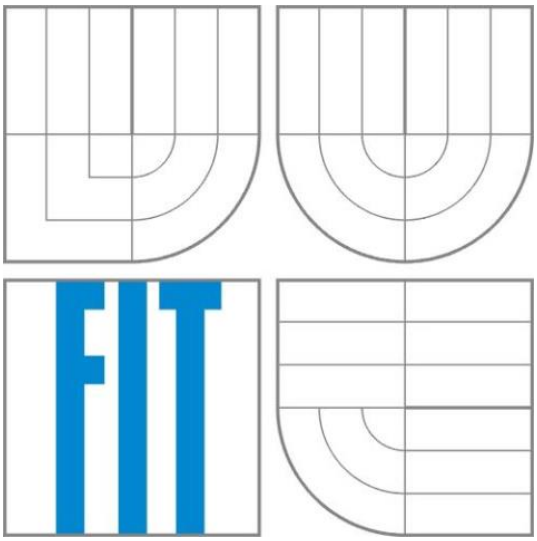
Vysoké Učení technické v Brně

Fakulta informačních technologií



Modelování a simulace

2012/2013

Projekt – SHO Počítačový server

Matěj Mareček (xmarec12) & Ondřej Kobližek (xkobli00)

Brno, 6. prosince 2012

1 Obsah

1 Úvod

V této práci je řešena implementace systému hromadné obsluhy (IMS, https://www.fit.vutbr.cz/study/courses/index.php?id=8662, slide č. 139) počítačového serveru, na kterém běží také webový a databázový server.

Na základě vytvořeného modelu a simulačních experimentů bude ukázáno chování systému při velké zátěži, běžném provozu i rozdílné konfigurace hardware. Experimenty mají za úkol odhalit slabá místa systému a ukázat, jak se zlepší odezva a celkový výkon serveru, pokud se změní parametry jednotlivých komponent počítače.

1.1 Autoři práce

- Matěj Mareček – xmarec12@stud.fit.vutbr.cz
- Ondřej Kobližek – xkobli00@stud.fit.vutbr.cz

Kromě nás dvou se na práci nepodílel nikdo jiný a ani nebylo potřeba kontaktovat jinou osobu kvůli informacím.

1.2 Validita projektu

Validita byla ověřena experimenty, jejichž výsledky ve formě grafu odpovídaly výsledkům naměřeným na referenčním počítači a to pomoci měřicích a analytických nástrojů uvedených v kapitole Původ technologií.

1.3 Zadání projektu

Oficiální zadání projekty je k dispozici na stránkách http://perchta.fit.vutbr.cz:8000/vyuka-ims/26.

„Uvažujme systém jednoho počítače s N procesory a jedním paměťovým diskem. Systém modelujte jako SHO. V počítači vznikají a zanikají uživatelské procesy. Proces příchodů (vznikání) uživatelských procesů sami navrhněte a dále navrhněte model doby jejich výpočtu a požadavky na zdroje systému (přístup do DB serveru, přístup k disku).

Kromě uživatelských procesů v systému pracuje i webový server a databázový server modelované jako zařízení s kapacitou jedna zpracovávaných požadavků. Na oba servery přichází z vnějšího okolí systému požadavky (modelujte proces příchodu a proces zpracování vhodnými rozloženími). Webový i databázový server s každým požadavkem přistupují k zařízení paměťového disku.“

Toto zadání bylo dále rozšířeno a mírně modifikováno tak, aby podle našich znalostí lépe odpovídalo realitě. Tyto modifikace jsou popsány dále v dokumentaci.

1.4 Cíle projektu

Mezi hlavní cíle při tvorbě projektu jsme si zařadili:

1. Analýza provozu za různých podmínek
2. Detekce slabého místa serveru
3. Nalezení ekonomicky výhodného řešení, které by dokázalo zvýšit výkon serveru.

V následujících kapitolách je ukázán sběr dat a zdroje informací, které byly použity pro tvorbu konceptuálního modelu systému serveru/počítače.

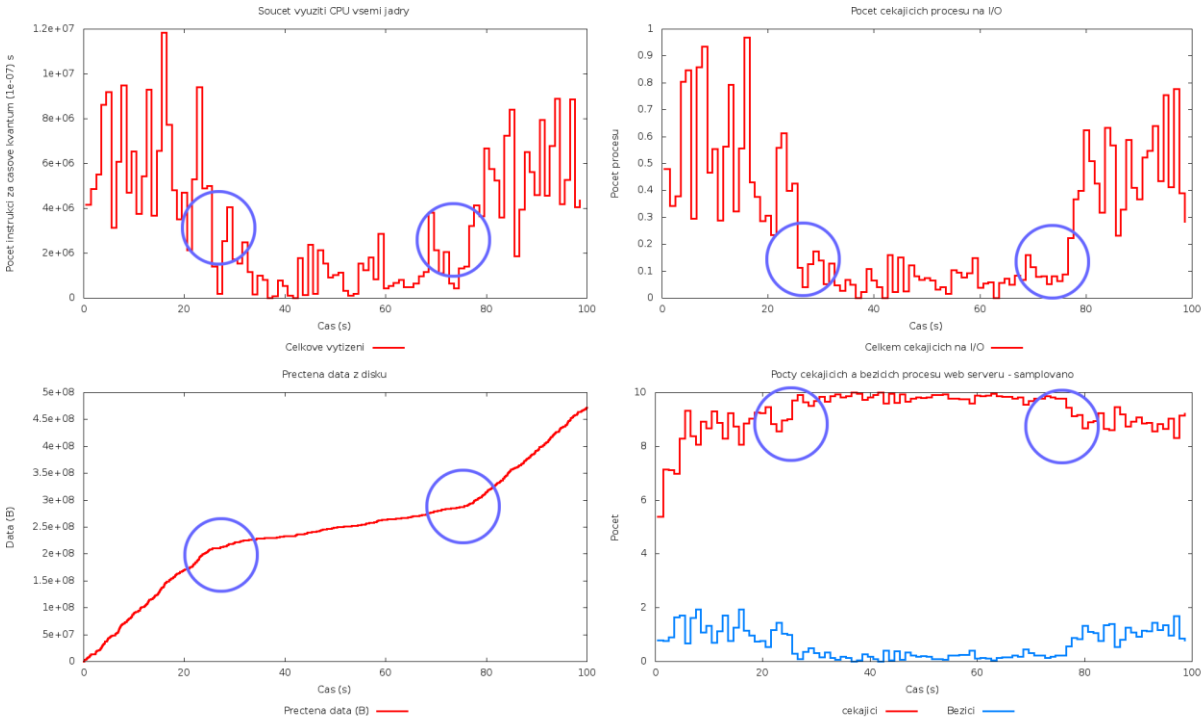
Na základě tohoto konceptuálního modelu poté vznik simulační model v jazyce C++/Simlib (http://www.fit.vutbr.cz/~peringer/SIMLIB/) na kterém jsme prováděli experimenty a testovali naše hypotézy ohledně chování systému.

V závěrech práce také naleznete, jaký má vliv na propustnost webových požadavků počet „předforkovaných“ procesů u software Apache HTTP Server (http://httpd.apache.org/, verze která má zapnutý preforking procesů) a jak důležité je mít, či nemít často používaná data v rychlé paměti/cache.

1.5 Validita a funkčnost

Projekt byl přeložen a spuštěn na serveru merlin.fit.vutbr.cz a byl shledán jakožto plně funkční.

Výsledné grafy z dat simulace potvrdily naše předpoklady a jednotlivé části korelovaly jak mezi sebou, tak s výsledky dosaženými při testování na referenčním stroji.



Obrázek 1 - výsledek experimentu, kdy byl mezi časy 25s-75s redukován počet požadavků na webový server

Na obrázku 1 zle vidět experiment při kterém byl webový server pod velkou zátěží prvních 25 sekund simulace a poté až do času 75 sekund zátěž polevila. Od času 75 až do konce simulace byly na server znovu generovány požadavky ve zvýšené míře.

Z grafů lze vypozorovat korelaci mezi počtem procesů webového serveru, které stíhaly požadavku obsluhovat, počtu volných před-vytvořených procesů web serveru, nárůstem a úbytkem množství celkových dat přečtených z disku (čte je databáze) s tím spojený počet požadavků čekající na I/O obsloužení a samozřejmě celkové zatížení procesoru (průměrné zatížení všech 8 jader CPU).

2 Rozbor tématu a použitých metod/technologií

2.1 Použité postupy

Ke tvorbě projektu byl použit jazyk C++ a simulační knihovna Simlib. Kromě toho, že tento krok byl podmíněn již samotným zadáním projektu, tak knihovna Simlib poskytuje spoustu věcí potřebných pro tvorbu simulačních modelů a kompilovaný kód C++ je poměrně rychlý.

Alternativním přístupem by bylo stáhnout zdrojové kódy nějakého virtualizačního nástroje, upravit si jej podle vlastních potřeb tak, aby zaznamenával, co se všechno se děje s virtuálním počítačem, který do něj nahrajeme a poté tyto výsledky analyzovat. Tento přístup ovšem neodpovídá zadání a byl by časově náročnější než vytvoření simulačního modelu pomocí C++/Simlib.

Pro získání parametrů hardware byly poté použity hodnoty udávané přímo výrobci (například HDD od Western Digital, http://www.wdc.com/) a měření prováděná pomocí systémových nástrojů systémů Ubuntu/Linux Mint + Windows 8 Pro (Performance Monitor, Process Explorer, CPU-Z a AIDA64 Extreme Edition) na referenčním stroji Acer Aspire 7745G (Intel Core i7 720QM, 6GB, Western Digital 640 GB SATA - 5400RPM - 8MB Cache (model: WD640BEVT-22A0RT0), http://notebooky.heureka.cz/acer-aspire-7745g-726g64mn-lx-pum02-062/specifikace/).

Pro přesné testovací výpočty a náhledy jednotlivých rozložení byl použit nástroj Wolfram Alpha. K automatickému generování grafů z výstupů simulačního modelu byl použit software gnuplot a DataSampler.

Měření rychlosti pevného disku na referenčním stroji bylo prováděno pomoci aplikace CrystalDiskMark za klidového stavu počítače.

