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

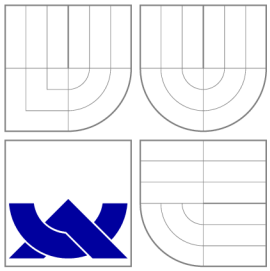
ZÁTĚŽOVÉ TESTY V PROSTŘEDÍ LAMP (LINUX/APACHE/MYSQL/PHP)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

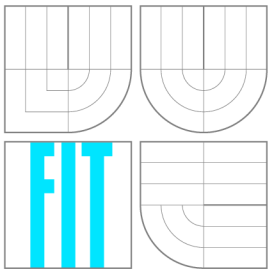
AUTOR PRÁCE
AUTHOR

TOMÁŠ KRÁL

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

ZÁTĚŽOVÉ TESTY V PROSTŘEDÍ LAMP (LINUX/APACHE/MYSQL/PHP)

BENCHTESTS OF LAMP (LINUX/APACHE/MYSQL/PHP) SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ KRÁL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá problematikou zátěžových testů systému LAMP. Jsou vní vysvětleny principy testování a popsány základní metodiky a programy pro provádění zátěžových testů.

Abstract

This bachelor's thesis deals with benchtests of LAMP systems. Work explain principes of testing and describe basic methodics and programs for testing.

Klíčová slova

Zátěžové testy, testy, Apache, PHP, MySQL, httperf, Apache Benchmark, Siege

Keywords

Benchtests, tests, Apache, PHP, MySQL, httperf, Apache Benchmark, Siege

Citace

Tomáš Král: Zátěžové testy v prostředí LAMP
(linux/apache/mysql/php), bakalářská práce, Brno, FIT VUT v Brně, 2009

Zátěžové testy v prostředí LAMP (linux/apache/mysql/php)

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Adama Herouta, Ph.D.
Další cenné informace mi poskytl Ing. Jiří Vrba.

.....
Tomáš Král
19. května 2009

Obsah

Kapitola 1

Úvod

Cílem této práce je rozebrat problematiku zátěžových testů LAMP a navrhnout ucelený způsob testování těchto systémů.

Práce je rozdělena celkem do osmi kapitol. První kapitolou je tento úvod. Druhá kapitola nazvaná "Principy zátěžových testů"obsahuje vysvětlení co to jsou zátěžové testy, k čemu se používají a s jakými problémy se lze při testování setkat. Třetí kapitola popisuje systém typu LAMP a jeho jednotlivé části. Čtvrtá kapitola se již věnuje programům, které se nejčastěji používají k provádění zátěžových testů systému LAMP. Pátá kapitola se zabývá metodikou provádění testů v této práci a popisuje použitý skript. Následující kapitola obsahuje popis výsledků provedených testů. Jsou zde prezentovány testy na různé typy zátěže a s různou konfigurací serveru. V sedmé kapitole jsou popsána základní doporučení a nastavení ke zvýšení výkonu systému LAMP. Poslední kapitolou je závěr. Zde je zhodnocení celé práce a dosažených výsledků.

Kapitola 2

Principy zátěžových testů

Zátěžové testy, často se používá i pojem výkonnostní testy, slouží k získávání výkonnostních charakteristik měřeného systému při různých typech zátěže (denní špička, noční či měsíční dávkové zpracování atd.). Často se také používají k hodnocení konfigurace systému. Hlavním přínosem zátěžových testů je získání informací o pravděpodobném chování systému při zatížení ještě před vlastním nasazením systému do ostrého provozu. Díky tomu lze i efektivněji plánovat náklady na případné další rozšiřování.

Zátěžové testy mají za úkol simulovat reálný provoz. Pokud chceme zjistit, kde má systém výkonnostní hranice, simuluje se provoz i mnohonásobně převyšující očekávaný reálný provoz. Testovací nástroj při testování zaznamenává odezvu systému. Tyto záznamy lze poté dále zpracovávat a vytvářet souhrny či grafy, které nám přehledně ukáží, jaký mělo zatížení vliv na chod systému. V našem případě budeme především sledovat časy zpracování dotazu na serveru (Apache, MySQL, PHP).

Nejčastější důvody k provádění výkonnostních testů jsou:

- Zjištění, zda stávající výkon dostačuje našim požadavkům.
- Ladění systému k dosažení maximálního výkonu.
- Zjištění výkonnostního limitu systému (do jaké míry lze systém zatížit, při zachování korektního chování).
- Ověření chování systému při výpadku některé z komponent (ověření nastavení loadbalanceru).

Při testování webových serverů se nejčastěji zaměřujeme na určení maximálního počtu dotazů za sekundu (viz kapitola 2.2), které je server schopen obsloužit a zároveň splnit dané podmínky. Nejčastější podmínky jsou:

- Klient nebude čekat na odpověď déle, než určený čas.
- Odpověď není považována za platnou, pokud se během zpracování vyskytla chyba.
- Pokud chceme testovat i mechanizmus cachování, měl by každý dotaz směřovat na náhodně vybraný soubor.

Vždy bychom se měli vyvarovat spouštění testovacího programu na samotném serveru, který chceme testovat. Server bude zatížen jak dotazy, které bude klient zasílat, tak i samotnou prací klientského programu. Výsledky, které takto získáme, mohou být velmi nepřesné.

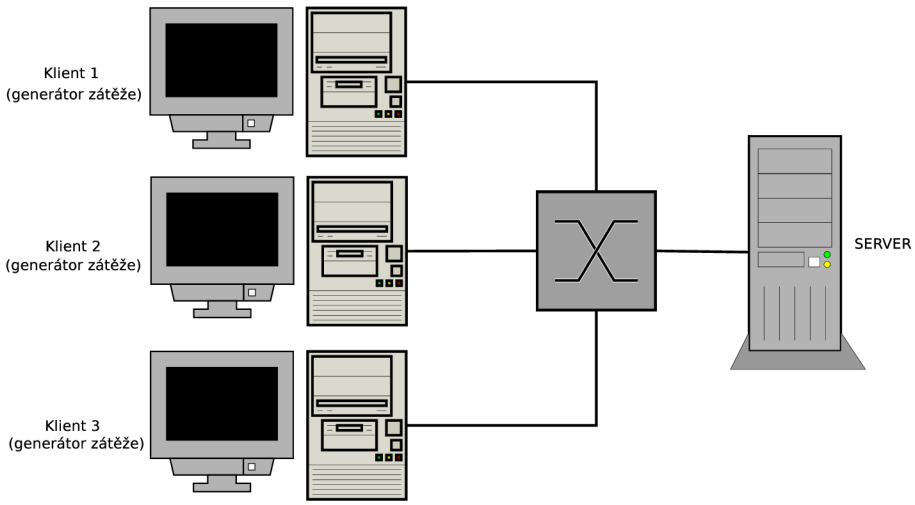
2.1 Úzká místa testování

K dosažení co nejpřesnějších výsledků při testování je důležité, aby jediným limitujícím prvkem byl samotný výkon serveru.

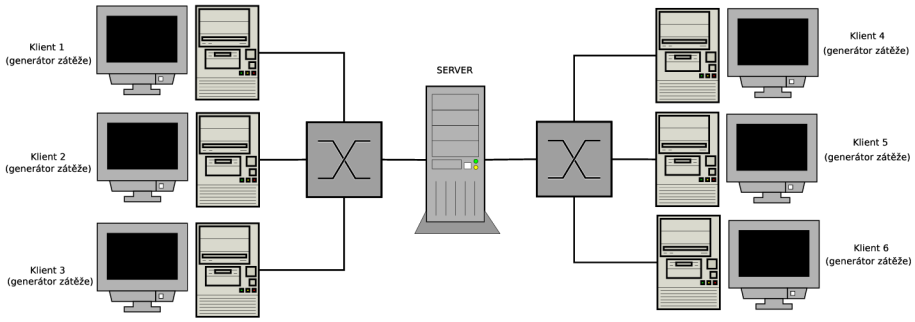
Výsledky testování mohou ovlivnit dva nejčastější limitující faktory: propustnost sítě a výkon klientského počítače.

Síť - je potřeba zajistit co největší propustnost sítě. Pro testovací účely je nejlepší vytvořit vlastní malou síť, kde nebude probíhat žádný jiný provoz mimo testování. Nejčastěji se používá zapojení, kde je klient přímo propojen se serverem. V případě, že vytěžujeme server více klienty, je vše propojeno za pomoci switche (viz obrázek 2.1). Pokud nás i přesto limituje propustnost sítě, je možné připojit k serveru více sítí (za předpokladu že je server vybaven více síťovými kartami). Příklad takového zapojení je na obrázku 2.2.

Klientský počítač - velmi často se může stát, že samotný počítač, na kterém je spuštěn generátor zátěže, nemá dostatečný výkon odeslat serveru požadované množství dotazů. Nejlepším způsobem, jak si ověřit, zda klient není úzkým místem systému, je spustit stejný test dvakrát s různým počtem klientů, ale vždy musí být užity takové parametry testu, aby zatížení serveru zůstalo u obou případů stejné (propojení více klientů se serverem viz obrázek 2.1). Pokud jsou oba výsledky rozdílné, lze nerovnost připsat na vrub nedostatečného výkonu klienta. Nelze se řídit pouze podle ukazatele zatížení procesoru klienta. Některé programy na testování, například *httperf*, ukazují 100 % vytížení procesoru, ačkoliv mají ještě k dispozici volný výpočetní výkon.



Obrázek 2.1: Síťové zapojení pomocí switche



Obrázek 2.2: Síťové zapojení eliminující vytížení sítě

2.2 Základní metriky

Existuje spousta různých metrik, které lze sledovat při měření výkonu webových serverů. Pro měření výkonnosti serveru se nejčastěji používá počet dotazů za sekundu, počet současných spojení a datová propustnost.

Počet dotazů za sekundu - počet dotazů, které je server schopen obsloužit za jednu sekundu. Tato hodnota je užitečná především k určení hrubé síly serveru. Při měření této hodnoty se však musí vzít v úvahu, zda se pro každý dotaz znovu navazovalo nové spojení, či zda se provedlo více dotazů během jednoho spojení.

Počet současných spojení - jde o počet spojení, které je server schopen obsloužit najednou bez výrazného zpomalení nebo chyb.

Datová propustnost - jde o počet bytů, které je server schopen přenést za časovou jednotku. Tato hodnota je užitečná hlavně při spojení s jinou metrikou. Je jednoduché dosáhnout velké propustnosti při přenosu jednoho velkého souboru. Pokud se však přenáší více menších souborů, je často naměřená propustnost mnohem nižší.

Kapitola 3

LAMP

LAMP je zkratka pro soubor softwaru obsahující webový server Apache, databázový server MySQL a skriptovací jazyk PHP (někdy se také může jednat o Perl nebo Python) [13], to vše běžící na operačním systému Linux. Ač je každý program vyvíjen samostatně jinou skupinou lidí, stala se tato kombinace velmi populární pro vývoj dynamických webových stránek. A to hlavně díky své nízké ceně (všechny programy jsou distribuovány pod open source licencí) a lehké dostupnosti. Veškeré potřebné komponenty jsou standardně obsaženy v každé distribuci Linuxu.

3.1 Linux

Linux je obecný název pro operační systémy založené na Linuxovém jádru, které původně napsal Linus Torvalds. Jádro je licencováno pod licencí GNU GPLv2.

Linux je nejrozšířenější především jako operační systém pro internetové a intranetové servery a v oblasti vysoce výkonných systémů (v žebříčku TOP 500 nejvýkonnějších počítačových systémů běží v současnosti na 87,8 % počítačů Linux). [14]

3.2 Apache HTTP Server

Apache je nejpoužívanější webový server na světě (s 48,89 % zastoupením za březen 2009) [11]. Program je distribuován pod open source Apache licencí.

