené konfigurace serveru)

Verze jádra systému, na kterém probíhalo testování byla 2.6.18.

Negativní vliv na výkon operačního systému a tudíž celého serveru může mít i nainstalování nepotřebných balíčků a služeb, které i když nejsou využívané, běží a zabírají systémové prostředky. Nejlepším doporučením je tedy nainstalovat pouze nezbytně nutný software.

5 Testování na produkčních serverech

Po testech provedených na testovací aplikaci vyzkouším aplikovat nastavení na reálné aplikace běžící na produkčních servrech. Každý produkční server již má vyladěn výkon alespoň nějakým způsobem oproti výchozí konfiguraci. Já jsem vybral několik serverů, na kterých běží standardní weby – žádné specifické řešení, pro které by byl webový server nebo jiná součást speciálně vyladěná nebo nastavená.

Testy budu provádět na dvou serverech společnosti ViaAurea, s.r.o., pro kterou pracuji, z nichž jeden je SunFire X2100 a jeden SunFire X2100 M2.

5.1 www.dreamlife.cz

Jako první test vyzkouším výkon serveru, kde běží e-magazín životního stylu, úspěchu a dosahování osobních cílů – DreamLife.cz. Nejdřív si však popíšeme současnou hardwarovou a softwarovou konfiguraci serveru.

5.1.1 Hardware

Projekt DreamLife.cz běží na serveru Sun Fire X2100 s jedním procesorem AMD Opteron 146 o frekvencí 2GHz. Server má 2GB operační paměti RAM a 2x 250GB SATA disky v RAID1.

5.1.2 Software

Na serveru běží Apache ve verzi 2.0.54 v 32-bitovém režimu s MPM Prefork. Zde uvádím hodnoty sledovaných direktiv a nastavení:

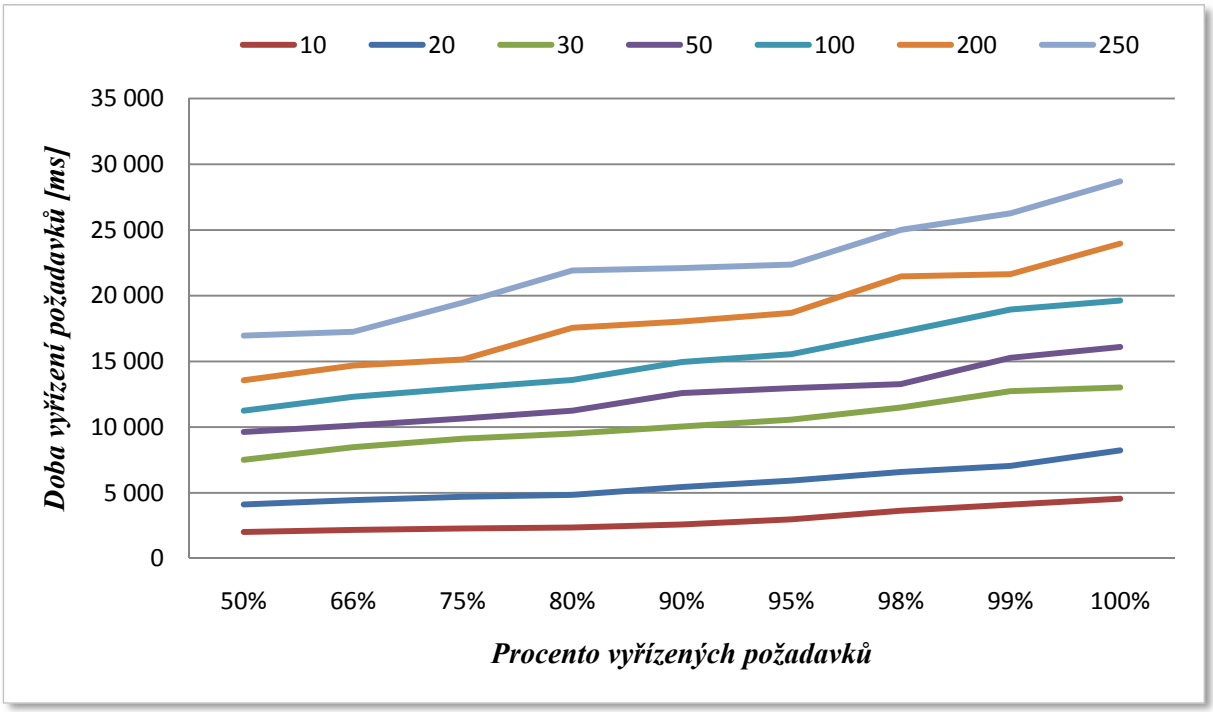
- *MaxClients* 250
- *MaxRequestsPerChild* 20000
- *KeepAlive* Zapnuto
- *KeepAliveTimeout* 5 vteřin
- *MaxKeepAliveRequests* 100
- *HostnameLookups* Off
- Modul *mod_deflate* Načten (zapnuta komprese)
- Modul *mod_rewrite* Načten (přepisování URL)
    - Pravidla přepisování URL nastavena v .htaccess a pro adresář ze soubory projektu nastaveno AllowOverride na All