2.2 Původ technologií

- C++ - http://en.wikipedia.org/wiki/C%2B%2B
- Simlib - http://www.fit.vutbr.cz/~peringer/SIMLIB/ (GNU LGPL)
- Windows 8 Pro x64 - http://windows.microsoft.com/cs-CZ/windows/home (akademická licence VUT - FIT)
- Linux - distribuce Ubuntu/Mint - http://www.ubuntu.com/(GNU GPL), http://linuxmint.com/(GLP)
- Process Explorer – http://technet.microsoft.com/en-US/sysinternals/bb896653 (Proprietary)
- CPU-Z - http://www.cpuid.com/softwares/cpu-z.html (Freeware)
- LAMP (software bundle) - http://en.wikipedia.org/wiki/LAMP_(software_bundle)
- AIDA64 Extreme Edition - http://www.aida64.com/product/aida64-extreme-edition/overview
- Gnuplot - http://www.gnuplot.info/ (Open source (own license))
- Wolfram Alpha - http://www.wolframalpha.com/
- CrystalDiskMark - http://crystalmark.info/software/CrystalDiskMark/index-e.html (Freeware)
- DataSampler - zdrojové kódy přibaleny v projektu (Open source)

2.3 Procesor

Jelikož CPU je esenciální komponentou počítače, bylo nutno tuto část namodelovat, tak aby co nejvíce odpovídala realitě. Bylo ovšem také nutné zvýšit míru abstrakce, už jen z toho důvodu, že

simulace takového modelu by trvala velice dlouho a oplatilo by se raději koupit požadovaný hardware. Z toho důvodu byly instrukce procesoru nahrazeny celými operacemi trvajícími tisíce až miliony cyklů (počet cyklů, které jednotlivé procesy spotřebují, byly měřeny pomoci nástroje Process Explorer v15.23).

Nejčastěji se v minulosti rychlost procesoru vyjadřovala ve frekvenci, na které jsou taktovány jeho vnitřní hodiny. Tento stěžejní údaj byl také nakonec použit jako hlavní ukazatel rychlosti procesoru i přesto, že v současné době jsou používány ke zvyšování výkonu jiné techniky než zvyšování taktu (procesor může mít v jednom čase rozpracováno více instrukcí, které jsou řízeny radičem paralelně, používá sdílenou cache mezi jádry atd.).

Dnešním trendem je místo osazování počítače více procesory, dát na počítač jeden procesor s vyšším počtem jader. Z tohoto důvodu jsme se s kolegou rozhodli mírně pozměnit původní zadání a počítat s tím, že máme procesor s více jádry (které mají sdílené cache paměti atp.). Dle našich hypotéz by se výsledek od modelu s N procesory neměl nijak výrazně lišit.

2.4 Paměť

Počítače v současné době používají několik úrovní paměti, podle rychlosti čtení dat, doby přístupu atd. Obecně se dá říci, že rychlý přístup do paměti je velice důležitý pro výkon většiny aplikací. Podle Wikipedie (http://cs.wikipedia.org/wiki/Cache#Cache_v_procesoru, 2012.12.07) je 90% operací prováděných na procesoru čtení z paměti.

2.4.1 Cache

Je velice rychlá paměť, která je umístěna blízko CPU. Má několik úrovní, L1, L2 a L3. Paměť L1 je rychlá jako procesor a minimalizuje tedy zpoždění (je také velice drahá). L2 je mezi procesorem a operační pamětí a L3 byla zavedena na přelomu let 2008/2009 a dostala se na trh spolu s procesory AMD Phenom a Intel Core i7 (http://en.wikipedia.org/wiki/Intel_Core).

K tomu, aby se minimalizovalo čekání CPU, je nutno mít vysokou míru úspěšnosti nalezení požadovaných dat v paměti cache. Pokud se data nenacházejí v L1, postupuje se v hierarchii dále, až se nakonec musí číst data z paměti RAM. Pokud nejsou data ani v operační paměti počítače a musí se načíst z disku/sítě, případně nějakého jiného pomalého zařízení, je potenciální ztráta výkonu obrovská. Proto CPU v takovém případě nečeká na data, ale započne zpracování jiného procesu, u kterého se nemusí tak dlouho čekat na požadovaná data.

2.4.2 RAM paměť

Je označovaná za hlavní paměť počítače. Není sice tak rychlá jako paměti cache na CPU, ale zato je její výroba relativně levná (paměť určená pro servery Kingston 12GB KIT DDR3 1333MHz CL9 ECC, stojí v internetovém obchodě Alza.cz 1689Kč bez DPH, http://www.alza.cz/kingston-12gb-kit-ddr3-1333mhz-cl9-ecc-box-d193272.htm, 2012.12.07) a má vyšší kapacitu.

Tabulka s parametry pamětí je zde (viz. tabulka 1) (Standard JEDEC, zdroj http://cs.wikipedia.org/wiki/DDR3)

Tabulka 1- parametry RAM pamětí

Standardní označení	Takt paměti	Doba cyklu	I/O takt sběrnice	Počet přenesených dat během sekundy	Časování	Označení modulu	Propustnost
DDR3-800	100 MHz	10 ns	400 MHz	800 milionů	CL5-6	PC3-6400	6,4 GB/s
DDR3-1066	133 MHz	7,5 ns	533 MHz	1,066 miliard (=1066 milionu)	CL6-8	PC3-8500	8,533 GB/s
DDR3-1333	166 MHz	6 ns	667 MHz	1,333 miliard	CL7-10	PC3-10600	10,667 GB/s
DDR3-1600	200 MHz	5 ns	800 MHz	1,6 miliard	CL8-11	PC3-12800	12,8 GB/s

2.4.3 Persistentní paměti:

Perzistentní paměti dokáží uchovávat data i po odpojení napájení, typicky mají vysokou kapacitu (stovku až tisíce GB v dnešní době).

Kromě velikosti paměti se zařízení určená k persistentnímu uchování dat liší dobou přístupu a rychlostí čtení/zápisu dat. Především tyto dva parametry jsou stěžejní pro celkový výkon počítače a prací se soubory.

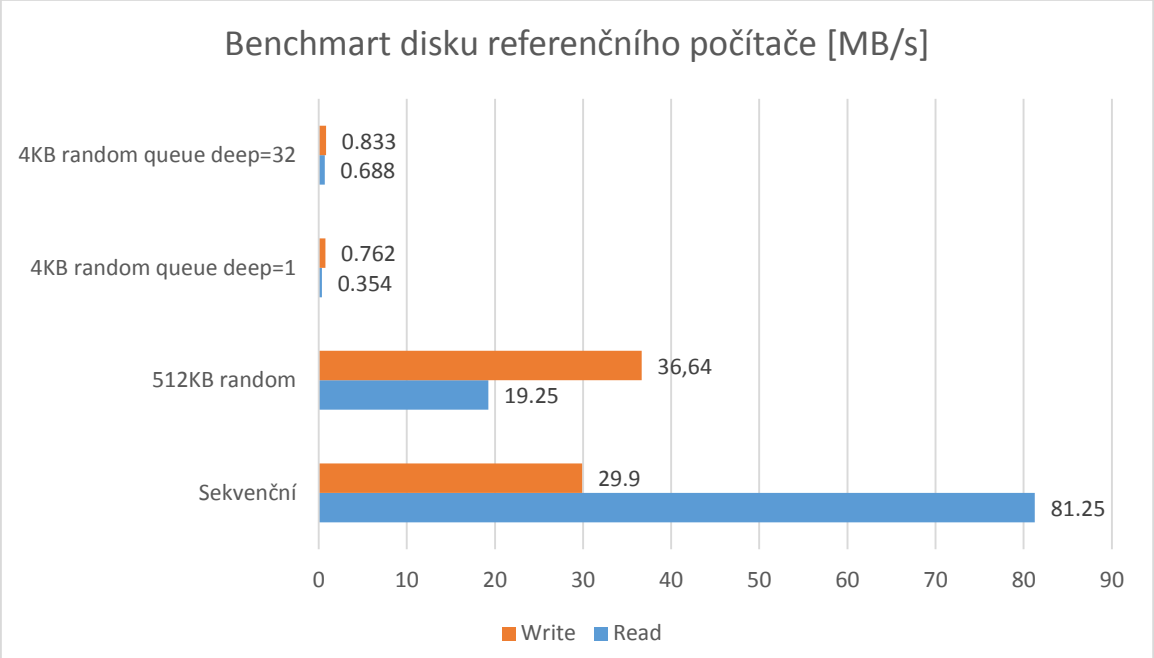
U HDD záleží na době vystavení hlavy, jejím zklidnění, natočení ploten atd. Dnešní pevné disky v sobě také obsahují několik (až desítek) megabajtů vyrovnávací paměti, která pomáhá procesoru/DMA k rychlejší práci, jelikož poté nemusejí čekat na disk který je řádově pomalejší (jak do odezvy, tak do rychlosti práce s daty) než i na dnešní dobu podprůměrně rychlý mikroprocesor v počítači. Pro naše potřeby ovšem stačí vzít následující parametry disku (zdroj, http://en.wikipedia.org/wiki/Hard_disk_drive):

Tabulka 2 - parametry disků

Rotational speed (rpm)	Average latency (ms)
15,000	2
10,000	3
7,200	4.16
5,400	5.55
4,800	6.25

Také je rozdíl mezi náhodným přístupem k paměti (SSD 0.1ms, HDD od 2ms do 12ms u pevných disků v noteboocích) a čtením souboru, který není nikterak fragmentován. V takovém případě doba přístupu k datům na HDD klesá.

Na referenčním stroji (konfigurace viz. kapitola Použité postupy) byly programem CrystalDiskMark naměřeny následující hodnoty (tabulka 3):



Tabulka 3 - benchmark disku referenčního stroje

2.5 Web server

Podle dlouhodobých statistik patří k nejpoužívanějším serverům Apache HTTP Server (kolem 50-60%), Microsoft Internet Information Services (15-20%) a na třetím místě je Nginx (přibližně 8%).

Statistiky byly čerpány ze zdrojů: http://w3techs.com/technologies/overview/web_server/all a http://trends.builtwith.com/Web-Server.

Pro tento projekt byl vybrán jako referenční server Apache HTTP Server, který je open source a poměrně dobře zdokumentovaný (http://httpd.apache.org/docs/).

Tento server je vysoce škálovatelný a má několik režimů práce. Ty se mohou lišit mimo jiné také podle operačního systému. Pro tento projekt byl vybrán model založený na „Preforking Architecture“.

Tato architektura byla první multi-taskovou architekturou u Apache a zůstala jí do verze Apache 2.0, kde se používá jako výchozí pro Unix (kapitola: 4.3.3.2 Preforking Architecture, http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html). Její výhoda oproti podobným předchozím konceptům spočívá v redukci režie při vytváření procesů (Takzvaném forkování. Tento pojem budu používat dále v textu se skloňováním. Bližší informace jsou k nalezení zde: http://en.wikipedia.org/wiki/Fork_(operating_system)), které dokáží obsluhovat požadavky klientů na webový server.