Apache je používán k poskytování jak statických, tak dynamických stránek. Díky svému modulárnímu návrhu není problém pomocí modulů server rozšířit o další funkce, hlavně pak o podporu skriptovacích jazyků pro vytváření dynamických stránek. Mezi nejpopulárnější jazyky, které se používají ve spojení s Apachem, patří hlavně PHP, Python, Perl, Ruby.

3.2.1 MPM moduly

Apache od verze 2.0 používá modulární přístup i u nejzákladnějších funkcí webového serveru. Program obsahuje výběr MPM[1] modulů pro souběžné zpracování. Každý modul MPM je odpovědný za spuštění procesu serveru a za obsluhování dotazů pomocí dceřiných procesů anebo vláken. To, zda se použijí vlákna nebo procesy, závisí na způsobu implementace konkrétního modulu.

[1] Multi-Processing Modules (MPMs)

Apache těží ze dvou hlavních výhod MPM:

- Díky MPM lze lépe a efektivněji podporovat různé operační systémy. Konkrétněji Apache pod Windows je mnohem efektivnější při použití modulu *mpm_winnt*. Díky tomu, že tento modul nemusí být přenositelný na jiné operační systémy, lze využít specifické vlastnosti pro tento systém a dosáhnout tak vyššího výkonu.

- Server může být lépe upraven pro konkrétní případy. Například stránky, které se musejí potýkat s rozšiřitelností, mohou využít moduly používající vlákna, například modul *worker*. Server vyžadující vysokou stabilitu nebo zpětnou kompatibilitu se starými programy může využít modulu *prefork*.

Z pohledu uživatele se může zdát, že MPM moduly jsou stejné jako normální moduly. Hlavní rozdíl je pouze v tom, že právě jeden MPM modul musí běžet na serveru.

MPM modul se musí vybrat již během konfigurace před samotnou kompilací a zkompilovat jej s jádrem serveru Apache. Právě díky tomu, že se modul vybírá ještě před kompilací, má kompilátor možnost optimalizovat mnoho funkcí využívajících vlákna a znatelně tak zvýšit výsledný výkon.

Výběr MPM modulu se provádí pomocí přepínače *–with-mpm=NAME*, kde *NAME* je název požadovaného MPM modulu.

Pokud chceme zjistit, který MPM modul využívá již zkompilovaný server, lze tak učinit pomocí `httpd -l`. Tento příkaz zobrazí seznam všech modulů, které byly zkompilovány společně se serverem, včetně MPM modulů.

Apache na různých operačních systémech používá různé výchozí MPM moduly. Tabulka 3.1 ukazuje, jaký MPM modul se použije na daném operačním systému, pokud při konfiguraci explicitně nespecifikujeme jinak.

operační systém	výchozí MPM modul
BeOS	beos
NetWare	mpm_netware
OS/2	mpmt_os2
Unix	prefork
Windows	mpm_winnt

Tabulka 3.1: Výchozí MPM moduly

V operačním systému Linux se nejčastěji používají dva MPM moduly: prefork a worker.

Modul prefork

Tento modul používá podobný způsob práce jako Apache do verze 1.3. Nepoužívají se zde vlákna, ale pro obsluhu dotazů se vytváří dceřiné procesy. Jestliže server vytvoří 20 dceřiných procesů, znamená to, že může současně obsloužit 20 dotazů. To je nejlepší způsob, jak od sebe izolovat jednotlivé dotazy. Dojde-li k problému s některým dotazem, neohrozí se ostatní. Tento modul se často využívá hlavně díky zpětné kompatibilitě ze staršími aplikacemi, které nejsou schopné pracovat s vlákny.

Modul prefork je velmi přizpůsobivý. Proto je jen zřídka potřeba měnit jeho nastavení. Nejdůležitější je, aby nastavení *MaxClients* bylo nastaveno na dostatečnou hodnotu. Tato

hodnota udává, kolik dotazů je server schopen zpracovat současně. Nastavení nesmí být neúměrně velké, aby zbyl dostatek volné paměti pro zpracování všech příchozích dotazů. [5]

Modul worker

Modul *worker* implementuje hybridní přístup. Využívá jak procesů, tak i vláken. Právě díky využití vláken je server schopen obsloužit velké množství dotazů s menším vytížením zdrojů systému. A zároveň díky využití procesů je si server schopen zachovat stabilitu.

Nejdůležitější direktivy používané pro řízení tohoto MPM modulu jsou:

- *TheadsPerChild* - udává počet vláken vytvořených v každém procesu.
- *MaxClients* - udává počet současně spuštěných vláken.
  [6]

3.3 MySQL

MySQL je relační databázový systém, vlastněný a vyvíjený firmou Sun Microsystems. Firma poskytuje MySQL jak pod GNU GPL licencí, tak i pod různými komerčními licencemi.

MySQL je populární především jako databázový systém pro webové aplikace. Svoji popularitu si získal především díky vysoké popularitě PHP, který je právě často kombinován s MySQL.

3.4 PHP

PHP je skriptovací jazyk původně napsaný právě pro generování dynamických webových stránek. Postupně se vyvíjel a nyní obsahuje i nástroje pro práci s příkazovým řádkem a může být využit společně s knihovnami pro grafické uživatelské rozhraní. Distribuce PHP probíhá pod vlastní open source PHP licencí.

PHP skripty jsou prováděny na straně serveru, k uživateli je přenesen pouze samotný výstup skriptu. Skriptům je možné předávat vstupní data. Výstup je nejčastěji ve formátu HTML.

Kapitola 4

Programy pro testování

Při jednoduchém statickém testování výkonu klient (nebo více klientů) provádí velké množství dotazů na stále stejný soubor na serveru a současně je měřen dosažený výkon (nejčastěji počet dotazů, které server obsloužil za sekundu). Díky tomu, že je opakovaně požadován po serveru stále stejný soubor, je tato metoda testování velmi odlišná od zatížení, pod kterým se systém ocitne při reálném provozu. Lze však díky tomu získat přehled o hrubé síle systému.

4.1 Apache Benchmark (ab)

Apache Benchmark (*ab*) je program pro příkazovou řádku. Program je součástí distribuce Apache, v některých distribucích je však v odděleném balíčku *apache2-utils*. Program *ab* jen stále dokola stahuje jeden zadaný soubor. Pokud se chceme co nejvíce přiblížit k reálnému zatížení, je lepší použít program *siege*, který umožňuje v cyklu spouštět více zadaných URL.

4.1.1 Volby

Nejčastějšími volbami, které se při testovaní pomocí *ab* používají, jsou *-n* a *-c*.

-n Určuje celkový počet dotazů.

-c Počet současně provedených dotazů.

Jako poslední argument se zadá celá adresa testovaného souboru včetně protokolu.

Dalším často používaným přepínačem je *-k*, který zapne podporu KeepAlive u protokolu HTTP. Standardně se pro každý dotaz znovu navazuje spojení. Při použití KeepAlive se během jednoho spojení provede více dotazů bez opakovaného odpojování a znovu navazování spojení.

Program má i pokročilejší možnosti. Umožňuje například přihlašování pomocí BASIC Authentication, posílat na server data metodou POST, vložit vlastní pole do HTTP hlavičky, nebo místo protokolu HTTP využít šifrovaného HTTPS. Všechny potřebné volby k těmto vlastnostem lze najít v manuálových stránkách [10].

4.1.2 Příklad použití

Následující příkaz provede tisíc dotazů na získání souboru *test.html* s tím, že vždy odešle deset dotazů najednou.

```
ab -n 1000 -c 10 http://www.example.com/test.html
```

4.1.3 Výstup

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

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.2.11
Server Hostname:        example.com
Server Port:            80

Document Path:          /test.html
Document Length:        1231 bytes

Concurrency Level:      10
Time taken for tests:   0.578 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      1473330 bytes
HTML transferred:       1237155 bytes
Requests per second:    1729.55 [#/sec] (mean)
Time per request:       5.782 [ms] (mean)
Time per request:       0.578 [ms] (mean, across all concurrent requests)
Transfer rate:          2488.47 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3   1.3      2       9
Processing:     1    3   1.3      3       9
Waiting:        0    2   1.3      2       8
Total:          2    6   1.4      5      11

Percentage of the requests served within a certain time (ms)
  50%      5
  66%      6
  75%      6
  80%      7
  90%      7
  95%      9
  98%     10
  99%     10
 100%     11 (longest request)
```

Výstup programu obsahuje souhrnné informace o provedeném testu. Jsou zde například údaje o celkovém množství přenesených dat, časové prodlevě odpovědi od serveru, o době

čekání, připojování, zpracovávání dotazů a případně informace o chybách, které během testování nastaly. Pokud nám tyto informace nestačí a chtěli bychom podrobnější údaje, lze využít volby *-e* nebo *-g*.

-e Uloží výsledek testování do souboru ve formátu CVS, kde je pro každé procento (1 - 100 %) z celkového počtu iterací uveden průměrný čas obsluhy dotazu. Identifikátory sloupců jsou uloženy na prvním řádku souboru.

-g Vytvoří rovněž soubor s podrobnostmi o testování, ale ve formátu TSV, který lze použít v programu *gnuplot* pro vytvoření grafů. Soubor obsahuje více informací než soubor vytvořeny pomocí *-e*. V tomto souboru jsou jednotlivé sloupce odděleny tabulátorem. Identifikátory k jednotlivým sloupcům jsou zapsány na prvním řádku souboru.

4.2 Httperf

Jde o testovací nástroj pro příkazovou řádku od HP Research Labs pro měření výkonosti webových serverů.

Httperf je navržen tak, aby byl co nejméně závislý na mechanizmech operačního systému. Proto program běží jako jeden proces nevyužívající vlákna. Pro komunikaci se serverem využívá neblokujících I/O[1] operací. Díky tomu je pro procesor jednodušší plánování procesů a minimalizuje se nutnost přepnutí kontextu[2]. Dále pak *httperf* využívá vlastní timeout management, který se snaží vyhnout náročným systémovým voláním a POSIX signálům. [1]

4.2.1 Problém nedostatku souborových deskriptorů

Pokud se během testování pomocí *httperf* objeví na výstupu chyby typu *fd-unvail*, znamená to, že byl vyčerpán povolený limit souběžně otevřených souborů. *Httperf* otevře nový soubor pro každé souběžné spojení. Většina operačních systémů používá limit pro počet souborových deskriptorů jak pro celý systém, tak pro proces. Limit otevřených souborů pro systém je většinou dostatečně velký a není jej třeba měnit. To ovšem neplatí pro limit souborů pro proces, který lze během testování velmi jednoduše překročit. Nejčastěji je tento limit nastaven v rozmezí mezi 256 a 2048. Pokud předpokládáme nastavený timeout na 5 sekund a limit deskriptorů 2000 pro proces, je maximální udržitelná rychlost cca 400 spojení za sekundu. [1]

Změna počtu povolených souborových deskriptorů se provádí v souboru */etc/security/limits.conf*. Na konec tohoto souboru je potřeba přidat následující řádek:

```
* hard nofile 65535
```

Místo * lze použít jméno uživatele, kterému chceme limit změnit, eventuelně při použití * se limit aplikuje pro všechny uživatele. Změnu je také potřeba provést v souboru */usr/include/bits/typesizes.h*. Zde je na řádku `#define __FD_SET_SIZE 1024` potřeba přepsat původní limit na nový. Původní limit v tomto systému byl 1024. Nyní už stačí jen zkompilovat *httperf* ze zdrojových kódů. [8]

[1] Input/Output - vstupně výstupní
[2] context switching

4.2.2 Volby

Httperf používá odlišný způsob určování parametrů testu než používá *ab*. U *ab* se nastavoval pouze celkový počet připojení a počet současných připojení (pokud nebylo explicitně zapnuto KeepAlive, odpovídalo jednomu dotazu jedno připojení). *Httperf* toto nastavení ještě rozšiřuje o počet dotazů, které se provedou během jednoho připojení. To znamená, že celkový počet dotazů, které se provedou během celého testu, je: $pocet\_pripojeni * pocet\_dotazu$.

Základní volby jsou:

--uri Adresa souboru, na který se budou odesílat dotazy.

--server IP adresa nebo název serveru, který se bude testovat.

--num-conns Počet vytvořených spojení.

--num-calls Udává, kolik dotazů se při každém spojení provede. Při testování s volbami *–num-conns=500 –num-calls=20* bude celkový počet dotazů 10000.

--rate Určuje, kolik nových spojení se naváže za každou sekundu. Při nastavení *–num-conns=500 –num-calls=20 –rate=10* testovací program provede 200 dotazů na soubor každou sekundu.

--hog Použije se tolik TCP portu, kolik bude potřeba. Bez této volby *httperf* použije pouze porty z rozsahu od 1024 do 5000.

--wlog=B,F Slouží místo *–uri*, umožňuje specifikovat více adres, na které se odesílají dotazy.

- **B** Může být "y"nebo "n". Pokud je "y", po dosažení konce souboru přejde znovu na začátek. Pokud je "n", poté co se dosáhne konce souboru, ukončí se i test.
- **F** Soubor se seznamem adres souborů. Jednotlivé adresy jsou odděleny znakem ASCII NUL (\0).

Stejně jako u Apache Benchmark má program *httperf* spoustu dalších pokročilých funkcí, jako například volby pro ssl, cookies, specifikace HTTP hlavičky ... [12]. Bohužel, *httperf* neumožňuje uložení podrobnějších informací o testování, jako tomu je například u *ab* pomocí voleb *-g* a *-e*.

Volby pro testování relacemi

Httperf obsahuje volby, které umožňují měření relací místo jednotlivých dotazů. Relace je několik dávek dotazů od sebe oddělených daným časovým intervalem. Každá dávka probíhá následujícím způsobem:

- je odeslán první dotaz
- po zpracování odpovědi na první dotaz se všechny zbývající dotazy z dávky odešlou současně

--burst-length Udává délku dávky. Počet dotazů provedených během jedné dávky.

--wsess=N1,N2,X

- **N1** Celkový počet relací.
- **N2** Počet dotazů během relace.
- **X** Prodleva v sekundách mezi dávkami.

Uvažujeme-li následující příklad: *–wsess=100,50,10 –burst-length=5*, pak bude tento test obsahovat celkem 100 relací, v každé relaci 50 dotazů. Dotazy se budou odesílat v dávkách po 5, v každé relaci proběhne těchto dávek 10. Mezi jednotlivými dávkami se vždy vyčká 10 sekund.

--wsesslog=N,X,F

- **N** Celkový počet relací.
- **X** Prodleva v sekundách mezi dávkami.
- **F** Soubor, který definuje relace.

Příklad souboru definujícího relace pro –wsesslog

```
# session 1 definition (this is a comment)
/foo.html think=2.0
  /pict1.gif
  /pict2.gif
/foo2.html method=POST contents='Post data'
  /pict3.gif
  /pict4.gif

# session 2 definition
/foo3.html method=POST contents="Multiline\ndata"
/foo4.html method=HEAD
```

Tento soubor obsahuje dvě relace.

První relace začíná dotazem na soubor */foo.html*. V okamžiku doručení odpovědi na */foo.html* se odešlou souběžně dotazy na soubor */pict1.gif* a */pict2.gif*. Po obdržení odpovědi na poslední z těchto souborů se vyčká 2 sekundy. Po uplynutí této doby se odešle dotaz s POST daty "Post data"na soubor */foo2.html*. Jakmile přijde odpověď na */foo2.html*, odešlou se souběžně dotazy na soubor */pict3.gif* a */pict4.gif*.

Druhá relace obsahuje 2 dotazy oddělené prodlevou, která byla nastavena parametrem *X* u přepínače *–wsesslog*.

Pokud je celkový počet relací nastavených parametrem *N* u *–wsesslog* větší, než je počet definovaných relací v souboru, sekvence relací se budou opakovat do okamžiku dosažení požadovaného počtu.

Pokud použijeme přepínače *–wsess* nebo *–wsesslog*, *–rate* již nebude udávat rychlost odesílání dotazů, ale rychlost vytváření relací.

4.2.3 Příklad použití

Následující příklad ukazuje spuštění testu na soubor *test.html* na serveru *www.example.com*. Každou sekundu se vytvoří 10 nových spojení, celkem jich bude 500. Během každého spojení se provede 10 dotazů.

```
httperf --hog --server=www.example.com --uri=/test.hml --num-conns=500 \
--num-calls=20 --rate=10
```

4.2.4 Výstup a interpretace výsledků

```
httperf --hog --client=0/1 --server=www.example.com --port=80 \
--uri=/test.html --rate=10 --send-buffer=4096 --recv-buffer=16384 \
--num-conns=500 --num-calls=20
Maximum connect burst length: 1

Total: connections 500 requests 10000 replies 10000 test-duration 49.908 s

Connection rate: 10.0 conn/s (99.8 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 5.6 avg 10.5 max 63.4 median 6.5 stddev 9.4
Connection time [ms]: connect 0.0
Connection length [replies/conn]: 20.000

Request rate: 200.4 req/s (5.0 ms/req)
Request size [B]: 62.0

Reply rate [replies/s]: min 199.2 avg 200.1 max 204.0 stddev 1.5 (9 samples)
Reply time [ms]: response 0.5 transfer 0.0
Reply size [B]: header 216.0 content 1036.0 footer 0.0 (total 1252.0)
Reply status: 1xx=0 2xx=10000 3xx=0 4xx=0 5xx=0

CPU time [s]: user 23.17 system 7.77 (user 4.2.1 Volby46.4% system 15.6% total 62.0%)
Net I/O: 257.1 KB/s (2.1*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Na prvním řádku výstupu se vždy vypíše, s jakými volbami byl program zavolán. Jsou zde i volby, které jsme nezadali a u nichž byly automaticky doplněny jejich výchozí hodnoty.

Pokud nás nejvíce zajímá, jak velké zatížení je server schopen obsloužit, jsou pro nás nejdůležitější následující dvě položky: *Connection rate*[3] a *Request rate*[4]. Podle voleb, se kterými jsme spustili test, lze vypočítat, jakých hodnot by měly tyto položky nabývat. *Rate* udává navázání 10 spojení každou sekundu. U každého spojení se podle *num-calls* provede 20 dotazů. Výsledkem by mělo být 200 odeslaných dotazů za každou sekundu ($rate * num\_calls$).

Jak je vidět na příkladu výstupu *Request rate* odpovídá námi vypočítaným hodnotám. Může ovšem nastat situace, že se na výstupu objeví čísla, která budou nižší než teoreticky očekáváme. To může být zapříčiněno několika důvody. Nedostatečným výkonem klientského stroje, malou propustností sítě nebo přetížením serveru.

Přetížení serveru ve většině případů není na závadu, často nám jde o dosažení limitu výkonu testovaného serveru. Problém nastává, chceme-li generovat velké množství dotazů, ale náš klient to nezvládá. Jednou z možností, jak eliminovat přetížení klienta, je spustit *httperf* na dvou různých počítačích zároveň (příklad zapojení sítě viz obrázek 2.1) a u každého nastavit *–rate* na poloviční hodnotu. Tímto postupem vytíženost klienta klesne na polovinu, ale server bude stále stejně zatížen ($2 * rate$).

[3] Rychlost vytváření nových spojení (počet spojení za sekundu)
[4] Rychlost odesílání dotazů (počet dotazů za sekundu)

Pokud je stále *Request rate* nižší než by měl být, jde s největší pravděpodobností o přetížení serveru.

4.3 Siege

Siege je rovněž program pro příkazovou řádku, stejně jako *ab* a *httperf*. Ale na rozdíl od programu *ab*, *siege* pracuje s více souběžně běžícími vlákny. Dokáže také pracovat se seznamem URL adres uložených v souboru, na něž náhodně přistupuje. Tímto způsobem se docílí co největší podobnosti s reálným provozem, kdy na server přistupuje větší množství uživatelů současně.

4.3.1 Volby

--concurrent Počet souběžně běžících uživatelů, které bude program simulovat. Jde o počet souběžných dotazů nikoliv, současných připojení.

--file Tento přepínač určuje soubor obsahující seznam adres, na které bude *siege* přistupovat. Každá adresa musí být na samostatném řádku.

--time Čas, po který test poběží. Za číslem je nutno specifikovat časovou jednotku (S, M, H pro sekundy, minuty, hodiny).

--reps Počet opakování. Pokud bude *–concurrent=5 –reps=10*, provede se celkem 50 spojení (5 současných uživatelů, kde každý provede 10 spojení).

--internet Adresy ze souboru specifikovaného pomocí *–file* se budou vybírat v náhodném pořadí.

--delay Časová prodleva mezi dotazy virtuálního uživatele. Interval je mezi 0 a zadaným číslem v sekundách.

Seznam všech přepínačů a jejich podrobný popis viz [15].

4.3.2 Příklad použití

```
siege -v --internet --file=urls.txt
```

Program bude přistupovat na náhodně vybrané adresy ze souboru *urls.txt*. Test simuluje 15 současně pracujících uživatelů a poběží do doby než ho sami ukončíme.

4.3.3 Výstup

```
* SIEGE 2.67
** Preparing 15 concurrent users for battle.
The server is now under siege...
HTTP/1.1 200   0.00 secs:    2276 bytes ==> /test1.html
HTTP/1.1 200   0.00 secs:    2535 bytes ==> /test2.html
                          .
                          .
                          .
HTTP/1.1 200   0.00 secs:    2276 bytes ==> /test1.html
HTTP/1.1 200   0.00 secs:    2535 bytes ==> /test2.html

Lifting the server siege...      done.
Transactions:                     26 hits
Availability:                 100.00 %
Elapsed time:                   1.11 secs
Data transferred:               0.09 MB
Response time:                  0.04 secs
Transaction rate:              23.42 trans/sec
Throughput:                     0.08 MB/sec
Concurrency:                    0.91
Successful transactions:          26
Failed transactions:               0
Longest transaction:            0.19
Shortest transaction:           0.00
```

Program postupně vypisuje název souboru, na který přistupuje, včetně stavového kódu protokolu HTTP a času od odeslání dotazu do obdržení odpovědi.

Na konci výstupu jsou souhrnné statistiky. Podobně jako u *httperf* se vypisuje počet dotazů průměrně zpracovaných za sekundu.

Kapitola 5

Metodika testování

Pro testování výkonu serveru jsem zvolil metodu, kdy na server pomocí programu *httperf* posílám dotazy konstantní rychlostí. Jako hlavní ukazatel vytížení serveru sleduji rychlost přijímání odpovědí. Tento test provedu postupně několikrát vždy s vyšší rychlostí odesílání dotazů. Pokud rychlost přijímání odpovědí klesne pod rychlost odesílání dotazů došlo k přetížení serveru. V tomto bodě server dosáhl svého maximálního výkonu.

Následně do grafu vynesu závislost rychlosti odesílání dotazů na rychlosti přijímání odpovědí. Pokud server pracuje správně, průběh grafu je lineární. V místě přerušení lineárního průběhu (počet přijatých odpovědí je menší než počet odeslaných dotazů) došlo k přetížení serveru. V tomto bodě server nestačil zpracovávat všechny dotazy včas. Toto je dobře znatelné na grafu v obrázku 6.1, kde k přetížení došlo okolo rychlosti 5000 odeslaných dotazů za sekundu. Pokud server vytěžujeme i dále za tento bod, začne rychlost odesílání odpovědí od serveru klesat a čas zpracovávání dotazů prudce stoupat. Jak postupně narůstá čas zpracovávání dotazů, začínají se objevovat chyby z důvodu vypršení timeoutu na straně klienta. Závislost odeslaných dotazů a chyb je například zobrazena na grafu v obrázku 6.2.