Databázový server MySQL běží na stejném serveru ve verzi 4.0.24. Nastavení sledovaných parametrů uvádím v následujícím seznamu:

- *key_buffer* 48M
- *table_cache* 2048
- *query_cache_limit* 2MB
- *query_cache_size* 32MB

5.1.3 Zátěžový test na původních nastaveních

Test byl spuštěn z počítače ze 100Mbps připojením a odesláno bylo 1 000 požadavků s konkurencí od 10 do 250, jak je možné vidět v horní části grafu. Zde jsou výsledky.



***Graf č. 16** – Zátěžový test www.dreamlife.cz na původních nastaveních*

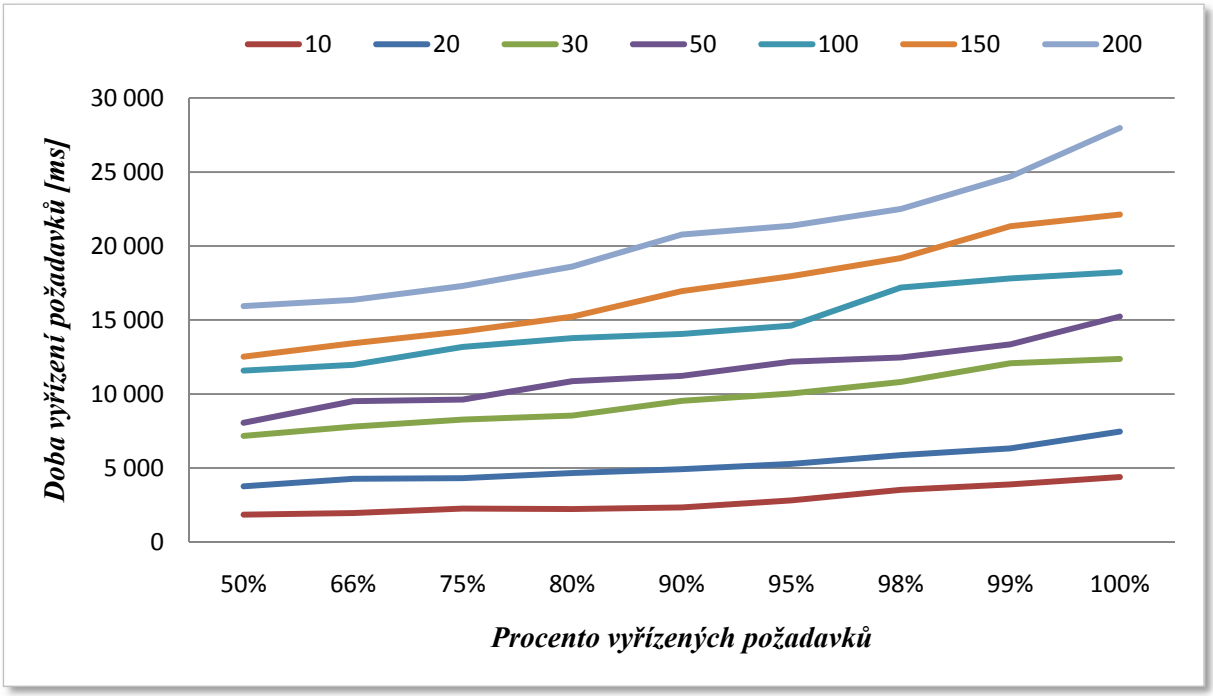
5.1.4 Změny nastavení dle doporučení a výsledky testů