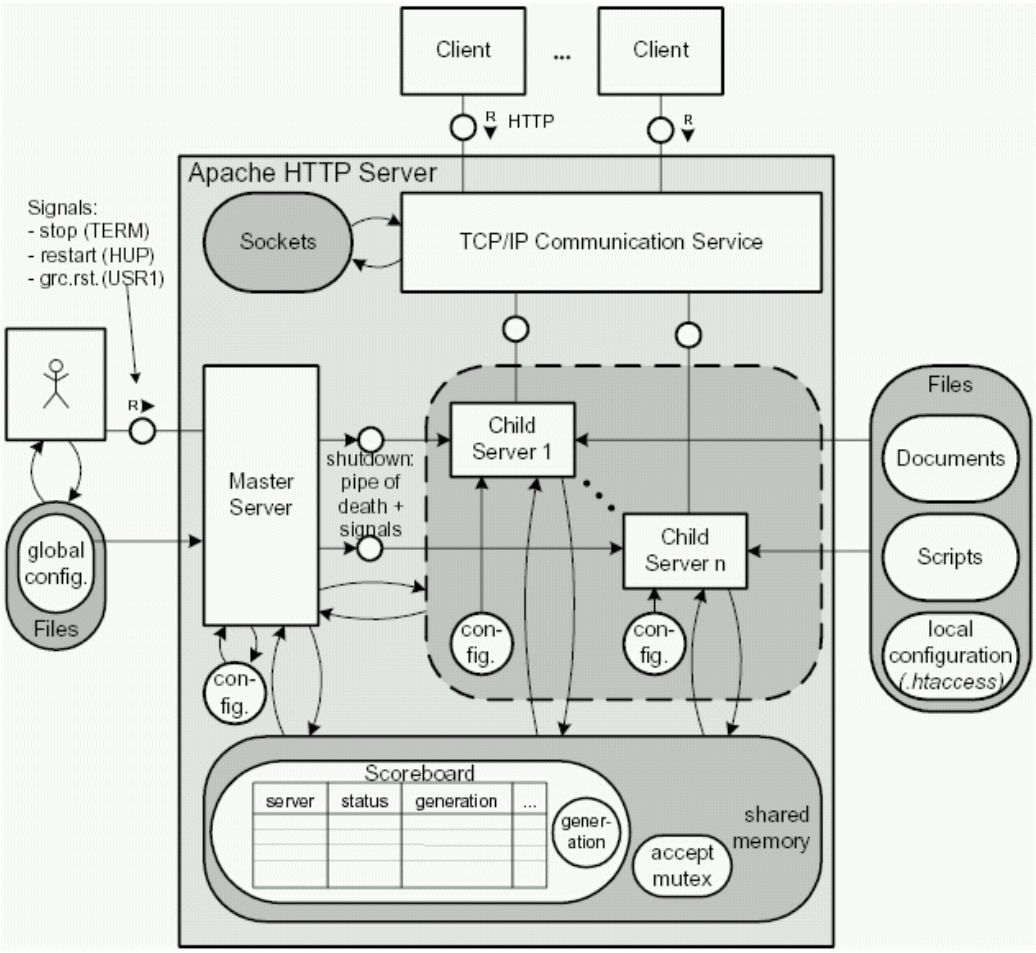
Na serveru běží jeden hlavní proces (Master server), který se stará o příchozí požadavky. Pokud by tento proces musel požadavek sám obsluhovat, tak po dobu obsluhy by nebylo možno obsloužit další uživatele a jejich požadavky. Proto se požadavek předá jinému procesu, který jej obslouží a Master server se může přijímat další požadavky.

Výhoda preforkingu spočívá v tom, že aby při každém přijmutí požadavku nemusel Master server provést forkování (s tím spojená režie vytváření a rušení dalšího procesu), tak si udržuje určitý počet obslužných procesů ve stavu, kdy nezatěžují procesor a pouze zabírají místo v operační paměti. Probuzení tohoto procesu a předání mu obsluhy požadavku je poté rychlejší než forkování.

Poté, co je požadavek obsloužen se proces zařadí zpět mezi čekající procesy, dokud nebude zase aktivován.

Jak již bylo zmíněno výše, tak každý proces zabírá místo v paměti. Pokud je vysoká zátěž serveru a je vytvořená spousta procesů, tak je vše v pořádku. Pokud ovšem zátěž opadne, je zbytečné mít zabranou paměť nevyužitými procesy. Proto je možné v Apache HTTP Server nakonfigurovat jaký má být maximální počet procesů, kolik jich má být minimálně a maximálně ve stavu nečinnosti a podle toho je poté řízen jejich počet za běhu.

Obrázek 2 - fungování Apache HTTP Serveru



Reálná implementace Apache HTTP Server používá „exponential mode“ při vytváření nových procesů. Tento mód se chová tak, že v první smyčce vytvoří jednoho potomka, ve druhé 2, ve třetí 4 a tak dále, dokud se nedosáhne požadovaného počtu potomků. Je to z toho důvodu, že pokud by se mělo během krátké chvíle vytvořit „větší množství procesů“, tak bude operační systém zahlcen.

Při ověřování těchto faktů na reálném stroji jsme také zjistili, že z množiny nečinných procesů (v našem případě vláken (Windows 8 Pro x64)) byl vybírán ten, který jako poslední prováděl obsluhu nějakého požadavku na webový server.

Podrobnější informace o chování Apache HTTP Server jsou k nalezení na stránce: http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html

2.6 Databáze

Jako referenční databází k prozkoumání jsme vybrali PostgreSQL (http://www.postgresql.org/). Tato databáze umožňuje spouštění bloků kódu na serveru a podporuje více jazyků než jen SQL.

Databáze také používá více různých bufferů v paměti RAM pro urychlení práce s daty (dochází tedy k alokaci paměti RAM). Velikosti těchto bufferů jsou nastavitelné (výchozí hodnota je 128MB, http://www.postgresql.org/docs/devel/static/runtime-config-resource.html). Dojde-li místo v těchto bufferech, odmažou se z nich nejdéle nepoužívaná data a ty se nahradí novými, pro které se musí již přistoupit k persistentní paměti počítače (v našem případě k pevnému disku). Dojde tedy ke zpomalení a zvětší se tak odezva na požadavek (měření na referenčním počítači ukázala, že pokud se nemusí jít pro data na disk, tak provedení jednoduchých dotazů nad malou databází (do 1 MB dat) je v rádu několika deseti tisícin sekundy).

U databáze silně záleží na jaké stroji a s jakou konfigurací je spuštěna. Čistě databázové servery si mohou zabrat většinu paměti, aby nemuselo pro data na pomalý disk. Databázové servery kombinované s webovými jsou poté často konfigurovatelné tak, aby dokázaly obsluhovat i několik set požadavků najednou.

3 Koncepce - modelářská témata

3.1 Model web serveru

Model web serveru vycházel z informací uvedených v kapitole Web server. Byla použita preforking architektura a ne přímo varianta ze zadání (viz. *„Kromě uživatelských procesů v systému pracuje i webový server a databázový server modelované jako zařízení s kapacitou jedna zpracovávaných požadavků“*).

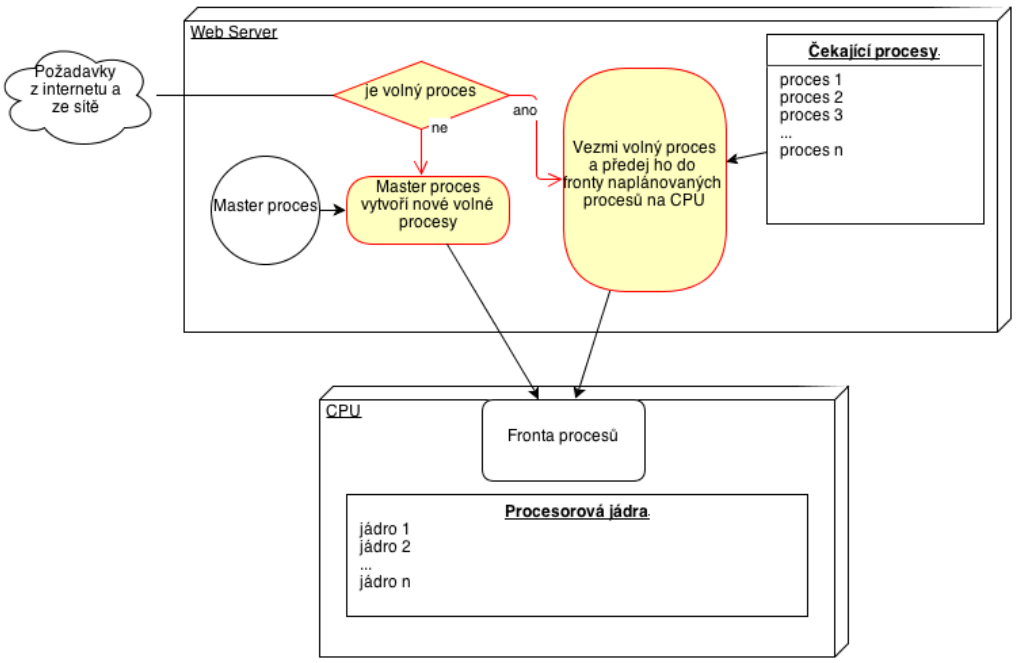
3.1.1 Použití preforking metody tvorby procesů

Důvodem je, že takovéto servery se v praxi nepoužívají a došlo by ke zkreslení výsledků a odchýlení se od skutečnosti.

Verze modelu pro tento projekt počítá s tím, že si Master server předpřipravuje procesy místo vláken. Také do modelu nebyl zahrnut „exponential mode“, který slouží k postupnému navyšování počtů procesů.

Web server část modelu tedy funguje tak, že na počátku je vytvořen Master Server a ten stejně jako Apache HTTP Server vytvoří určitý počet čekajících procesů, které budou obsluhovat požadavky přicházející na web server.

Pokud se požadavek nepodaří obsloužit okamžitě (nejsou k dispozici volné procesy), tak je zařazen do fronty a tam čeká tak dlouho, dokud na něj nepřijde řada.



Obrázek 3 - model web serveru a komunikace s CPU

Po obsloužení požadavku se vlákno zařadí zpět do množiny čekajících vláken. Pokud je ovšem překročen limit pro maximální počet čekacích procesů, je některý z procesů ukončen.

Dle našeho pozorování se Apache HTTP Server snaží využívat stále dokola těch procesů, které naposledy obsluhovaly požadavky. Podle našeho názoru je to z důvodu, že u takového procesu je největší pravděpodobnost, že je část jeho dat v některé rychle přístupné cache paměti. Tento fakt ovšem nebyl do modelu zahrnut.