Testování jsem prováděl pomocí programu *httperf* a vlastního skriptu. Kód skriptu je v příloze A.

Skript spouští *httperf* pokaždé s jiným nastavením *–rate* a to v rozsahu a kroku, který určíme. Skript je napsaný tak, že se test vždy provádí po stanovenou dobu. Po uplynutí tohoto časového úseku skript ukončí test a vyčká určený interval. Následně zvýší *–rate* o zvolený krok a začne znovu testovat.

Skript má následující volby:

-s Jméno nebo IP adresa testovaného serveru.

-u Cesta k testovanému souboru. Výchozí hodnota je /.

-i Minimální *–rate*, se kterým se začne testovat. Výchozí hodnota je 1000.

-a Maximální *–rate*, se kterým se ukončí testování. Výchozí hodnota je 3000.

-r Krok, o který se bude *–rate* zvyšovat. Výchozí hodnota je 100.

-c Počet dotazů, které se odešlou při každém spojení. Tato volba odpovídá *–num-calls* viz 4.2.2 .Výchozí hodnota je 1.

-t Doba v sekundách, po kterou se bude provádět jedno testování. Výchozí hodnota je 30.

-w Doba v sekundách, která udává dobu čekání mezi jednotlivými testy. Výchozí hodnota je 30.

-x Timeout, jde o maximální dobu, kterou bude *httperf* čekat na reakci od serveru. Výchozí hodnota je 10.

-h Vypíše nápovědu.

Na standardní výstup se vypisuje čas, kdy byl *httperf* spuštěn, s jakým *–rate* byl spuštěn, jaká byla skutečná rychlost odesílání dotazů, rychlost odpovědí, čas než přišla odpověď, čas připojování a počty chyb, které při testování nastaly. Pomocí programu *gnuplot* lze poté lehce vygenerovat patřičné grafy, které přehledně ukáží, jak byl systém zatížen.

Jako ukazatele vytížení systému jsem zvolil počet přijatých odpovědí za sekundu, průměrnou dobu zpracování dotazu a počet chyb. Tyto hodnoty jsem pomocí programu *gnuplot* vynesl do grafu. Skript pro generování grafů je v příloze B.

Kapitola 6

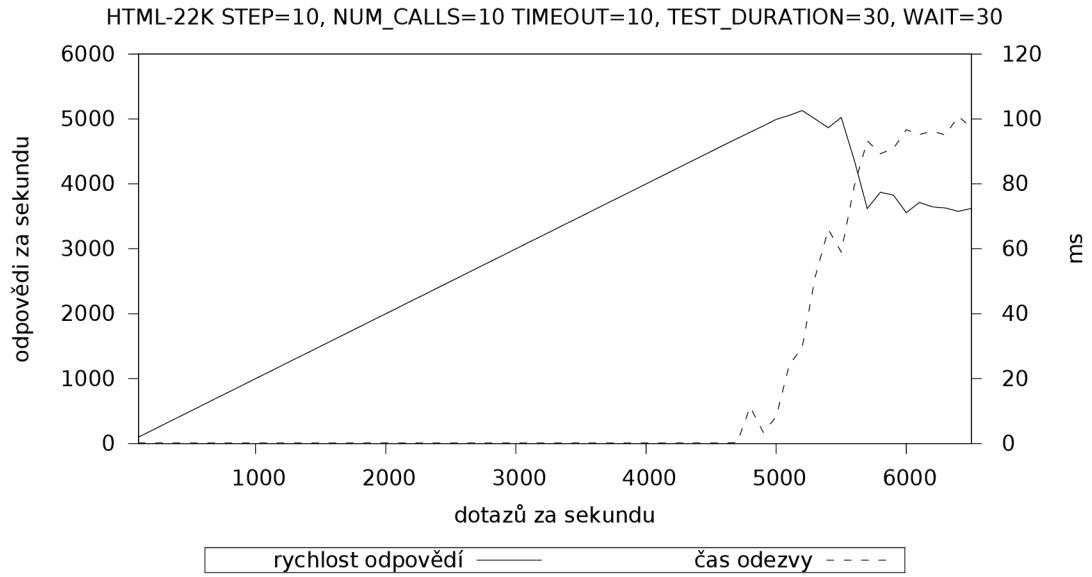
Testy

Pro veškeré zde provedené testy byl jako server použit počítač s procesorem AMD X2 (2,2 GHz), 2 GB RAM a diskem 7200RPM 16MB cache připojeným přes rozhraní SATA II. Jako operační systém byl zvolen Archlinux s jádrem 2.6.29-ARCH. Pokud u testu není uvedeno jinak, byly použity standardní balíčky ze systému v posledních stabilních verzích s výchozím nastavením. Konkrétně: Apache 2.2.11-3, MySQL 5.1.34-1, PHP 5.2.9-3.

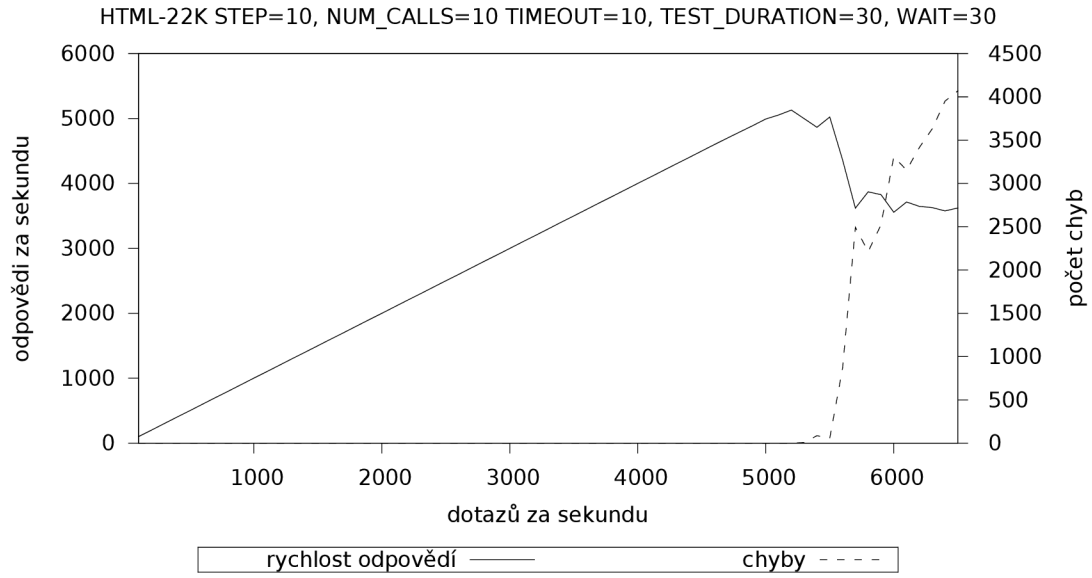
6.1 Jeden statický HTML soubor

Pro první test jsem zvolil pouze statický html soubor. Soubor obsahoval html kód stránek www.vutbr.cz, o velikosti 22KiB. Tento test prověří hrubý výkon HTTP serveru Apache.

Výstup testu v podobě grafu je na obrázcích 6.1 a 6.2. Pokud server poskytuje převážně statické soubory, největší zatížení je kladeno na CPU a disk. Nemalá část výkonu CPU je spotřebována na navazování a údržbu TCP spojení. Proto se ve velké míře projeví použití KeepAlive u HTTP 1.1. Poměrně velkého výkonu bylo dosaženo díky tomu, že při dotazu na jeden soubor byla využita cache paměť serveru Apache. Server následně nemusí při každém dotazu znovu načítat soubor z disku, ale uloží si ho do paměti cache a odtud bere obsah souboru.



Obrázek 6.1: Graf výsledku testování na statický html soubor o velikosti 12 KiB



Obrázek 6.2: Graf výsledku testování na statický html soubor o velikosti 12 KiB

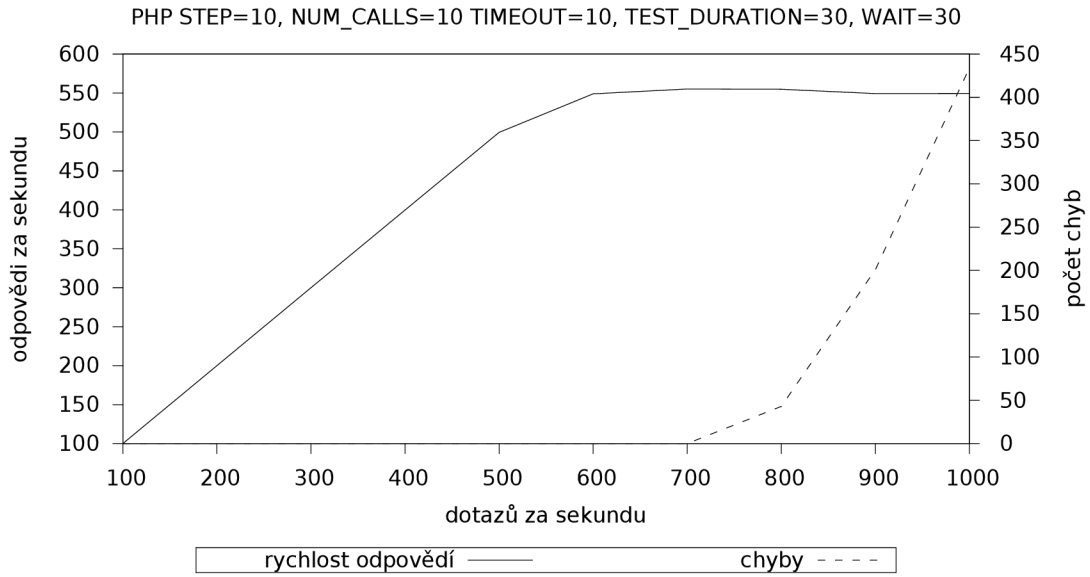
6.2 PHP skript bez práce s databází

Pro testování výkonu samotného PHP jsem zvolil skript, který v cyklu generuje velké množství náhodných čísel.

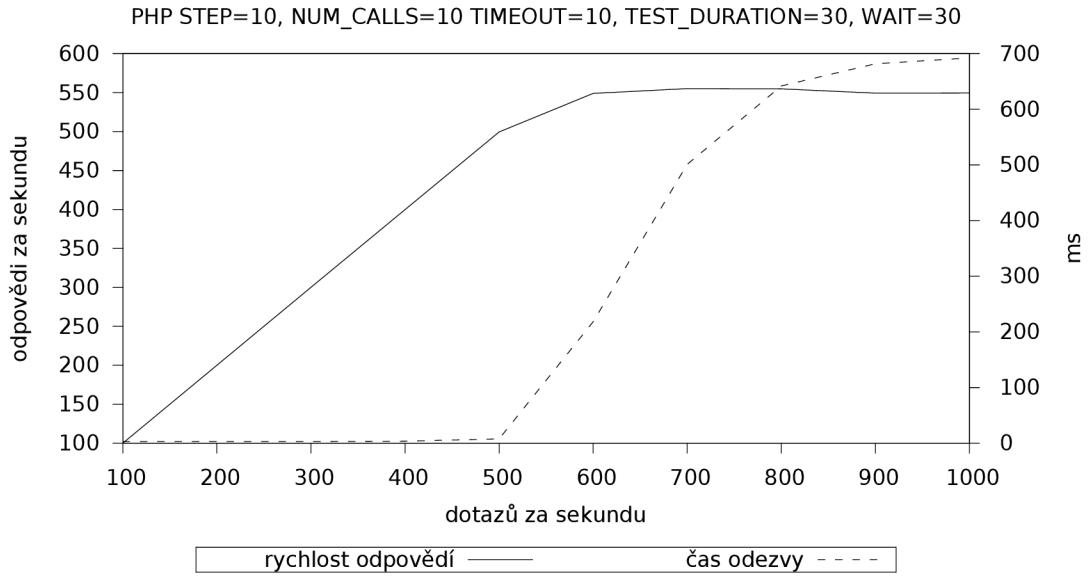
Zdrojový kód generující náhodná čísla

```
<?php
$start = microtime(true);

class test {
  function get_random() {
    return rand(0,100);
  }
  function generate_numbers($size) {
    $res = "";
    for ($i=0 ; $i<=$size ; $i++) {
      $res = $res . $this->get_random();
    }
    return $res;
  }
}

$t = new test();
$r = $t->generate_numbers(1000);
echo $r;

$end = microtime(true);
echo"\n<hr>\n";
echo $end-$start;
?>
```

Výsledek testu je na obrázku 6.3 a 6.4. U výše uvedeného skriptu je největší zatížení kladeno na CPU. Víme-li, že systém bude převážně zpracovávat podobně výpočetně náročnější skripty je dobré věnovat pozornost při výběru dostatečně výkonného procesoru. Protože má každá stránka jiný obsah, nevyužije se cache HTTP serveru.