Na první pohled bych přesunul nastavení modulu mod_rewrite a přepisovacích pravidel do konfigurace virtuálního hostitele a následně bych zakázal použití direktivy AllowOverride. Toto nastavení však není možné provést, protože na serveru běží několik dalších aplikací, na které by zejména vypnutí AllowOverride mohlo mít negativní účinky.

Každopádně bych vyzkoušel snížit hodnotu direktivy KeepAliveTimeout a zvýšit hodnotu MaxKeepAliveRequests. Z hlediska konfigurace Apache je to asi vše, co pro vyladění výkonu můžu v tuto chvíli udělat. Direktiva MaxClients je vzhledem k dostupné operační paměti asi na svém maximu a v době nejvyššího vytížení serveru nezbývá moc místa pro další navyšování.

Podle stavových proměnných MySQL jsem však zjistil, že dochází k poměrně častému využívání mezipaměti dotazů a tak je vhodné vyzkoušet navýšit nastavení těchto hodnot.

Na serveru je již nasazeno cachování PHP a proto nelze vyzkoušet, jaký by mělo dopad na jeho výkon.



***Graf č. 17** – Nejlepší dosažený výkon po vyladění konfigurace Apache a MySQL*

Graf č. 17 ukazuje výkon serveru po provedení změn. KeepAliveTimeout byl nastaven na 2 vteřiny, MaxKeepAliveRequests na hodnotu 300.

Parametry MySQL jsem změnil následovně:

- query_cache_limit z původních 2MB na 5MB
- query_cache_size z původních 32MB na 100MB

5.1.5 Závěr

Oba zátěžové testy jsem prováděl několikrát a použil jsem nejlepší dosažené výsledky, ze kterých je patrné, že došlo ke zvýšení výkonu o necelých 6%. Tento výsledek si vysvětluji především snížením hodnoty direktivy KeepAliveTimeout, které předchozí hodnota mohla způsobovat spožděné ukončování procesů webového serveru a tím docházelo k prodlevám při vyřizování většího počtu požadavků.

5.2 www.impuls.cz

Dalším ostrým projektem je portál Rádia Impuls na adrese http://www.impuls.cz/.

5.2.1 Hardware

Samotná aplikace projektu Rádia Impuls běží na serveru SunFire X2100 s jedním dvoujádrovým procesorem AMD Opteron 1210 s frekvencí 1,8 GHz. Operační paměť serveru má velikost 6GB a server je osazen dvouma pevnými disky SATA s velikostí 250GB v RAID 1.

Databáze však běží na odděleném serveru se stejnou základní konfiguraci, akorát s 8GB operační paměti a 2x 500GB disky v RAID 1. Oba servery jsou na stejné síti a jsou spojeny 100Mbps linkou.