3.1.2 Čas potřebný pro obsluhu požadavku:

Při analýze Apache HTTP Server bylo také zjištěno, že i když na server nechodí žádné požadavky, tak Master Server proces konzumuje průměrně 17 000 cyklů procesoru za sekundu. Měření byla prováděna nástroji Process Explorer (verze 15.23, http://technet.microsoft.com/cs-cz/sysinternals/bb896653.aspx) a Average CPU Cycles (verze 2.3.1, http://www.softpedia.com/get/System/System-Info/Average-CPU-Cycles.shtml) pod systémem Microsoft Windows 8 Pro x64. Výsledky měření těchto nástrojů spolu úzce korelovaly (lišily se v řádu jednotek procent), tedy tato měření považuji za korektní.

To, kolik časového kvanta bude potřebovat jeden proces web serveru ke splnění požadavku od klienta je možno nastavit v konfiguračním souboru k simulátoru. Jednotkou je počet taktů procesoru na jeden bajt zpracovaných dat. Z měření prováděných na referenčním stroji nám vyšlo číslo 120 taktů na 1 B. Tento výsledek se, ovšem může měnit u různých webových serverů a dalších technologií na to navázaných (PHP, ASP/.NET atp.).

Celkový počet taktů potřebných pro obsluhu požadavku se poté vypočítá následovně:
$početCyklůCPU = početTaktůNaBajt * Normal(0, velikostDatPožadovanýchKlientem)$.

Algoritmus dále zajistí, aby generovaný velikost stránky nebyla záporná a větší či menší než 1/10 velikosti zadané v konfiguračním souboru.

3.2 Databáze

Databáze byla modelována jako obslužná linka s kapacitou jedna zpracovávaných požadavků. Zde jsme se drželi oficiálního zadání projektu, kde bylo napsáno *„Kromě uživatelských procesů v systému pracuje i webový server a databázový server modelované jako zařízení s kapacitou jedna zpracovávaných požadavků“*.

Důvod je ten, že databáze by stejně musela čekat buď na data z disku, nebo z bufferů (velikost bufferu je u modelu parametrizovaná) alokovaných v paměti RAM.

Mimo jiné každé provádění dotazu na databázi spotřebovává výpočetní čas některého z jader procesoru. To, kolik času daný úkon potřebuje k provedení je dáno parametrem v konfiguračním souboru a generátorem normálního rozložení (IMS, https://www.fit.vutbr.cz/study/courses/index.php?id=8662, slide č. 96) pravděpodobnosti. Lze tak jednoduše provádět experimenty s různě složitými dotazy na databázi.

3.3 Model procesoru

Model procesoru je modelován pomoci třídy facility (IMS, https://www.fit.vutbr.cz/study/courses/index.php?id=8662, slide č. 149). Fronta procesů připravených běžet je modelována pomocí fronty Q1 příslušného zařízení.

Předpokládáme, že se v jednom taktu jádra procesoru vykoná pouze jedna instrukce. Paralelní běh více instrukcí najednou v jednom jádře zde není zohledněn.

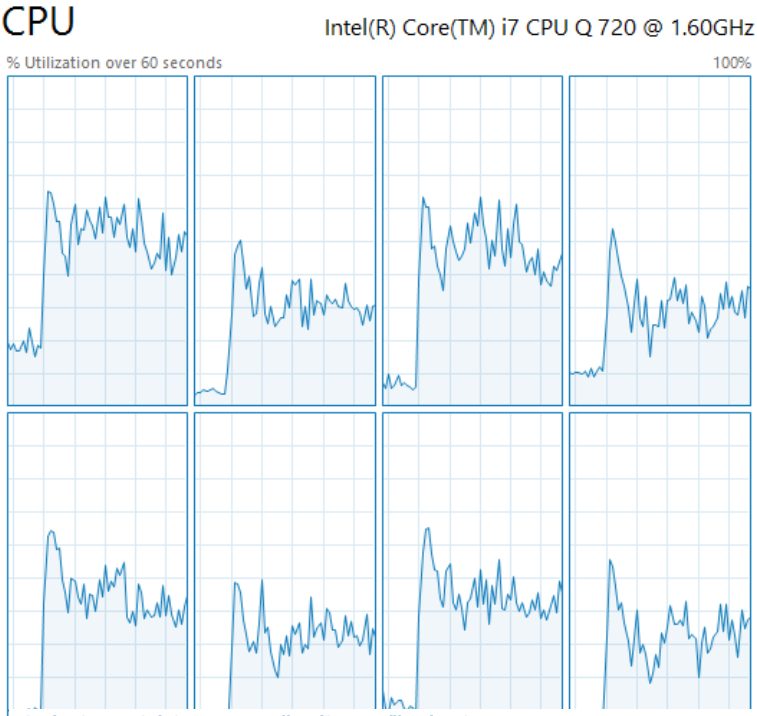
3.3.1 Round robin

Jako plánovač byl pro procesor vybrán Round-robin scheduler (http://en.wikipedia.org/wiki/Round-robin_scheduling). Ten funguje na principu přidělování časového kvanta procesům běžícím na CPU. Jakmile je dané časové kvantum vyčerpáno, je proces nahrazen jiným a zařazen zpět do fronty na čekání. U tohoto plánování se předpokládá, že všechny procesy běžící na procesoru mají konstantní prioritu.

Tento způsob plánování se sice v dnešní době moc nepoužívá, ale při rozhodování, jaký plánovač namodelovat jsme vycházeli z předpokladu, že pokud výsledky simulací budou v dostatečné míře korelovat s výsledky reálného počítače, tak není třeba používat nic složitějšího.

3.3.2 Řazení procesů do front jader CPU

Při testování procesoru referenčního stroje (Intel Core i7 720QM) se ukázalo, že zátěž se rozkládá přibližně rovnoměrně na jednotlivá jádra procesoru. Na základě tohoto faktu jsme naprogramovali algoritmus, který pracuje s příchozími procesy podobným způsobem.



Obrázek 4 - ukázka rovnoměrného zatížení jader CPU

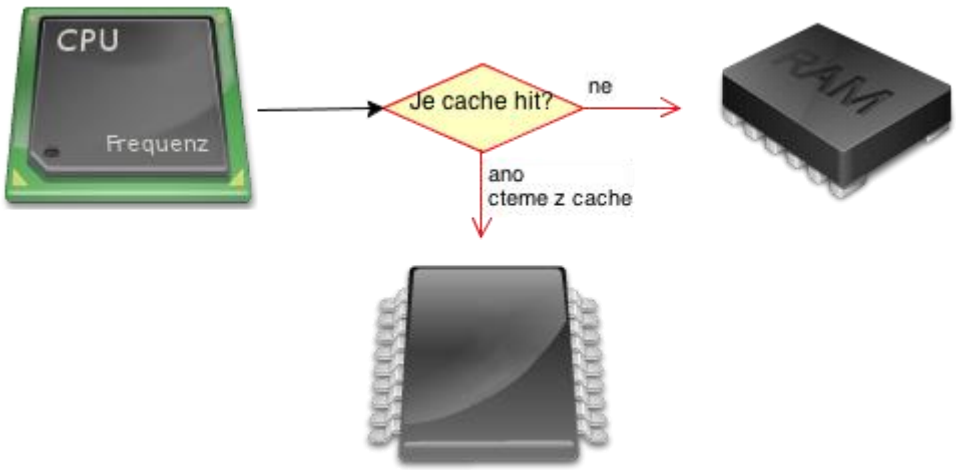
Má-li být proces zařazen do fronty, čekajících na obsloužení (na nějakém z N jader), je pro jeho zařazení použit následující algoritmus:

```
1. Set<FrontaJadra> fronty_jader;
2. Set<FrontaJadra> nejkratsi_fronty;
3. FrontaJadra nejkratsi_fronta;
4. ...
5. nejkratsi_fronty = fronty_jader.getNejkratsiFronty(); // vrati
   mnozinu front, ktere maji nejkratsi fronty cekajicich procesu
   na svych jadrech
6. nejkratsi_fronty.shuffle(); // provedeme nahodne promichani
   mnoziny
7. nejkratsi_fronta = nejkratsi_fronty.get(0); // ziskame prvni
   frontu z mnoziny
```

3.3.3 Cache procesoru

Kvůli dnešní relativně vysoké rychlosti obvodů procesoru je potřeba mít co možná nejrychlejší přístup k datům do paměti. Proto má CPU vlastní cache paměti několika úrovní. Do našeho modelu jsme

ovšem zahrnuli jen jednu obecnou úroveň, která má nahrazovat všechny ostatní a obecně slouží k redukci přístupu k datům.



Obrázek 5 - modelování přístupu do cache a do RAM paměti

Pravděpodobnost nalezení dat v cache paměti a urychlení oproti přístupu do paměti RAM je nastavitelná v konfiguračním souboru k simulátoru (pro nastavení přibližný hodnot jsme čerpali z materiálů k předmětu IOS http://www.fit.vutbr.cz/study/courses/IOS/public/ a materiálu na stránce http://people.engr.ncsu.edu/efg/506/sum99/001/lec3-cache.pdf).

Algoritmus pro výpočet urychlení přístupu do paměti je následující:

```
1. // vzorec vypoctu doby obsluhy RAM
2. process.working_time = (size / ((data_rate * 8.0) * 2.0));

3. // velmi pravdepodobny pristup pres cache procesoru
4. Number cache_hit = Uniform(0.0, 1.0); // generovani uniforniho
   rozlozeni pro
5. if (cache_hit < global_parametr.hit_rate) {//pokud je cache hit
6.     process.working_time *= global_parametr.hit_speed_up;
7.     // zkratime predtim vypocitanou dobu
8. }
```

3.4 Model persistentní paměti

Do modelu jsme zahrnuli pouze možnost sekvenčního čtení. Náhodné čtení malých bloků dat z disku, případně fragmentace větších souborů zde modelovány nebyly. Pokud chceme simulovat, že se čtou z disku fragmentovaná data (případně více malých souborů náhodně rozložených na disku), tak se musí změnit přímo parametry disku v konfiguračním souboru.

Model persistentní paměti je parametrizován následovně:

Parametr	Běžné hodnoty
Velikost disku	Stovky až tisíce GB
Rychlost čtení	Desítky až stovky MB/s

Rychlost zápisu	Desítky až stovky MS/s
Přístupová doba	12ms až 0.1ms (HDD až SSD disky)

Tabulka 4- parametry modelu persistentní paměti

Vzorec pro výpočet času čtení z disku je:

$$rychlost = \frac{velikostDat}{rychlostČtení} + přístupováDobaDisku$$

3.5 Model paměti RAM

Základ pro výpočet rychlosti paměti RAM je parametr data rate z konfiguračního souboru pro simulaci.