Obrázek 6.3: Graf výsledku testování na PHP skript



Obrázek 6.4: Graf výsledku testování na PHP skript

6.3 PHP skript s prací s MySQL

Další test jsem již provedl na PHP skript, který pracoval s MySQL databází. Skript se připojí k databázi, následně z tabulky test1 vybere náhodný řádek a nakonec zapíše do tabulky test2 čas spuštění, čas ukončení a dobu zpracování celého skriptu.

Definice tabulek v jazyku SQL

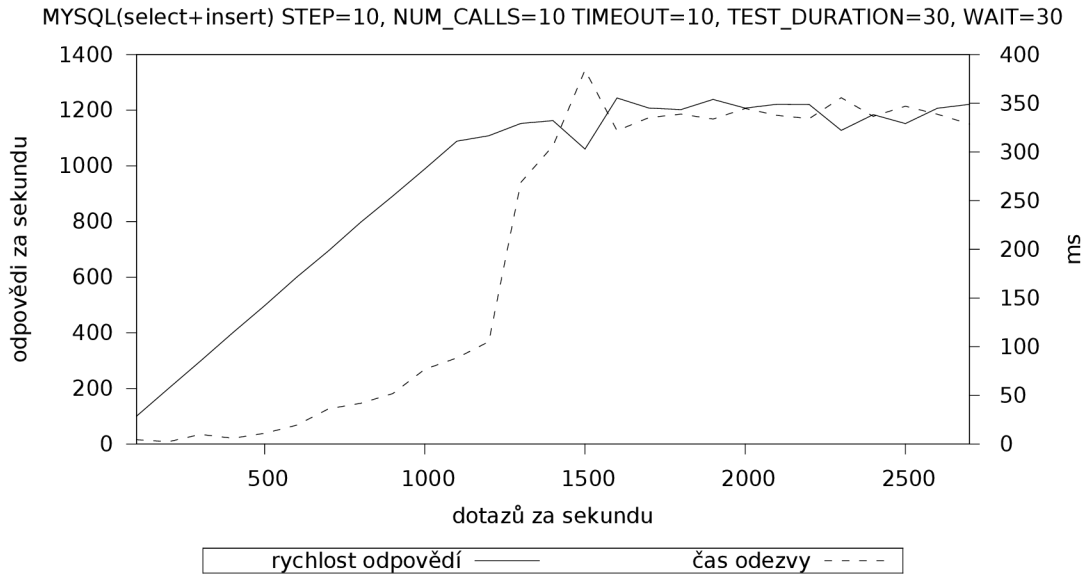
```
CREATE TABLE ‘test1‘ (
  ‘id‘ int(3) NOT NULL AUTO_INCREMENT,
  ‘name‘ varchar(160) NOT NULL,
  PRIMARY KEY (‘id‘)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

CREATE TABLE ‘test2‘ (
  ‘id‘ int(6) NOT NULL AUTO_INCREMENT,
  ‘start‘ double NOT NULL,
  ‘stop‘ double NOT NULL,
  ‘diff‘ double NOT NULL,
  PRIMARY KEY (‘id‘)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

Kód skriptu, na který probíhal test

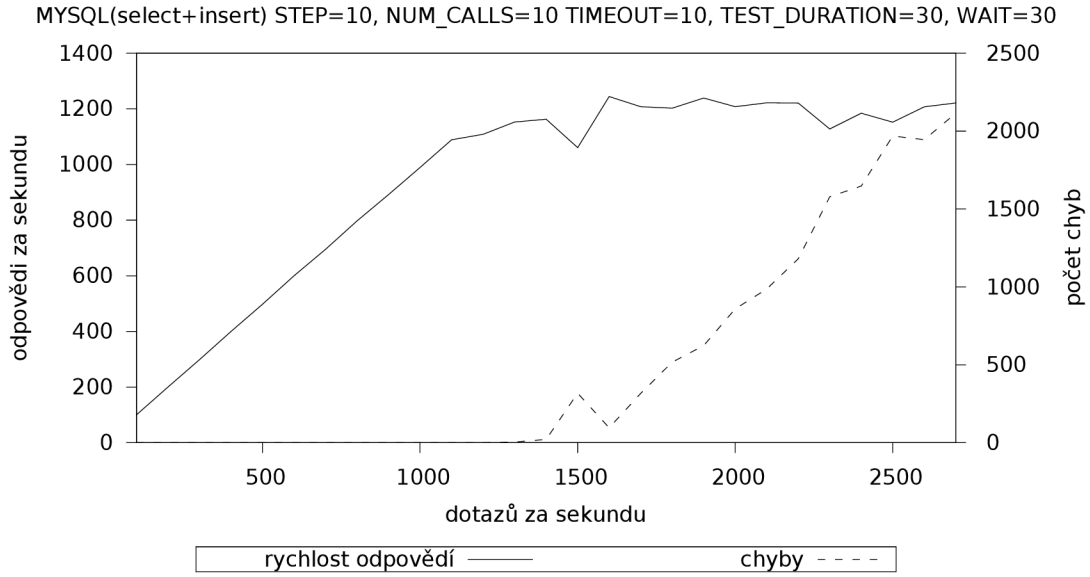
```
<?php
$start = microtime(true);

$mysql_user="root";
$mysql_pass="heslo";
$mysql_host="localhost";
$mysql_db="test";

$conn = mysql_connect($mysql_host,$mysql_user,$mysql_pass);
$db = mysql_select_db($mysql_db,$conn);

$random_id = rand(1,26);

$sql = "SELECT * FROM test1 WHERE id=$random_id";
$res = mysql_query($sql);
$row = mysql_fetch_assoc($res);
echo "$row[id] - $row[name]";

$stop = microtime(true);