5.2.2 Software

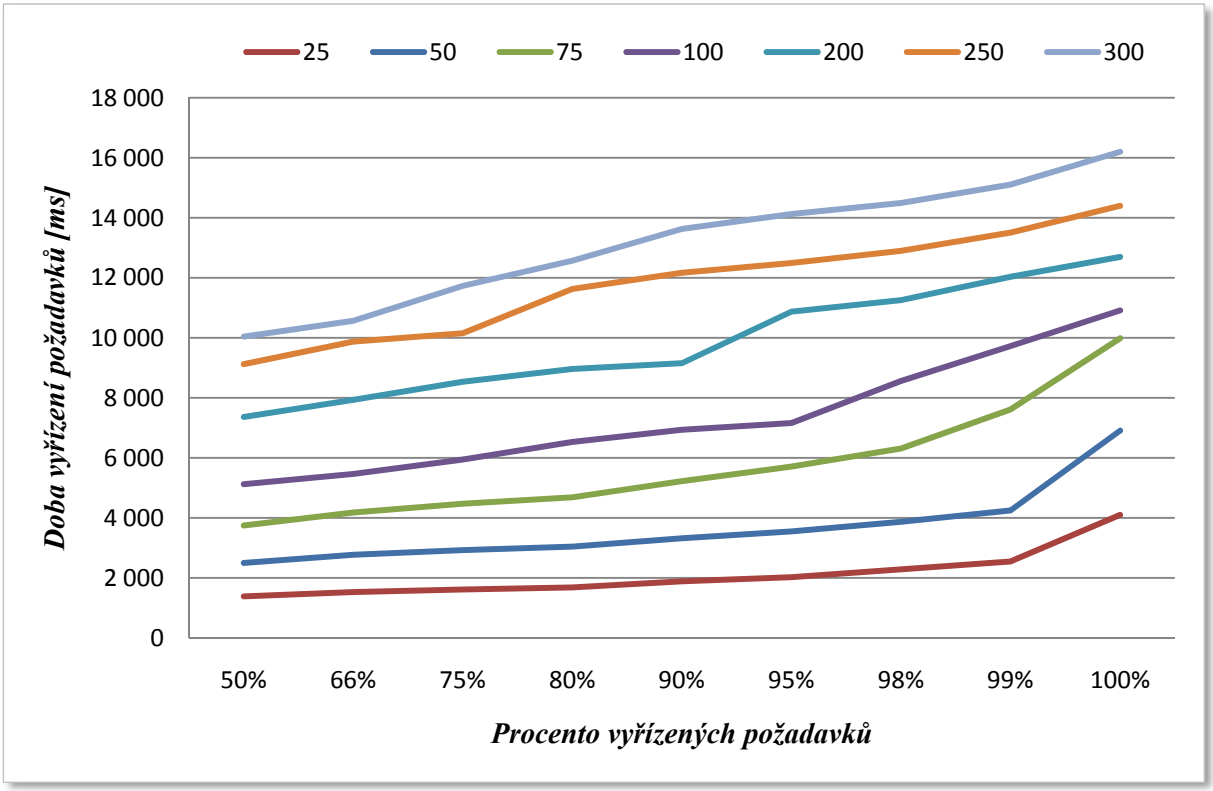
Na serveru běží Apache ve verzi 2.2.3 v 64-bitovém režimu s MPM Prefork. Nastavení zobrazuje následující seznam:

- *MaxClients* 200
- *MaxRequestsPerChild* 500
- *KeepAlive* Zapnuto
- *KeepAliveTimeout* 1 vteřina
- *MaxKeepAliveRequests* 100
- *HostnameLookups* Off
- Modul *mod_deflate* Načten (zapnuta komprese)
- Modul *mod_rewrite* Načten (přepisování URL)
    - Pravidla přepisování URL nastavena v .htaccess a pro adresář ze soubory projektu nastaveno AllowOverride na All

Databáze běží na vyhrazeném databázovém serveru, ve verzi 5.0.32. Zde jsou hodnoty sledovaných nastavení:

- *key_buffer* 32M
- *table_cache* 200
- *query_cache_limit* 16MB
- *query_cache_size* 100MB