Rovnice pro výpočet doby přenosu dat je dovozena z rovnice maximální teoretické přenosové rychlosti (http://www.hardwaresecrets.com/article/Understanding-RAM-Timings/26):

$$časPřenosuDat = \frac{velikostData}{dataRate * 8 * 2}$$

Paměť RAM je také využívaná databázovým serverem jako cache dat načtených z disku. Předpokládáme, že serverový operační systém má defaultně cachování disku vypnuto a starají se o něj samy aplikace, které tento způsob urychlení práce s daty používají.

3.6 Procesy operačního systému a uživatelů

Procesy operačního systému, respektive různé služby běžící na pozadí jsou modelovány stejným způsobem. V systému představují neustálou zátěž, která je přítomna i v době, kdy nepřichází žádné požadavky na web server nebo databázový server.

Všechny procesy v modelu jsou odvozeny od třídy Process z knihovny Simlib (IMS, https://www.fit.vutbr.cz/study/courses/index.php?id=8662, slide č. 164).

3.7 Jednotky užité v modelech

V modelech se používají základní jednotky běžně užívané v informatice a fyzice (jednotky SI, http://en.wikipedia.org/wiki/International_System_of_Units).

3.7.1 Čas

Jako časová jednotka je použita sekunda [s]. Sekunda je použita pro stanovení délky doby simulace, dobu odezvy různých částí systému a také se od ní odvozují další jednotky, jako je frekvence procesoru a rychlost přenosu dat.

3.7.2 Data

Jednotkou pro data je jeden byte [B], tedy 8 bitů [b]. Tato jednotka se používá v modelu pro stanovení velikostí pamětí RAM a HDD/SSD.

3.7.3 Frekvence

Frekvence je použita jako parametr u procesoru a paměti RAM. Z této frekvence se vypočítává časové kvantum pro doby běhu procesů a dobu přístupu do RAM.

3.7.4 Rychlost přenosu dat

Je tvořen základními jednotkami. Tedy jedná se o počet přenesených bajtů za sekundu [B/s]. Používá se při čtení dat z cache procesoru, RAM a persistentní paměti.

4 Architektura simulačního modelu/simulátoru

Jak již bylo zmíněno v kapitole 2.2 Požité postupy, simulační model byl psát v jazyce C++ za pomoci simulační knihovny Simlib. Při psaní zdrojového kódu se také ve velké míře uplatnily prvky objektově orientovaného programování a bylo užito i C++ STL (Standard Template Library) především kvůli podpoře různých typů kolekcí.

4.1 Mapování konceptuálního modelu do simulačního

Třídy a jejich instance ve zdrojovém kódu odpovídají objektům reálného světa popsaným konceptuálním modelem následovně:

Konceptuální model	Třída
CPU	Cpu
Proces na CPU	CpuProcess
Databázový server	DbServer
Persistentní paměť	HardDisk
RAM paměť	Ram
Webový server	WebServer

Tabulka 5 - mapování konceptuálního modelu na simulační

4.2 Struktura projektu

4.2.1 Zdrojové kódy

Zdrojové kódy simulátoru jsou uloženy ve složce src. Ty, které končí příponou *.cc obsahují C++ kód a soubory s příponou *.h jsou příslušné hlavičkové soubory.

4.2.2 Makefile

V kořenové složce je k nalezení makefile, pomocí nějž se celý projekt překládá.

Příkaz	Popis příkazu
$ make	Přeloží projekt
$ make run	Pustí přeložený projekt a zajistí vygenerování grafů
$ make plot	Zajistí pouze vygenerování grafů (za předpokladu, že jsou dostupná data)
$ make tar	Vytvoří archiv projektu k odevzdání
$ make clean	Smaže veškerý vygenerovaný výstup a přeložený kód (kromě DataSampler.jar)
$ make auto	Provede překlad, spuštění a vytvoření grafů dohromady.

Tabulka 6 - příkazy pro make

4.2.3 Další programy

V rámci celého projektu byl vytvořen program DataSampler pro platformu Java, který se stará o samplování a zprůměrování dat ze simulace.

Jako argumenty programu se předává `[cesta_k_datovému_souboru, (int)velikost samplování]`. Sloupce by měly být odděleny jedním tabulátorem a program si sám detekuje jejich počet. Řádkové komentáře musí začínat znakem `#`. Výsledek konverze je poté vypsán na standardní výstup, případně jej můžete přesměrovat do výstupního souboru.

Příklad spuštění: `$ java -jar DataSampler.jar out/input.data 150 > out/sampled.data`

POZNÁMKA: Pro kompilaci a použití DataSampleru je potřeba mít nainstalované JDK 1.6 a vyšší.

4.3 Základní třídy v projektu

- **ArgsParser** – Stará se o parsování konfiguračního souboru. Parametry, které si z tohoto souboru přečte, uloží do svých hierarchicky poskládaných proměnných. Například k velikosti procesu web serveru se přistupuje následovně: `glob_sim_args->web_server->process_size`. Globální proměnná s tímto objektem je k nalezení v souboru global.h jakožto `extern ArgsParser *glob_sim_args;`.
- **Cpu** – Modeluje jádra procesoru a stará se o vkládání procesů do jednotlivých jader. Propojuje všechny ostatní komponenty v počítači.
- **CpuProcess** – Je odvozena od třídy Process z knihovny Simlib reprezentuje skutečné procesy na serveru.
- **DbServer** – Slouží pro simulaci databázového serveru. Běží od začátku simulace až do jejího konce.
- **HardDisk** – Slouží pro simulaci přístupu k persistentní paměti serveru.
- **Logger** – Slouží k vytváření datových souborů pro program gnuplot.
- **Ram** – Slouží pro simulaci přístupu k hlavní paměti počítače.
- **WebServer** – Instance této třídy se stará o chod web serveru, alokaci paměti, vkládání obslužných procesů do CPU, konsolidaci přebytku či nedostatku volných serverů atd. Běží od počátku simulace až do jejího konce.

4.4 Vykreslení grafů

Grafy se kreslí z dat vygenerovaných při simulaci a to programem gnuplot. V určitých případech (je-li vhodné data zprůměrovat a zvýraznit hlavní linii průběhu) se používá Java program DataSampler, který je součástí projetu i se svými zdrojovými kódy.

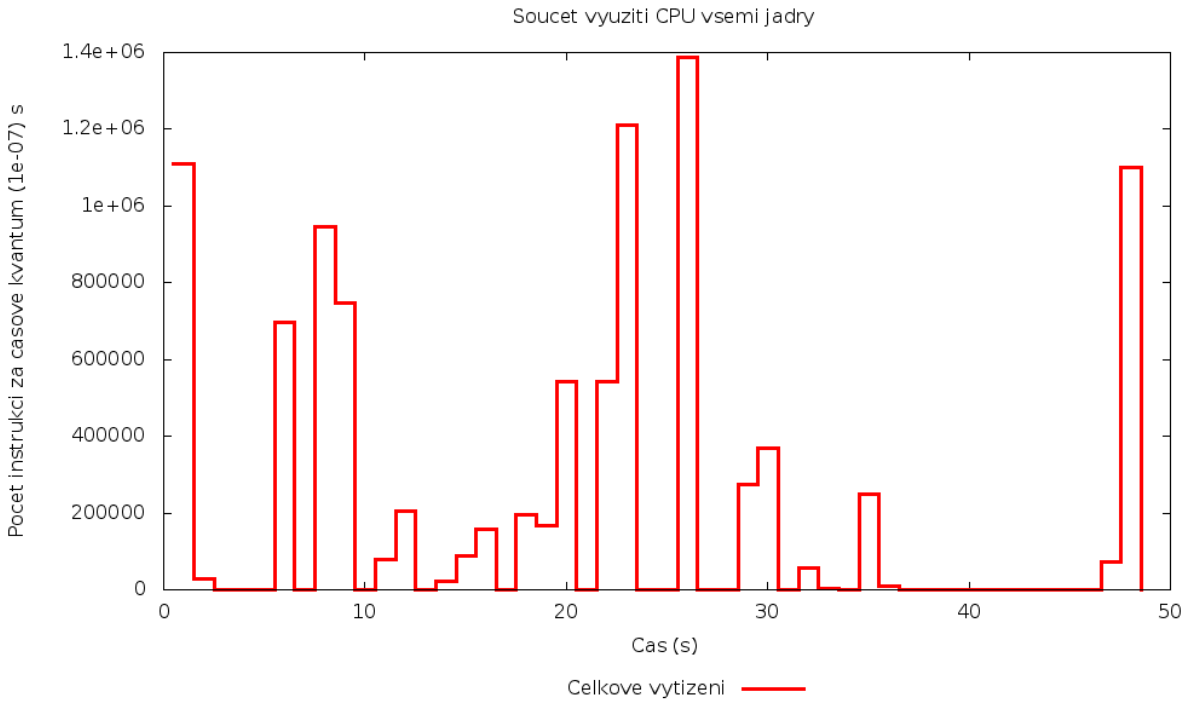
Výsledné data a grafy jsou k nalezení ve složce out.

5 Podstata simulačních experimentů a jejich průběh

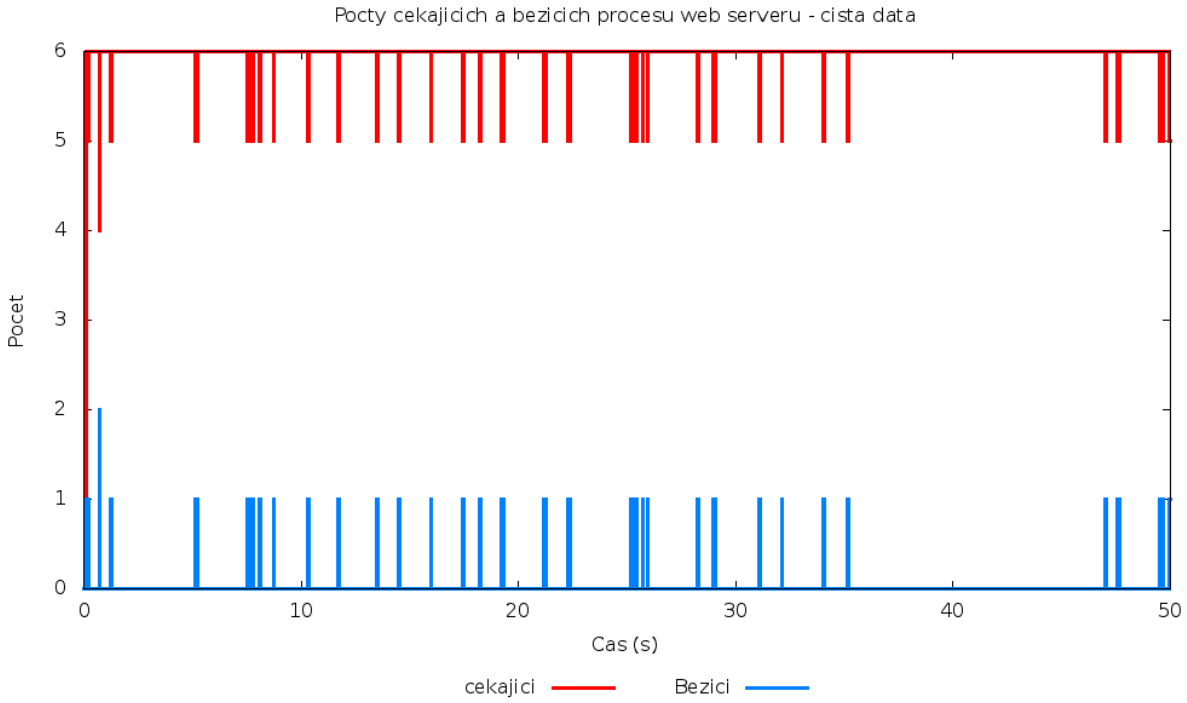
Simulační experimenty měly za úkol ukázat chování serveru při nízké, střední a vysoké zátěži a potvrdit/vyvrátit hypotézu, že slabým místem většiny dnešních počítačů jsou pevné disky.