$diff = $stop - $start;
$sql = "INSERT INTO test2 (start,stop,diff) VALUES (’$start’, ’$stop’,’$diff’)";
$res = mysql_query($sql);
?>
```

Výstup testu v podobě grafu je znázorněn na obrázcích 6.5, 6.6. Zde je vidět, že oproti statickému souboru došlo k značnému poklesu výkonu. V porovnání k testu v kapitole 6.2 bylo dosaženo téměř dvojnásobného výkonu. Zpomalení skriptu bylo hlavně důsledkem dlouhého cyklu v kapitole 6.2. Generování jedné stránky s 1000 náhodných čísel trvalo 0.002 sekundy. U tohoto skriptu, který pracuje s MySQL databází, trvá generování jedné stránky 0.0005 sekund. V délce čekání na výsledek PHP skriptu je tedy rozdíl jednoho řádu.

HTTP server nemusí tak dlouho čekat na výsledek PHP skriptu a dokáže s podstatně větší rychlostí odesílat odpovědi.



Obrázek 6.5: Graf výsledku testování na php skript se čtením a zápisem do MySQL

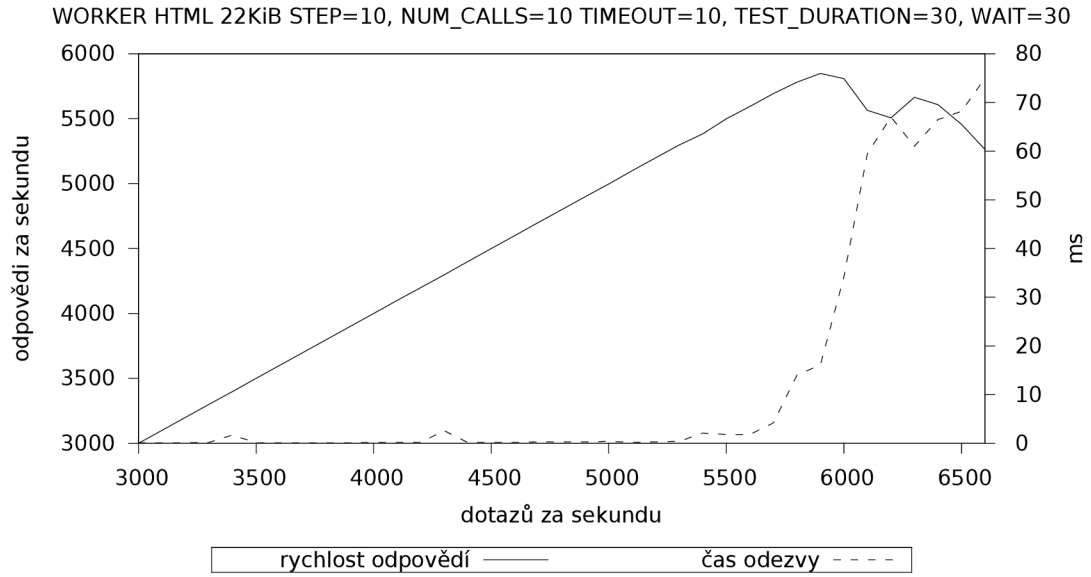


Obrázek 6.6: Graf výsledku testování na php skript se čtením a zápisem do MySQL

6.4 Použití MPM worker

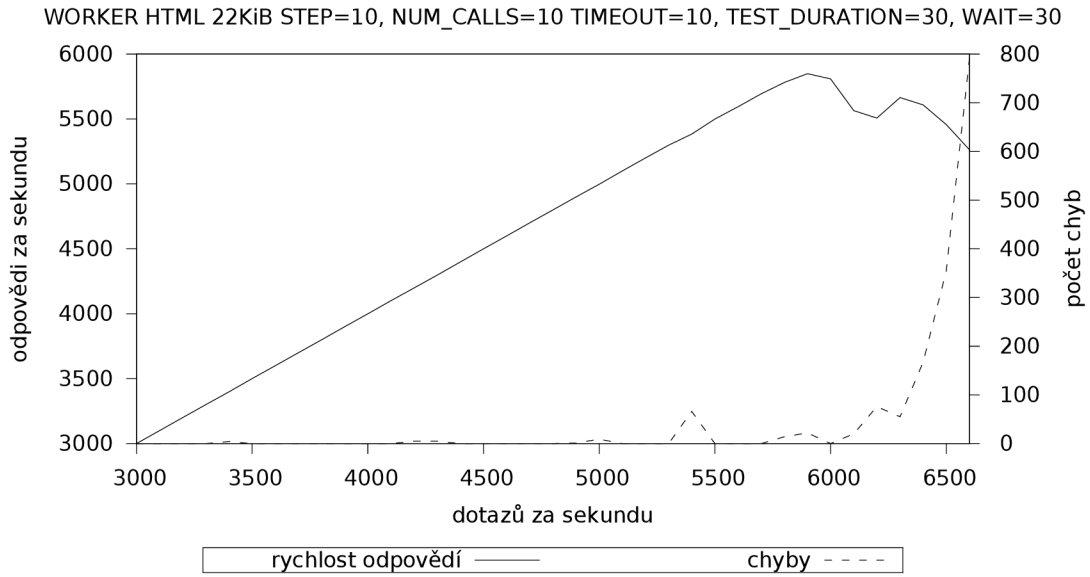
Pro testy v této kapitole jsem překompiloval Apache tak, aby využíval vlákna pomocí modulu MPM worker. Pro vytvoření nového balíčku jsem využil Archlinux ABS[1]. Kromě přidání volby *–with-mpm=worker* jsem veškeré ostatní přepínače a nastavení nechal na původních hodnotách. Jelikož není doporučené při použití MPM worker využívat modulu *libphp* (zdůvodnění je v PHP FAQ [9]), byl pro zprovoznění PHP použit modul *mod_fastcgi*. Při aplikaci toho to způsobu se PHP spouští pomocí FastCGI a místo sdílení paměťového prostoru s Apachem běží ve svém vlastním.

Statický HTML soubor



Obrázek 6.7: Graf výsledku testování na statický soubor s konfigurací Apache využívající MPM worker

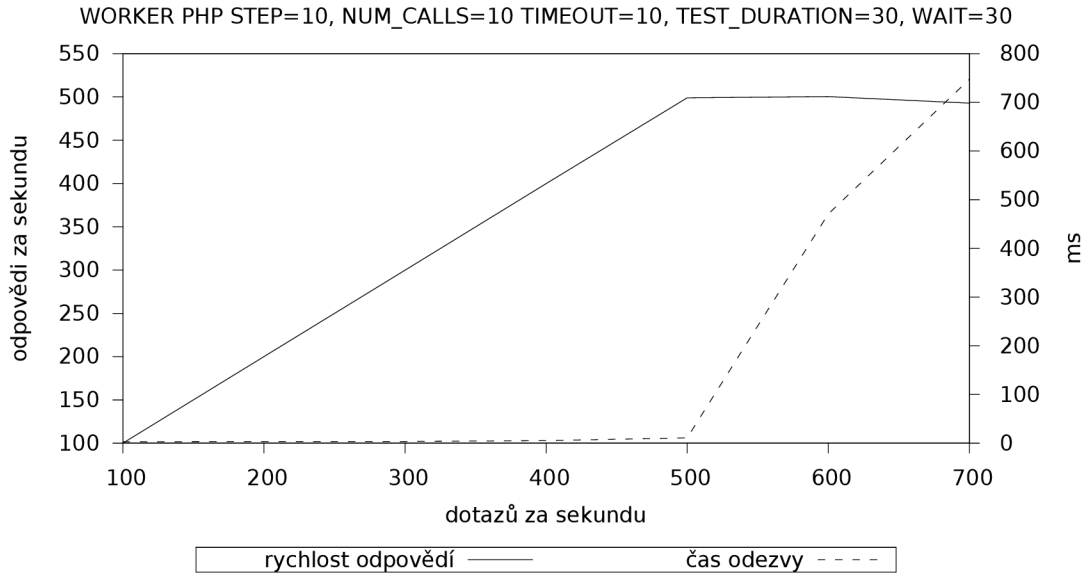
[1] Arch Build System - http://wiki.archlinux.org/index.php/ABS



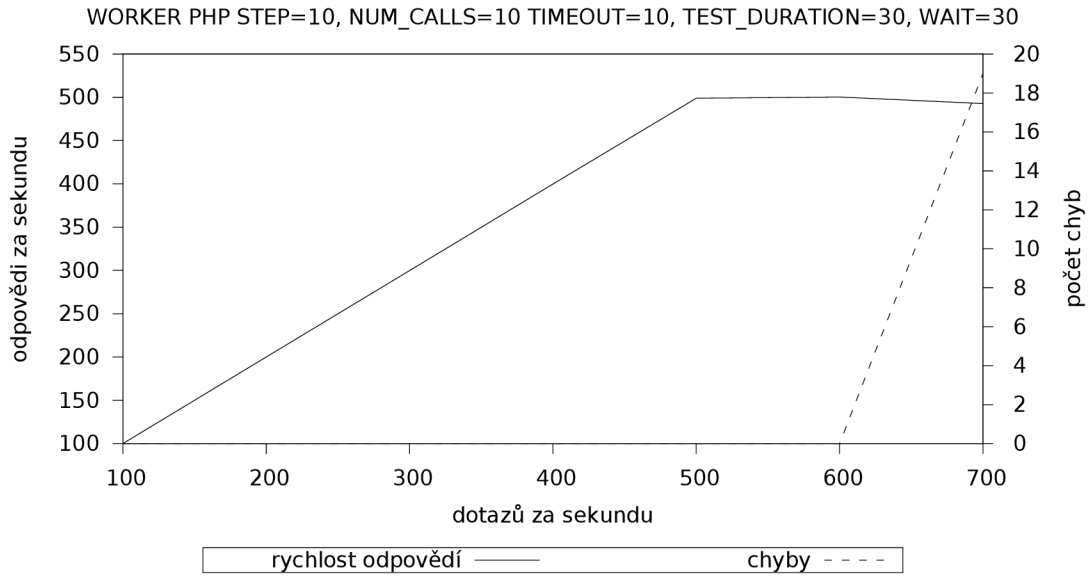
Obrázek 6.8: Graf výsledku testování na statický soubor s konfigurací Apache využívající MPM worker

Protože MPM worker zpracovává dotazy pomocí vláken, která méně zatěžují systém, byl schopen zpracovat téměř o 1000 dotazů za sekundu více než při použití MPM prefork.

PHP skript bez práce s databází



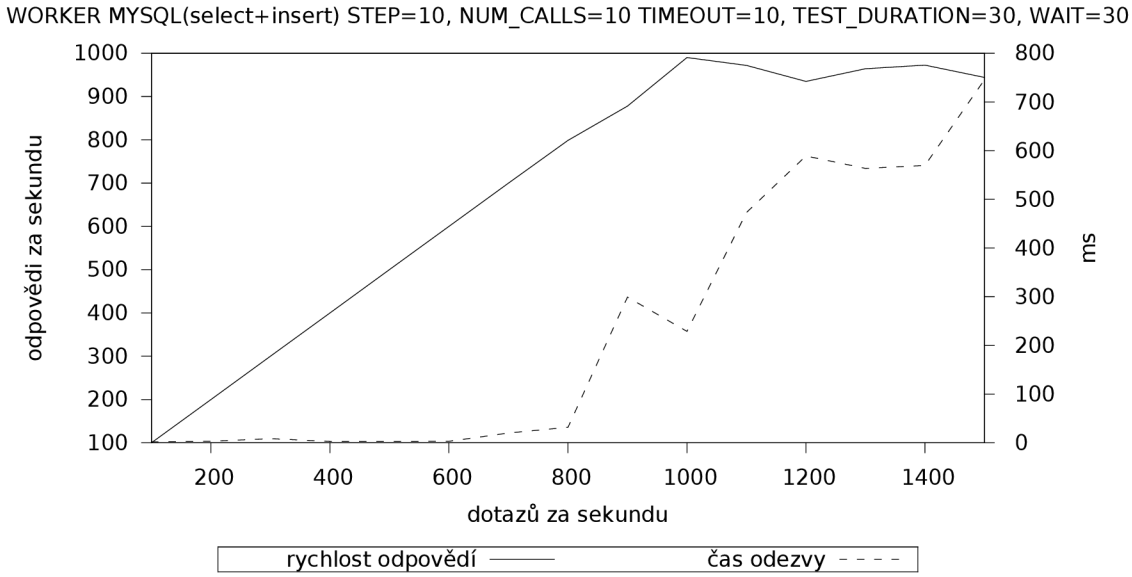
Obrázek 6.9: Graf výsledku testování na PHP skript při využití MPM worker



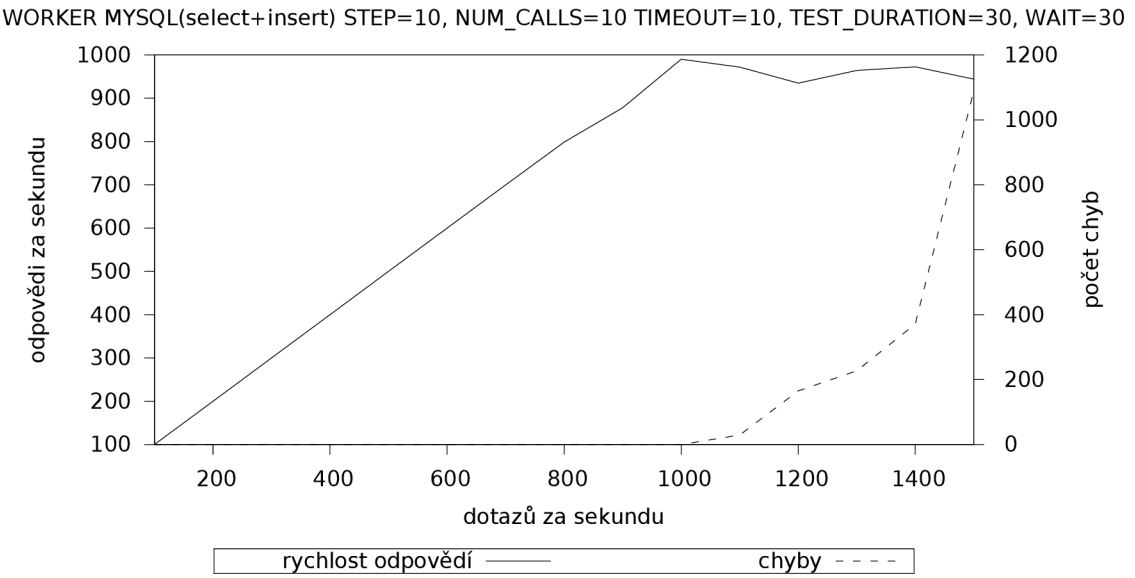
Obrázek 6.10: Graf výsledku testování na PHP skript při využití MPM worker

Pokud tyto grafy porovnáme s výsledky testování při použití modulu prefork (obrázky 6.3 a 6.4), je vidět, že nedošlo k žádnému nárůstu ani k poklesu výkonu. Jak bylo již vysvětleno v kapitole 6.2. při tomto testu se netestuje výkon HTTP serveru, ale převážně výkon PHP. Výkon PHP zůstal stejný.

PHP skript s prací s databází



Obrázek 6.11: Graf výsledku testování na php skript se čtením a zápisem do MySQL při využití MPM worker



Obrázek 6.12: Graf výsledku testování na php skript se čtením a zápisem do MySQL při využití MPM worker

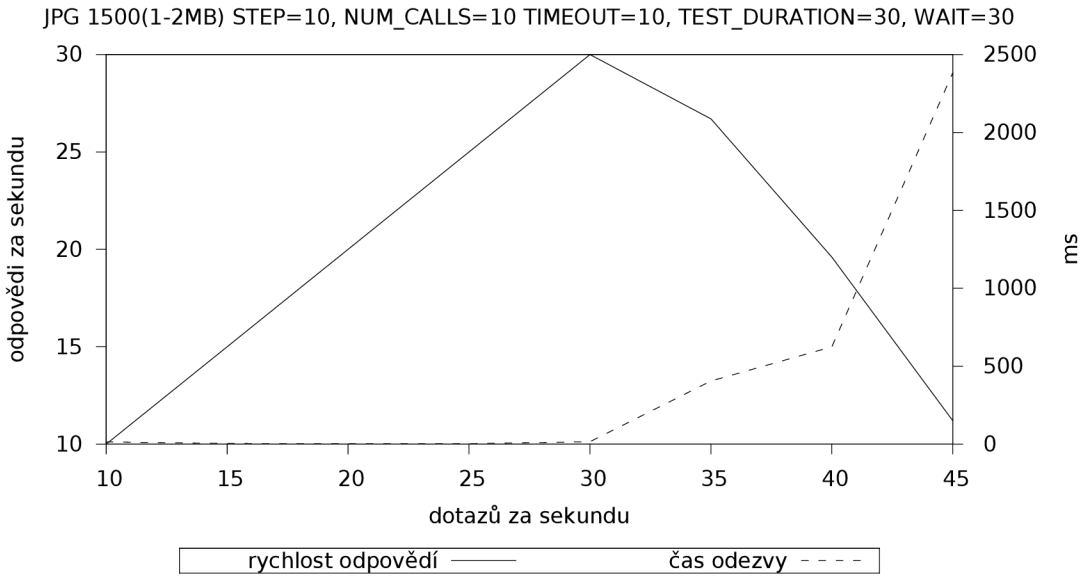
Na tomto testu se projevila menší nevýhoda PHP spouštěného pomocí FastCGI. Při použití FastCGI se dotazy zpracovávají v samostatných PHP procesech. Díky tomu je zde dosaženo

lehce menšího výkonu, než při použití *libphp*.

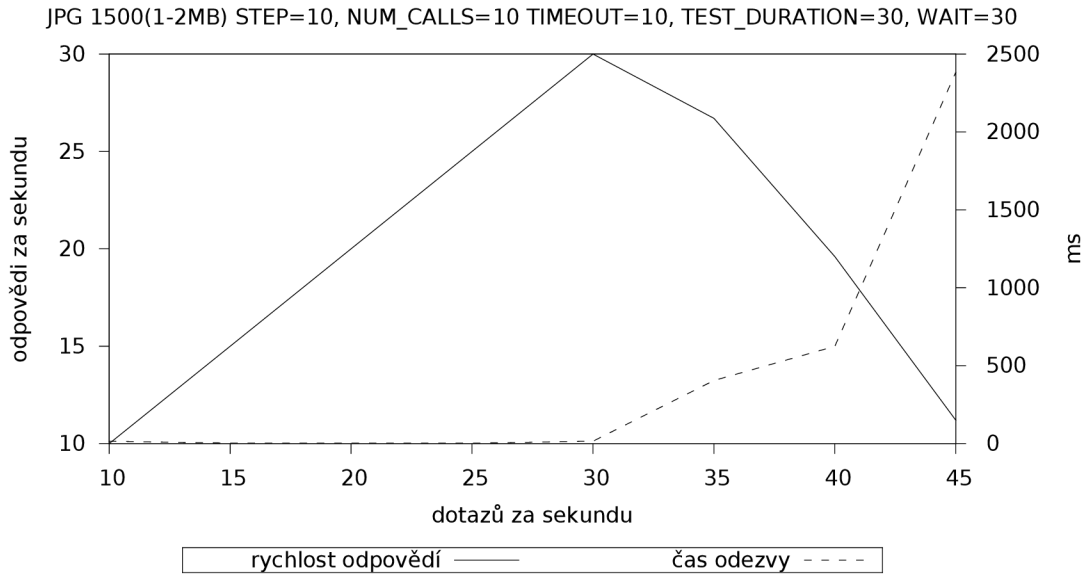
6.5 Dotazy na více souborů

Při tomto testu bylo využito volby *–wlog* programu *httperf*. Test se stále opakovaně dotazoval na celkem 1500 souborů typu JPEG. Všechny soubory byly v jednom adresáři, jehož celková velikost byla 2.3 GiB. Velikosti jednotlivých souborů byly v rozmezí 1-2 MiB.

Výsledek toho testu je na obrázcích 6.13 a 6.14.



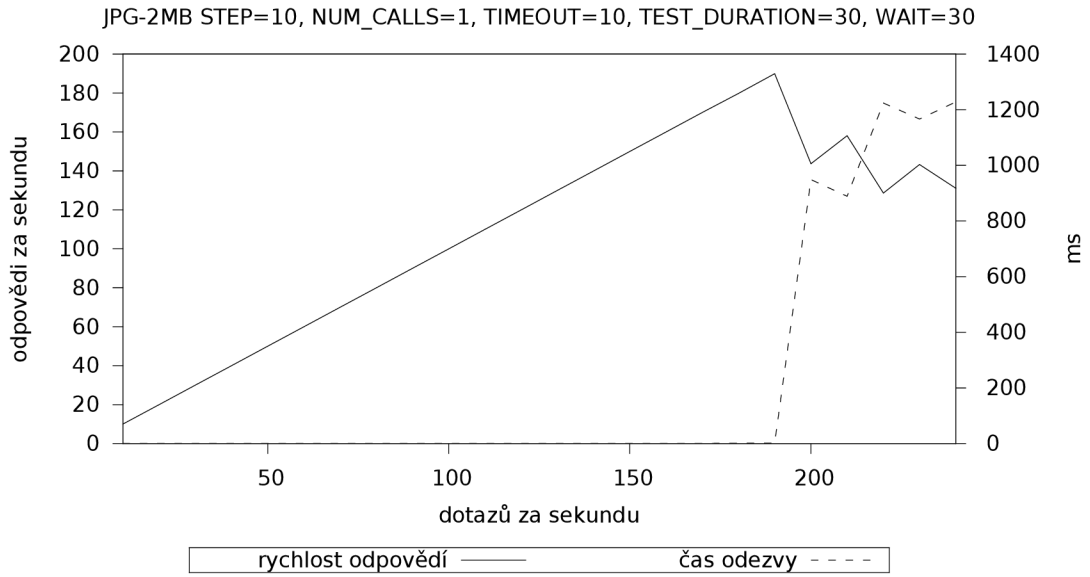
Obrázek 6.13: Graf výsledku testování na více různých obrázků



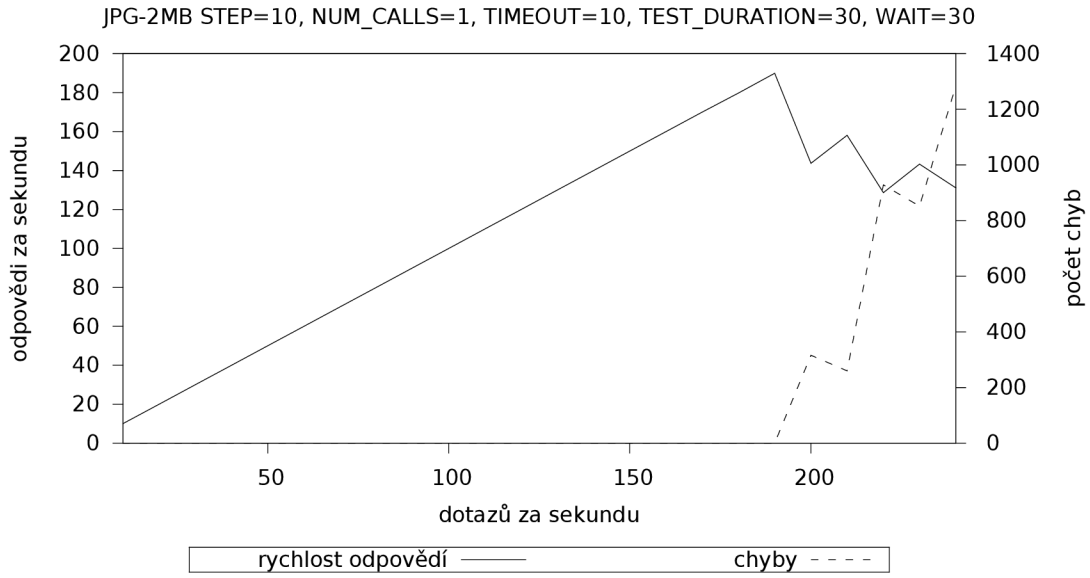
Obrázek 6.14: Graf výsledku testování na více různých obrázků

Na grafech je vidět, že v tomto případě je výkon Apache proti ostatním provedeným testům značně malý. Pokud uvažujeme průměrnou velikost souboru 1,5 MiB, tak při rychlosti 30 dotazů za sekundu je potřeba číst data z pevného disku minimální rychlostí 45 MiB/s. Při tomto testu je tedy kladena zátěž především na pevný disk, který je zde úzkým místem.