5.2.3 Zátěžový test na původních nastaveních



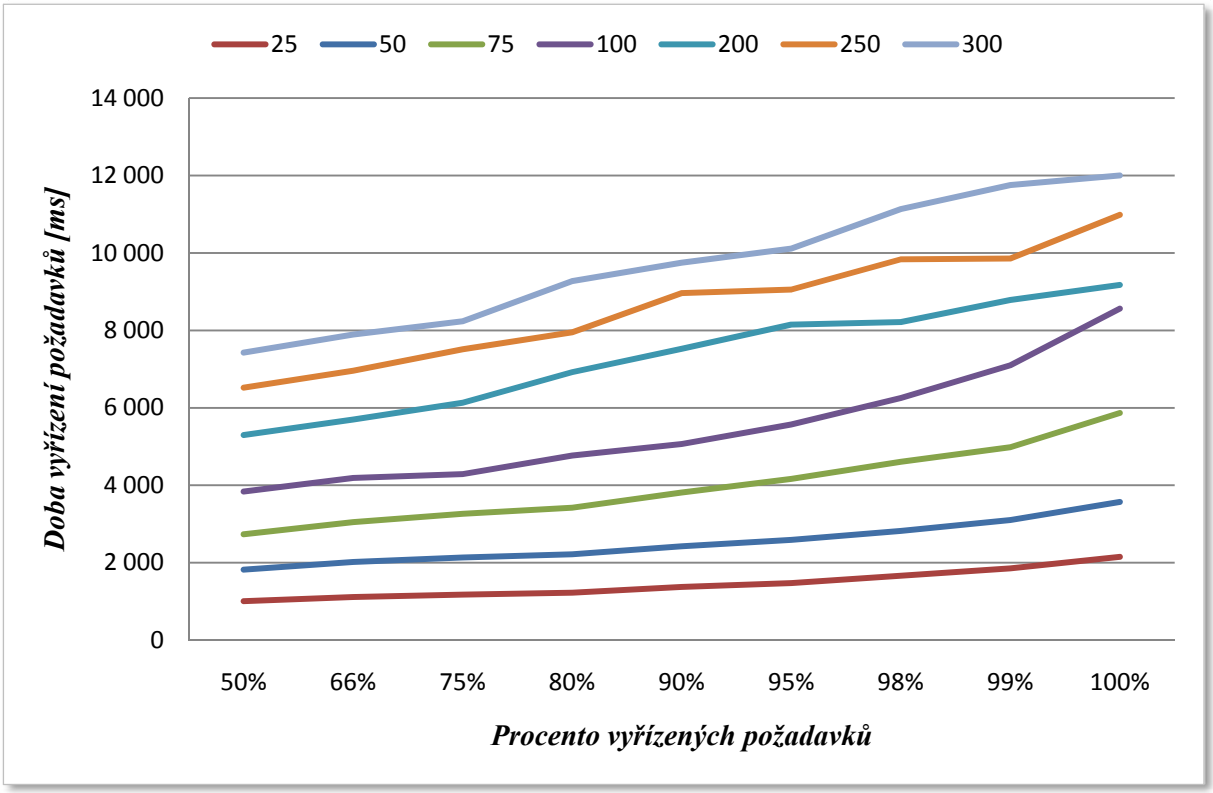
Graf č. 18 *– Zátěžový test www.impuls.cz na původních nastaveních*

5.2.4 Změny nastavení dle doporučení a výsledky testů

Sledováním serveru jsem zjistil, že i v době největší zátěže zůstává určité procento paměti volné. Vyzkouším tedy zvýšit hodnotu direktivy MaxClients na 400. Vzhledem k tomuto nastavení budu zátěžové testy provádět s konkurencí od 25 do 300.

Zjišťováním hodnot stavových proměnných MySQL jsem zjistil vysoké číslo u parametru opened_tables. Jak jsem zmínil v kapitole o MySQL, vysoká hodnota této proměnné ukazuje na časté operace s tabulkami. Novou hodnotu table_cache nastavíme na několikanásobek původní hodnoty.

PHP na tomto serveru nepoužívá cachování. Vyzkouším tedy, jaký bude mít doinstalování APC (Alternative PHP Cache) dopad na výkon.



Graf č. 19 *– Nejlepší dosažený výkon po vyladění konfigurace Apache, MySQL a PHP*

5.2.5 Závěr

Zvýšení výkonu (graf č. 19) o 22 – 25% způsobilo především cachování PHP a zvýšení hodnoty direktivy MaxClients, které se projevilo hlavně při vysoké konkurenci.