5.1 Nízká zátěž

Podíváme-li se na experiment s nízkou zátěží, tak uvidíme, že procesor je podle očekávání poměrně málo vytížen. Stejně tak web serveru stačí většinou jen 1 proces na zpracování požadavku a zbytek má předforkovaný.

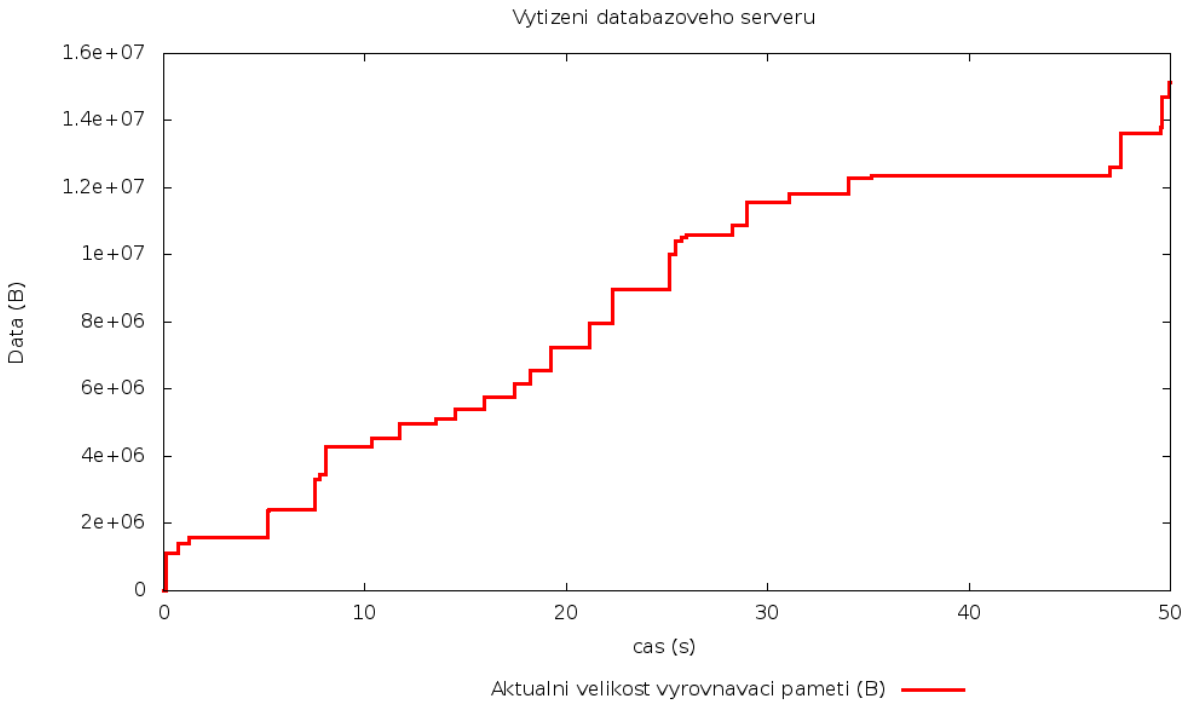


Obrázek 7 - průměrné vytížení procesoru je při menší zátěži také menší

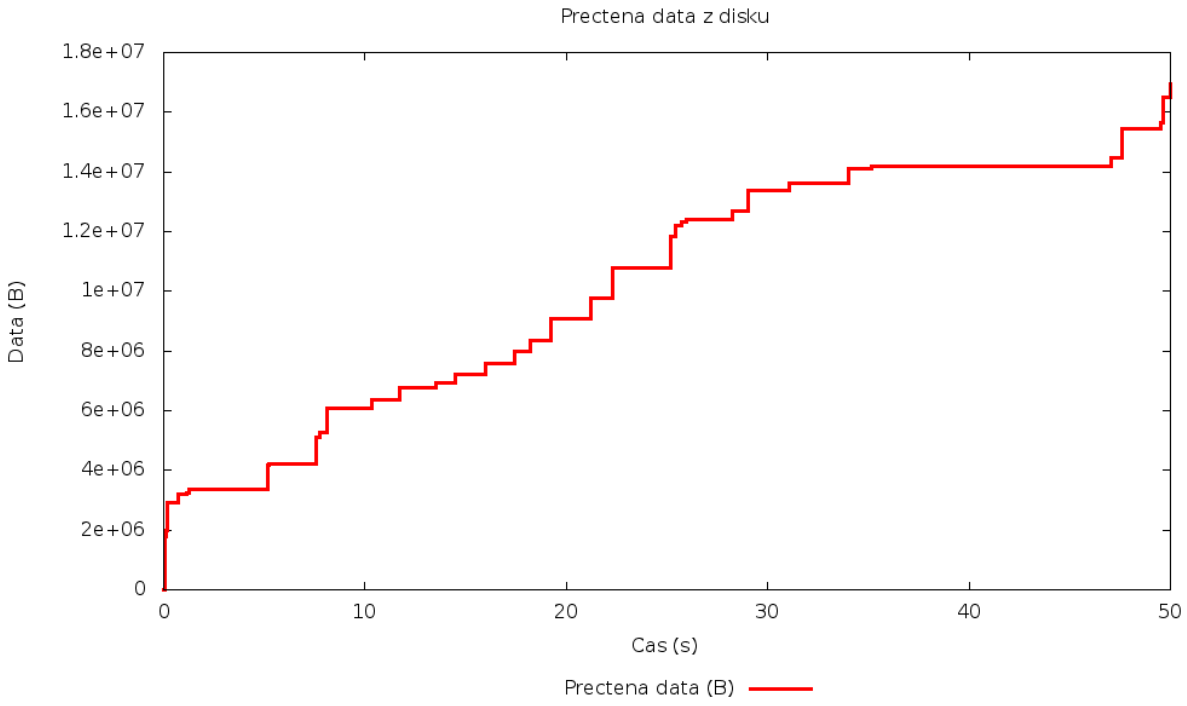


Obrázek 6 - při malé zátěži si web server vytvoří minimální počet volných procesů, který je dán v konfiguračním souboru a poté z těchto volných procesů využívá 1, maximálně 2

Dále můžeme pozorovat, že počet dat uložených do bufferu databáze koreluje s množstvím dat přečtených z disku (je malá zátěž a velikost bufferu v RAM databázi stačí).



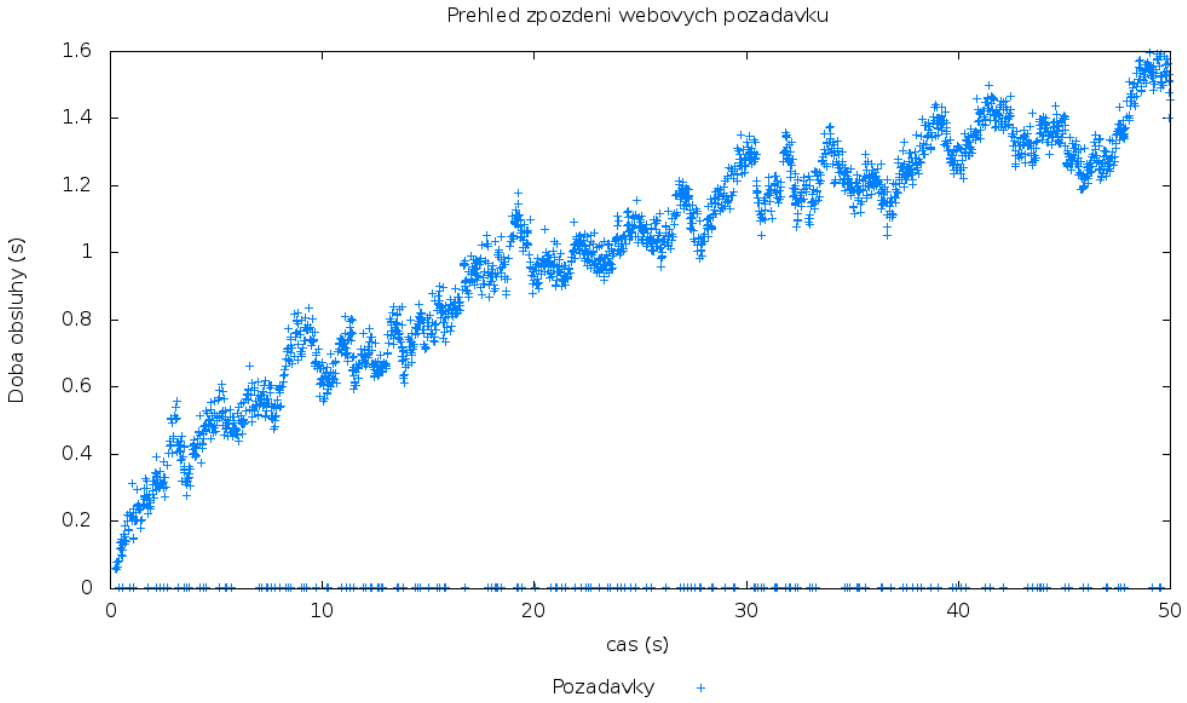
Obrázek 8 – data v bufferu databáze narůstají postupně a zároveň fakt, že graf nedosáhl svého maxima a tam se nezastavil ukazuje, že je bufferu stále volné místo.