Pro porovnání byl proveden test podobný testu 6.1, ale dotazy byly odesílány na největší ze souborů ze seznamu, který byl použit u testu výše. Tento soubor měl 2 MiB. Na výsledcích tohoto testu (obrázky 6.15 a 6.16) je vidět, že bylo dosaženo mnohonásobně většího výkonu. Tento nárůst je důsledkem cachovacího algoritmu uvnitř severu Apache, jenž je schopen uložit do paměti soubor, který je často požadován.



Obrázek 6.15: Graf výsledku testování na obrázek



Obrázek 6.16: Graf výsledku testování na obrázek

Kapitola 7

Optimalizace LAMP

V této kapitole jsou uvedena některá obecná doporučení pro optimalizaci systému LAMP a dosažení vyššího výkonu.

7.1 Apache

- `HostnameLookups Off` - Vypne vyhledávání DNS. Pokud je tato volba zapnutá, Apache pro každý dotaz vyhledá záznam v DNS pro danou IP. Toto vyhledávání může negativně ovlivnit výkon serveru.
- `Options FollowSymLinks` - Povolí procházení symbolických odkazů. Pokud je tato volba zapnutá, Apache u každého souboru prověří, zda nejde o symbolický odkaz. Bohužel, toto nastavení má i bezpečnostní rizika. Pokud se v DocumentRoot objeví odkaz na nějaký soubor mimo DocumetnRoot, bude k němu pomocí tohoto odkazu umožněn přístup. Je proto dobré zamezit výskytu symbolických odkazů v DocumentRoot bez našeho vědomí.
- `AllowOverride None` - Zakáže používání souboru *.htaccess*. Pokud tuto hodnotu povolíme, Apache musí při každém dotazu kontrolovat, zda se v adresáři vyskytuje soubor *.htaccess* s nastavením. Tento soubor se vyhledává v každém adresáři od kořenového až po cílový. Proto pokud je to možné je dobré veškerá nastavení uložit do hlavního konfiguračního souboru.
- `Keep Alive On` - Povolí provádění více dotazů během jednoho TCP spojení. Pomocí `KeepAliveTimeout` lze určit kolik sekund se bude čekat na další dotaz než se spojení uzavře. Pomocí `MaxKeepAliveRequests` lze určit maximální počet dotazů během jednoho spojení. Pokud je nastaveno na hodnotu 0, bude povoleno neomezené množství dotazů.
  [2]

7.2 PHP

V *php.ini* by měly být aktivní jen ty moduly, které jsou opravdu ve skriptech využívány. V tabulce 7.1 jsou čtyři nejdůležitější volby, které řídí využití systémových prostředků interpretem PHP.

volba	popis	doporučená hodnota
`max_execution_time`	Kolik sekund procesorového času může být skript maximálně zpracováván.	30
`max_input_time`	Jak dlouho bude skript maximálně čekat na vstupní data.	60
`memory_limit`	Kolik paměti (v bytech) může skript využít než bude ukončen.	32M
`output_buffering`	Kolik dat (v bytech) se uloží do paměti (buffer) něž se odešlou klientu.	4096

Tabulka 7.1: Volby v php.ini, podle kterých se přidělují systémové prostředky

Nastavení těchto voleb závisí především na povaze konkrétní PHP aplikace. Například pokud je potřeba, aby uživatelé mohli nahrávat velké soubory, je dobré navýšit `max_input_time`. [3]

Často využívanou možností, jak urychlit zpracování PHP skriptů, je použití Truck MMCache, nebo některého z podobných produktů (Zend Performance Suite, PHP Accelerator, Alternative PHP Cache, AfterBurner Cache). Truck MMCache je open source PHP akcelerátor, optimalizér a dynamická cache pro PHP. Při zavolání PHP skriptu jej interpret PHP zkompiluje a následně spustí, ale skript ve zkompilované podobě nikde neuchová. Truck MMCache uchovává skripty ve zkompilované podobě. Skripty není třeba pokaždé znovu kompilovat. Díky tomuto mechanizmu je možné snížit zatížení systému až desetkrát.[7]

7.3 MySQL

Většina LAMP aplikací pracuje s velkým množstvím opakujících se dotazů. MySQL musí pro každý dotaz provést tyto shodné operace: analyzovat dotaz, zjistit způsob provedení, nahrát data z disku a vrátit výsledek klientovi. Aby se opakující dotaz nemusel tímto způsobem provádět celý znovu, využívá MySQL funkce zvané *query cache*. V této cache si server uchovává výsledky provedených dotazů, které může opakovaně využívat. *Query cache* lze zapnout v */etc/my.conf* přidáním řádku `query_cache_size = 32M`. Pro uložení výsledků bude v tomto případě vyhrazeno 32MB.

Především u systémů, kde na jednom počítači běží MySQL společně s Apachem, je dobré nastavit následující limity. Tyto limity zamezí případu, kdy by samotné MySQL mohlo vytížit celý systém.

- `set-variable=max_connections=500` Maximální počet současně otevřených spojení.
- `set-variable=wait_timeout=10` Čas v sekundách, po jehož uplynutí se ukončí neaktivní spojení.

Pro případnou optimalizaci prováděných dotazů je vhodné zapnout zaznamenávání pomalých dotazů.