Sledování stavových proměnných ukázalo, že nastavení table_cache na hodnotu 1000 se zdá být nejoptimálnějším. Hodnota počítadla opened_tables výrazně klesla, tudíž MySQL neprovádí operace spojené s otevíraním a zavíraním tabulek tak často.

6 Závěr

Ve své práci jsem se zabýval možnostmi optimalizace výkonu LAMP. Jednotlivé části webového serveru jsem testoval samostatně a každou provedenou změnu v konfiguraci jsem pečlivě otestoval. Z důvodu omezení vlivu vedlejších faktorů na jednotlivá měření jsem testy prováděl několikrát a v práci uvádím ty nejlepší výsledky. Porovnání vlivu jednotlivých nastavení na výkon jsem uváděl vzhledem k výchozímu nastavení serveru.

V konečném důsledku se podařilo konfiguraci vyladit tak, že server stíhal vyřizovat všechny požadavky, které jsem poslal, ve stanoveném časovém limitu. Oproti hodnotám testů ve výchozím nastavení je to vynikající výsledek. Největší dopad na výkon u webového serveru Apache mělo použití komprese a modulu mod_deflate. Důvodem je snížení objemu přenášených dat, čímž došlo k redukci zatížení na síťovém rozhraní a tím ke značnému zvýšení výkonu. Komprese byla v testech na modelové aplikaci parametrem, který výkon serveru ovlivnil z největší části.

Hlavním přínosem této práce je důkladnější pohled na problematiku ladění výkonu webového serveru. Výsledky, které jsem uvedl v této bakalářské práci budou jistě přínosem jak pro mne samotného, tak pro společnost, ve spolupráci se kterou probíhalo testování na reálných aplikacích. Jedná se o společnost ViaAurea, s.r.o. zabývající se řešeními na míru v oblasti vývoje interaktivních webových aplikací. Pro tuto společnost v současné době pracuji.

Vzhledem ke složitosti architektury webového serveru LAMP nelze jednoduše a jednoznačně sestavit seznam obsahující obecná doporučení platná pro všechny systémy. Každý server je jedinečný a častokrát vyžaduje specifická nastavení v kooperaci s aplikacemi, které na něm běží. Cílem této práce bylo shrnout možnosti optimalizace a jejich názorná ukázka jak na modelové aplikaci, tak na reálném systému. Odpovědnost za nastavení spadá do kompetence daných systémových administrátorů. Můžu však říci, že vhodným nastavením zde uvedených parametrů lze zvýšit výkon téměř každého webového serveru.

Konzultace k teoretické části mi byly poskytnuty ze strany společnosti QCM, s.r.o. Tímto bych rád poděkoval panu Ing. Jiřímu Vrbovi, Ph.D. za poskytnuté informace. Rád bych také poděkoval vedoucímu mé práce panu Ing. Adamu Heroutovi, Ph.D.

Literatura

[1] Gutmans, A., Bakken, S., Rethans, D.: Mistrovství v PHP5. Computer Press, Brno, 2008. ISBN 978-80-251-1519-0

[2] Kofler, M.: Mistrovství v MySQL5. Computer Press, Brno, 2007. ISBN 978-80-251-1502-2

[3] Sobell, M.: Mistrovství v Linuxu. Computer Press, Brno, 2007. ISBN 978-80-251-1726-2

[4] Apache Software Foundation: Apache HTTP Server Version 2.2 Documentation [online]. Dostupné na URL: <http://httpd.apache.org/docs/2.2/>

[5] MySQL AB: MySQL 5.0 Reference Manual [online]. Dostupné na URL: <http://dev.mysql.com/doc/refman/5.0/en/>

[6] The PHP Group: PHP Manual [online]. Dostupné na URL: <http://www.php.net/manual/en/>

[7] Netcraft Web Server Survey [online]. Dostupné na URL: <http://news.netcraft.com/archives/web_server_survey.html>

Seznam příloh

Příloha 1. CD se zdrojovými texty modelové aplikace a výsledky provedených testů.