Obrázek 9 - celkový počet přečtených dat z disku má v čase schodovitý tvar a ukazuje, že disk není ani zdaleka plně vytížen

5.2 Vysoká zátěž

Smyslem tohoto experimentu bylo získat výsledek experimentu, kdy je server pod velikou zátěží a web server musí zahazovat požadavky od klientů, jelikož má omezený buffer na maximální velikost 200 požadavků.

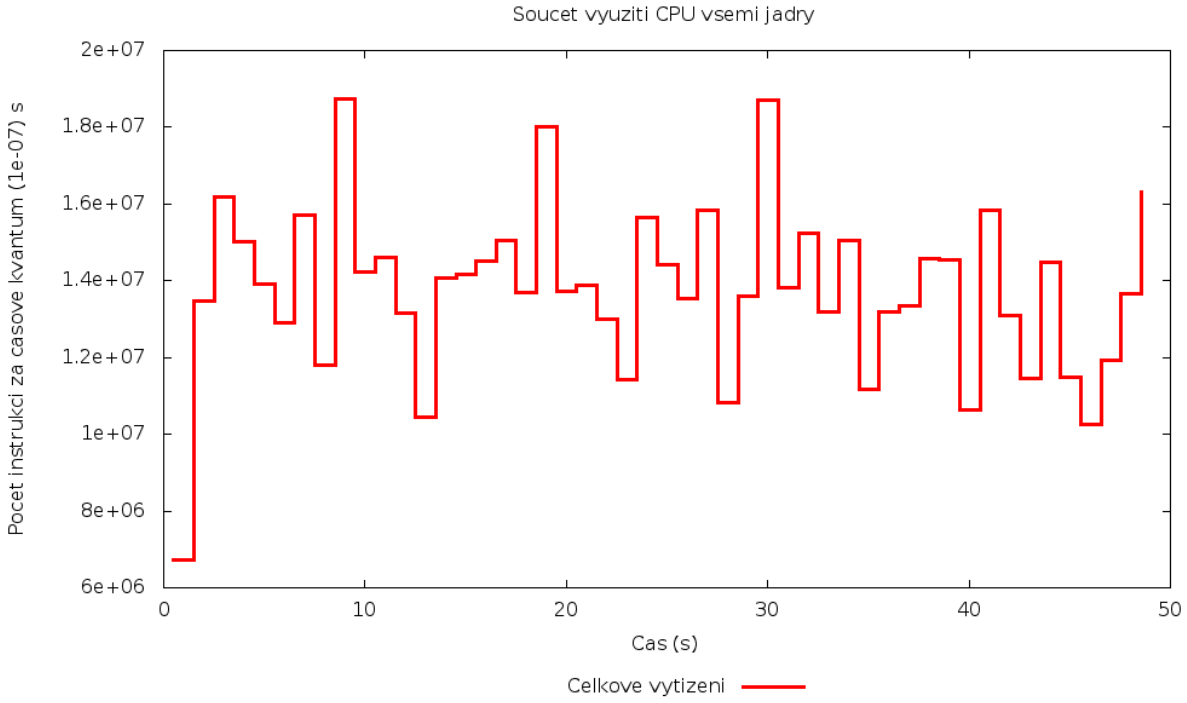


Obrázek 10 - velké zatížení serveru, doba obsluhy požadavků stoupá a část jich je zahazována

Křížky na spodu grafu (doba obsluhu == 0)ukazují, že požadavky na web server se nevlezly do bufferu a musely být zahazovány, přičemž s dobou simulace doba obsluhy požadavku rostla (v delším časovém horizontu by web server vytvořil tolik procesů, kolik má dovoleno jako maximum a požadavky, které by se nestíhaly, obsluhovat by zahazoval).

Obrázek 11 - graf znázorňuje počet procesů web serveru, které jsou postupně forkovány a rostou v čase

Na grafu znázorňujícím počet procesů web serveru je patrné, že se Master proces snaží vytvořit co nejvíce svých potomků, přičemž nezůstávají žádné čekající procesy, které by nic nedělaly.

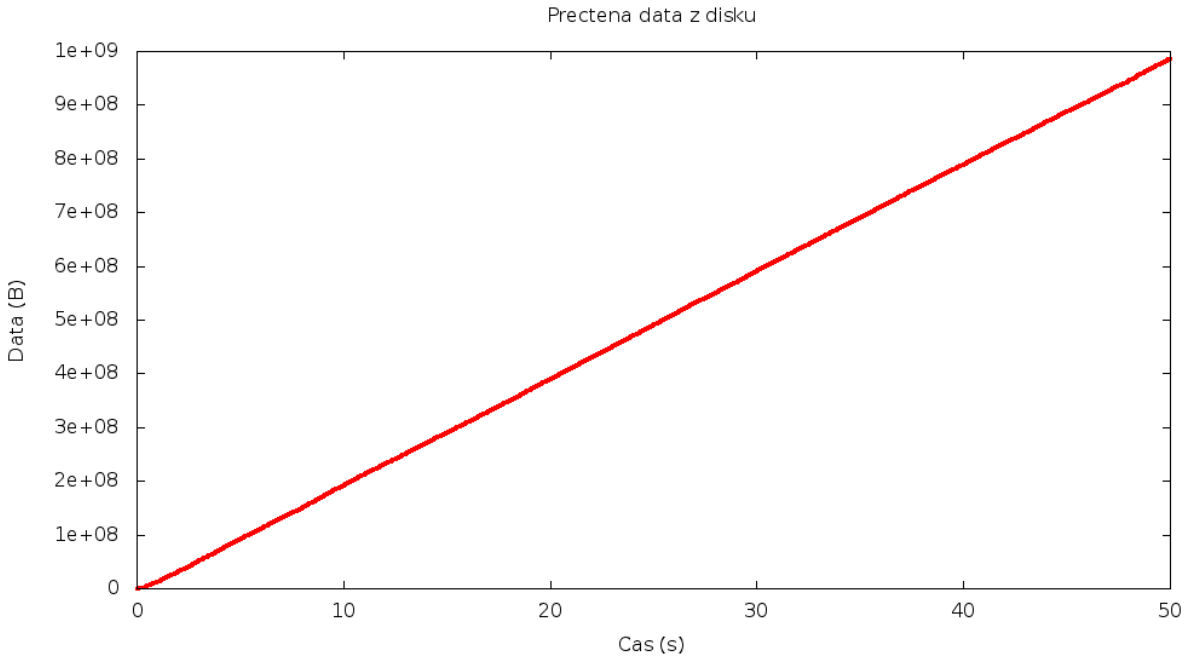


Obrázek 12 - celkové vytížení procesoru

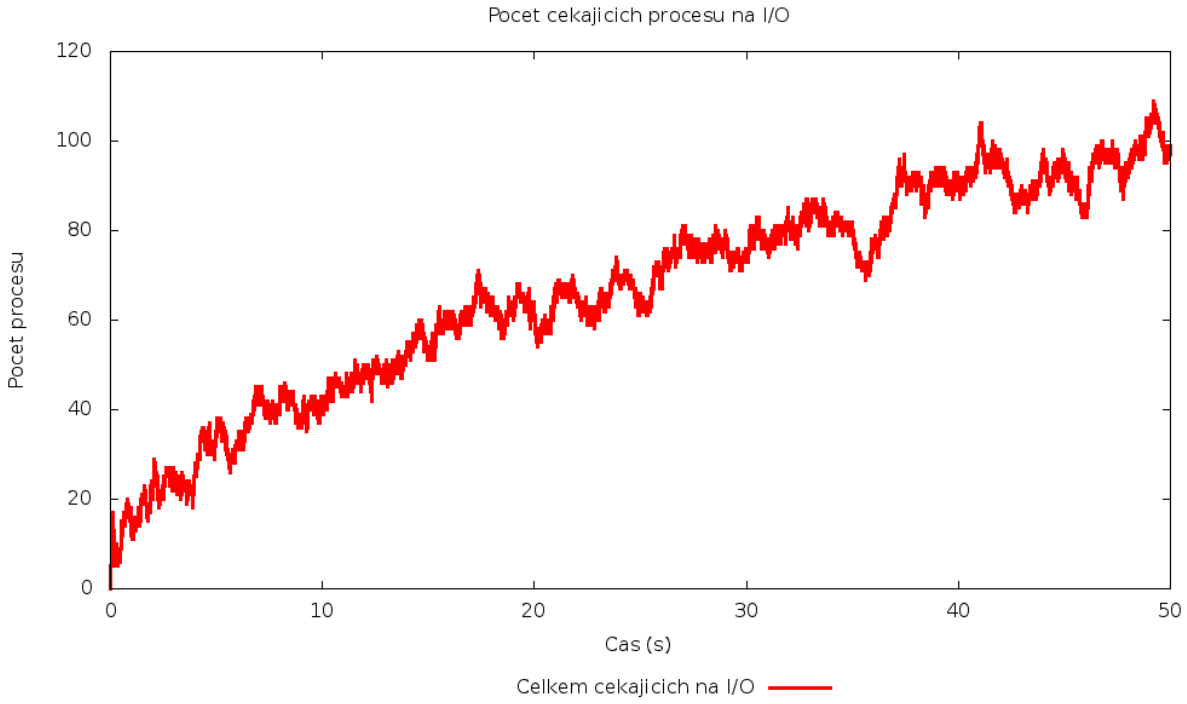
Podíváme-li se na průměrné zatížení všech osmi jader procesoru, uvidíme, že zdaleka nejsme na maximu. Čím je tedy způsobeno, že doba obsluhy požadavků stále roste, když procesor stíhá?

Odpovědí na tuto otázku je graf, který znázorňuje čtení z pevného disku a počet čekající procesů na IO operaci. Je zde možné spatřit, že objem celkově přečtených dat v čase lineárně roste, což značí, že

disk je vytížen naplno a akorát brzdí databázi, která zase brzdí procesy web serveru a díky tomu se zvyšuje odezva na webové požadavky.