```
[mysqld]
; enable the slow query log, default 10 seconds
log-slow-queries
; log queries taking longer than 5 seconds
```

```
long_query_time = 5
; log queries that don't use indexes even if they take less than long_query_time
; MySQL 4.1 and newer only
log-queries-not-using-indexes
```

Toto nastavení zaznamená všechny dotazy, které budou trvat déle než 5 sekund a také dotazy nevyužívající indexy. Pro čtení tohoto logu je nevhodnější použít program *mysqldumpslow*, který zobrazí seznam pomalých dotazů, dobu jejich provádění a počet výskytů v logu. [4]

Kapitola 8

Závěr

Cílem této práce bylo prostudovat a popsat problematiku výkonu systému typu LAMP a shromáždit dosažené poznatky do metodiky, případně do sady nástrojů. Po nastudování několika programů jsem se rozhodl použít jeden z nich a rozšířit ho o vlastní skript, který řídí celé testování. Tento skript byl následně použit k provedení sady testů. Testy ukázaly, jak se měnil výkon podle typu zatížení a podle použitého MPM. V práci je vysvětlen rozdíl výkonu mezi testováním na jeden soubor a na více souborů. Následně byla shrnuta obecná doporučení pro optimalizace systémů LAMP.

Práci je možno rozšířit o další varianty testů. V těchto testech by například bylo možno zkrátit čekací doby mezi jednotlivými kroky testu. Nebo lze provést takové varianty testů, které by kombinovaly různé typy zátěže v jednom testu.

Literatura

[1] D. Mosberger, T. J.: httperf: A Tool for Measuring Web Server Performance. http://www.hpl.hp.com/research/linux/httperf/wisp98/httperf.pdf, 1998.

[2] Kabir, M. J.: *Apache Server 2*. Computer Press, 2004, ISBN 80-251-0319-6.

[3] Walberg, S. A.: Tuning LAMP systems, Part 2: Optimizing Apache and PHP. http://www.ibm.com/developerworks/linux/library/l-tune-lamp-2.html, 2007-03-30, [cit. 2009-05-10].

[4] Walberg, S. A.: Tuning LAMP systems, Part 3: Tuning your MySQL server. http://www.ibm.com/developerworks/linux/library/l-tune-lamp-3.html, 2007-06-07, [cit. 2009-05-10].

[5] WWW stránky: Apache MPM prefork. http://httpd.apache.org/docs/2.2/mod/prefork.html.

[6] WWW stránky: Apache MPM worker. http://httpd.apache.org/docs/2.2/mod/worker.html.

[7] WWW stránky: httperf and File Descriptors. http://turck-mmcache.sourceforge.net/index_old.html, 2001, [cit. 2009-05-10].

[8] WWW stránky: httperf and File Descriptors. http://gom-jabbar.org/articles/2009/02/04/httperf-and-file-descriptors, 2009-02-04, [cit. 2009-05-01].

[9] WWW stránky: PHP: Installation - Manual. http://www.php.net/manual/en/faq.installation.php#faq.installation.apache2, 2009-05-15, [cit. 2009-05-15].

[10] WWW stránky: ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/trunk/programs/ab.html, 2009, [cit. 2009-04-04].

[11] WWW stránky: Apache HTTP Server — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Apache_HTTP_Server&oldid=282349531, 2009, [cit. 2009-04-04].

[12] WWW stránky: httperf [online]. http://www.hpl.hp.com/research/linux/httperf, 2009, [cit. 2009-04-04].

[13] WWW stránky: LAMP (software bundle) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=LAMP_(software_bundle)&oldid=281479219, 2009, [cit. 2009-04-04].

[14] WWW stránky: Linux — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Linux&oldid=281567181, 2009, [cit. 2009-04-04].

[15] WWW stránky: Siege [online]. http://www.joedog.org/index/siege-home, 2009, [cit. 2009-04-04].

Příloha A

Skript test.sh

```
#!/bin/bash
help()
{
cat << EOF
usage: $0 options

OPTIONS:
   -s   server
   -u   uri
   -i   rate min
   -a   rate max
   -r   rate step
   -c   num calls
   -t   test time
   -w   wait time
   -x   timeout
   -h   help
EOF
}

#vychozi hodnoty
URI="/"
RATE_MIN=1000
RATE_MAX=3000
RATE_STEP=100
TIMEOUT="10"
TEST_DURATION="30"
WAIT="30"
NUM_CALLS="1"

TMP_FILE="/tmp/httperf-$RANDOM"

#zpracovani argumentu
while getopts "s:u:i:a:r:t:w:x:c:h" OPTION
do
  case $OPTION in
    h)
      help
      exit 1
      ;;
    s)
      HOST=$OPTARG
      ;;
```

```
    u)
      URI=$OPTARG
      ;;
    i)
      RATE_MIN=$OPTARG
      ;;
    a)
      RATE_MAX=$OPTARG
      ;;
    r)
      RATE_STEP=$OPTARG
      ;;
    t)
      TEST_DURATION=$OPTARG
      ;;
    w)
      WAIT=$OPTARG
      ;;
    x)
      TIMEOUT=$OPTARG
      ;;
    c)
      NUM_CALLS=$OPTARG
      ;;

    ?)
       help
       exit
      ;;
  esac
done

#povinne paramtetry
if [[ -z $HOST ]]
then
     help
     exit 1
fi

echo "#START"
echo "#HOST=$HOST, URI=$URI, RATE_MIN=$RATE_MIN, RATE_MAX=$RATE_MAX, RATE_STEP=$RATE_STEP,\
 NUM_CALLS=$NUM_CALLS TIMEOUT=$TIMEOUT, TEST_DURATION=$TEST_DURATION, WAIT=$WAIT"
echo "#TimeStart DemandedReqRate RealReqRate ReplyRate ReplyRateMax ReplyRateMin\
 ReplyRateStDev ReplyTime ConnTime ConnTimeMax ConnTimeMin ConnTimeStDev Errors_total\
  Errors_client-timo Errors_socket-timo Errors_connrefused Errors_connreset Errors_fd-unavail\
  Errors_addrunavail Errors_ftab-full Errors_other"

for RATE in `seq $RATE_MIN $RATE_STEP $RATE_MAX`;
do
  echo -n "`date +'%s'` $RATE "

  #vypocitani kolik musi probehnout spojeni pri maximalnim rate aby test trval 30s
  NUM_CONNS=$(($RATE_MAX*$TEST_DURATION))

  #spusteni testu ulozeni vysledku do docasneho souboru
  ~/bin/httperf --hog --server=$HOST --uri=$URI --num-conns=$NUM_CONNS \
    --rate=$RATE --num-calls=$NUM_CALLS --timeout=$TIMEOUT > $TMP_FILE &

  #pid httperf procesu beziciho na pozadi
```

```
  HTTPERF_PID=$!

  #ukonceni testu po danem case
  sleep ${TEST_DURATION}s
  kill -SIGINT $HTTPERF_PID

  #cekani na uklidneni systemu
  sleep ${WAIT}s

  #nasilne ukonceni pokud stale bezi
  kill $HTTPERF_PID 2> /dev/null

  #vypis vysledku
  RES_CONN_RATE="`cat $TMP_FILE | grep 'Connection rate' | awk '{print $3}'`"

  RES_REP_RATE="`cat $TMP_FILE | grep 'Reply rate' | awk '{print $7}'`"
  RES_REP_RATE_MAX="`cat $TMP_FILE | grep 'Reply rate' | awk '{print $9}'`"
  RES_REP_RATE_MIN="`cat $TMP_FILE | grep 'Reply rate' | awk '{print $5}'`"
  RES_REP_RATE_STD="`cat $TMP_FILE | grep 'Reply rate' | awk '{print $11}'`"

  RES_REP_TIME="`cat $TMP_FILE | grep 'Reply time' | awk '{print $5}'`"

  RES_CONN_TIME="`cat $TMP_FILE | grep 'Connection time \[ms\]: min' | awk '{print $7}'`"
  RES_CONN_TIME_MIN="`cat $TMP_FILE | grep 'Connection time \[ms\]: min' | awk '{print $5}'`"
  RES_CONN_TIME_MAX="`cat $TMP_FILE | grep 'Connection time \[ms\]: min' | awk '{print $9}'`"
  RES_CONN_TIME_STD="`cat $TMP_FILE | grep 'Connection time \[ms\]: min' | awk '{print $13}'`"

  ERROR_1="`cat $TMP_FILE | grep  'Errors: total' | awk '{print $3, $5, $7, $9, $11}'`"
  ERROR_2="`cat $TMP_FILE | grep  'Errors: fd-unavail' | awk '{print $3, $5, $7, $9}'`"


  echo "$RES_CONN_RATE $RES_REP_RATE $RES_REP_RATE_MAX $RES_REP_RATE_MIN $RES_REP_RATE_STD\
   $RES_REP_TIME $RES_CONN_TIME $RES_CONN_TIME_MAX $RES_CONN_TIME_MIN $RES_CONN_TIME_STD\
   $ERROR_1 $ERROR_2"

done

rm $TMP_FILE
echo "#END"
```

Příloha B

Generování grafů

```
#!/usr/bin/gnuplot

datfile="log"
calls=10

set terminal pdf monochrome dashed size 15cm,8cm
set output "reply.pdf"

set autoscale x
set autoscale y
set autoscale y2
set xrange[(10*calls):(270*calls)]

set title "MYSQL(select+insert) STEP=10, NUM_CALLS=10 TIMEOUT=10, TEST_DURATION=30, WAIT=30"

set xlabel "datazů za sekundu"
set ylabel "odpovědi za sekundu"
set y2label "ms"

set ytics nomirror
set xtics nomirror
set y2tics
set tics out

set key below  box

plot datfile using ($2*calls):4 axes x1y1 with lines title "rychlost odpovědí",\
     datfile  using ($2*calls):8 axes x1y2 with lines title "čas odezvy"

#-------------------------------------------------------------------------

set output "errors.pdf"
set y2label "počet chyb"

plot datfile using ($2*calls):4 axes x1y1 with lines title "rychlost odpovědí",\
     datfile using ($2*calls):13 axes x1y2 with lines title "chyby"
```