Obrázek 13 - celkový počet dat přečtených z disku roste lineárně v čase, což značí plné vytížení disku

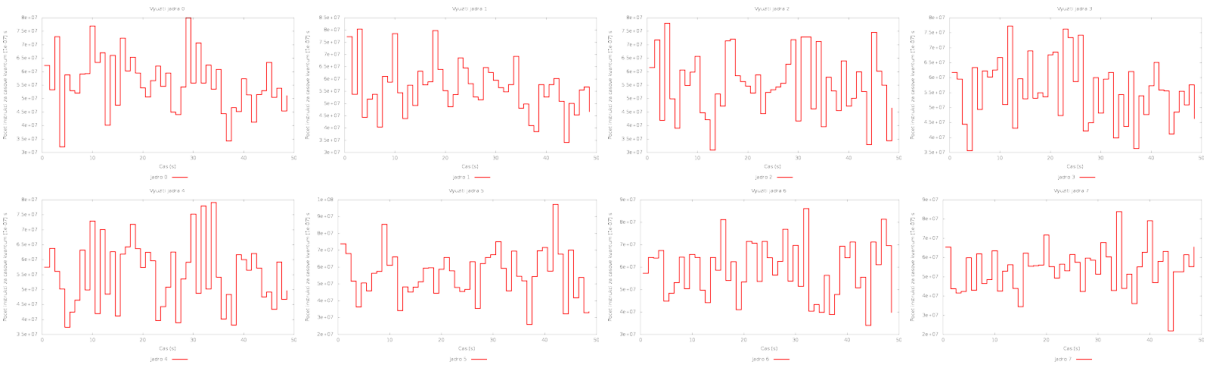


Obrázek 14 - počet čekající operací na disk postupně roste, jak HDD nestíhá

5.3 Rovnoměrné rozdělení zátěže mezi jádry procesoru

Zajímavé je také sledovat, jak si procesor poradí s rozložením zátěže na jednotlivá jádra. Jak je na grafu vidět, tak vypadá velice podobně jako jakýkoli jiný reálný graf se 4 jádrovým procesorem, který

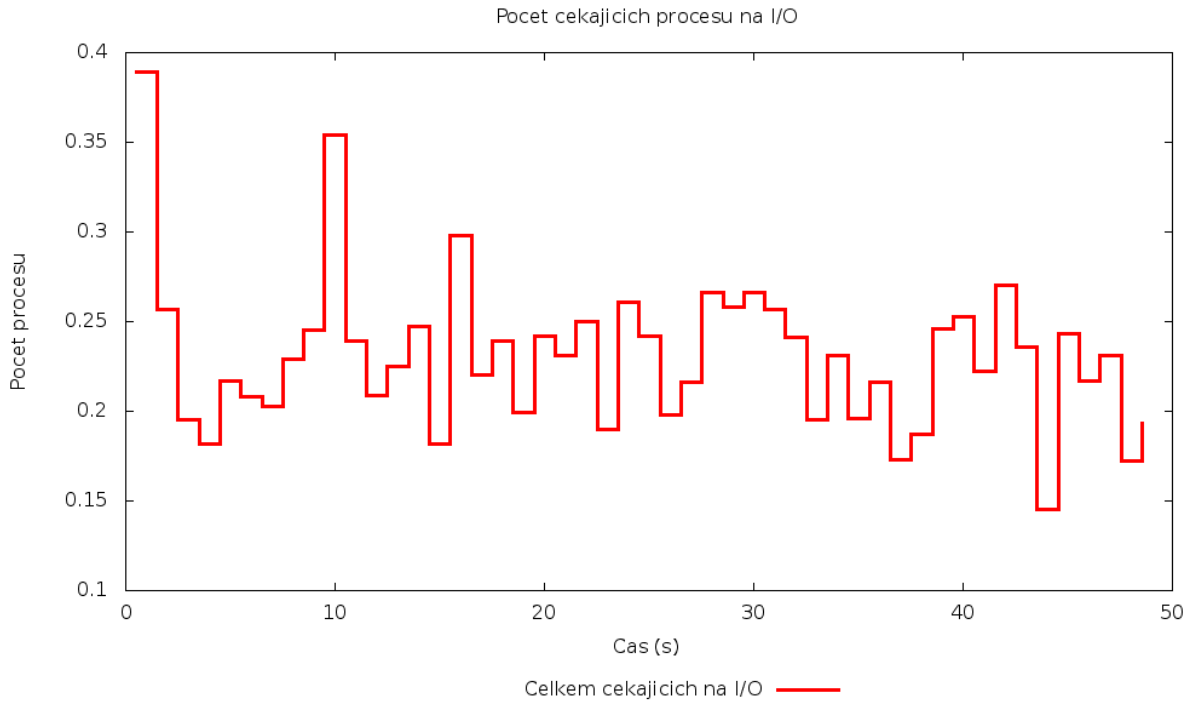
je schopen zpracovávat 8 vláken najednou (viz. kapitola 3.3.2 Řazení procesů do front jader CPU, obrázek 4).



Obrázek 15 - ukázka rovnoměrného rozložení zátěže mezi jádry procesoru

5.4 Server s SSD diskem a vysokou zátěží

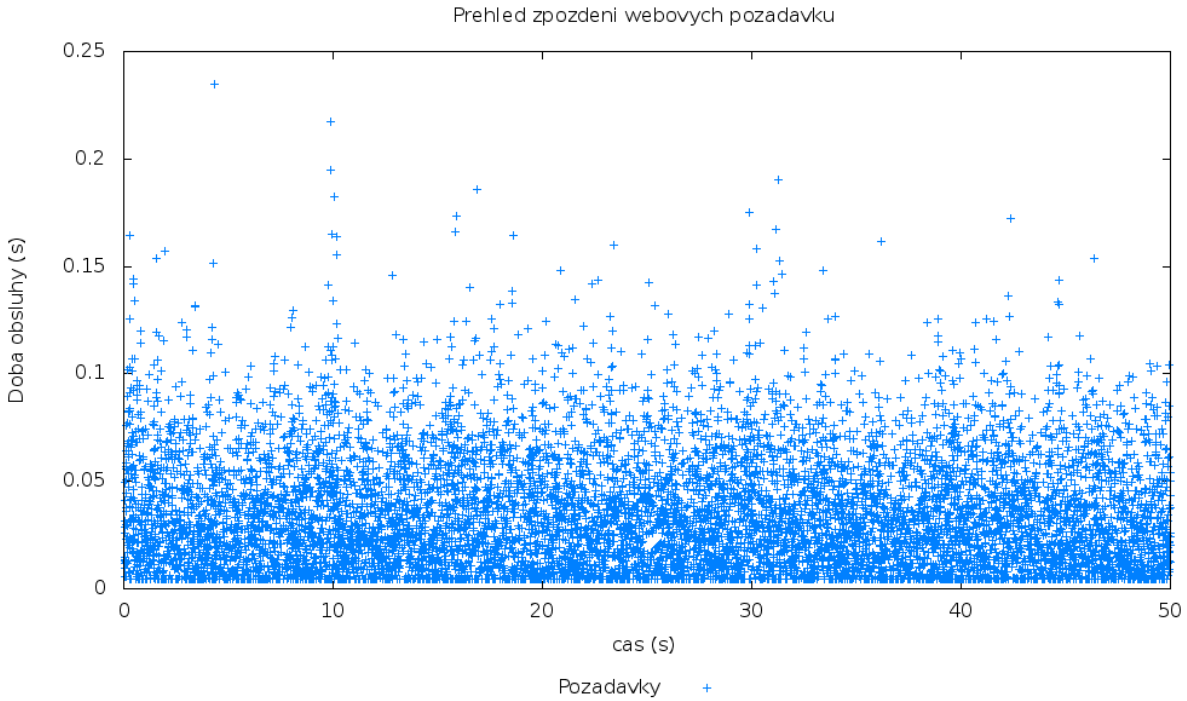
K tomu, abychom mohli potvrdit naši hypotézu, že odezvu serveru nejvíce brzdí pomalá persistentní paměť, nakonfigurovali jsme tuto paměť parametry typickými pro SSD (odezva 0.1ms a rychlost čtení 500MB/s, http://en.wikipedia.org/wiki/Solid-state_drive). Ostatní parametry zůstaly stejné jako u experimentu s vysokou zátěží serveru.



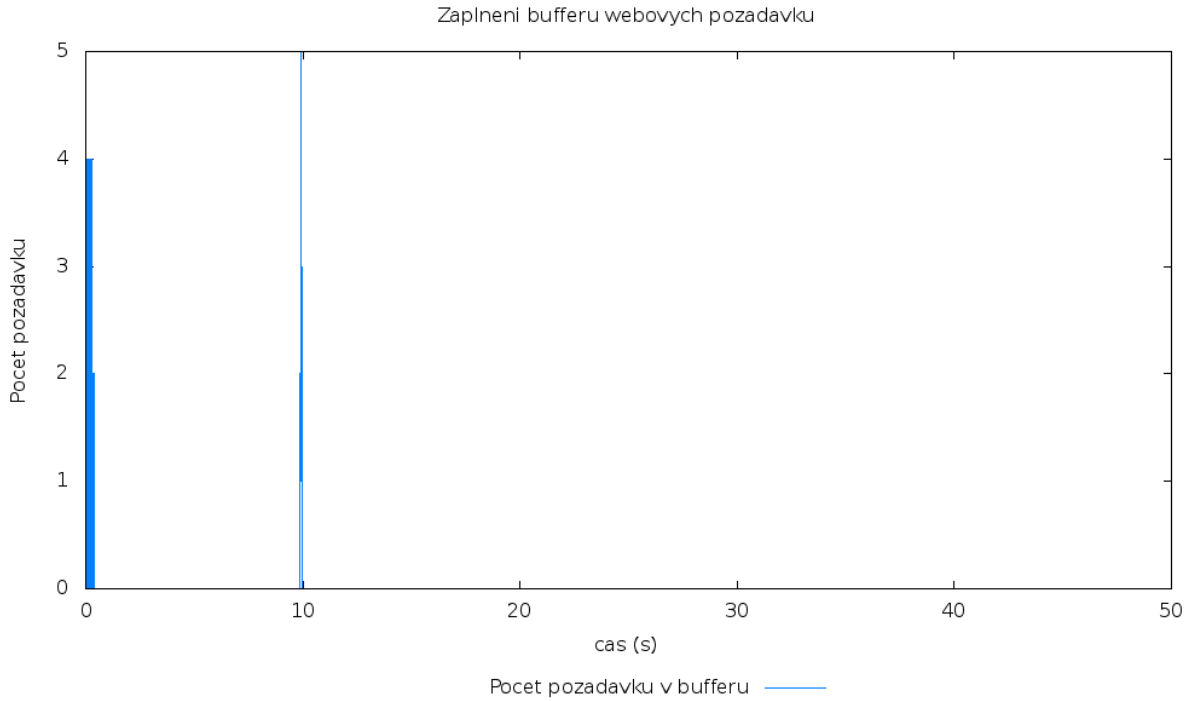
Obrázek 16 - vytížení procesoru při vysoké zátěži a použitím SSD

Jak můžeme na grafu zatížení procesoru pozorovat, tak to je podobné jako u experimentu s vysokou zátěží. Zde tedy žádný větší rozdíl nenalezneme.

Kde se ovšem použití disku projeví, tak je doba odezvy webového serveru (rychlost zpracování požadavků a minimální zaplnění bufferu), jelikož dojde ke značnému urychlení databáze.

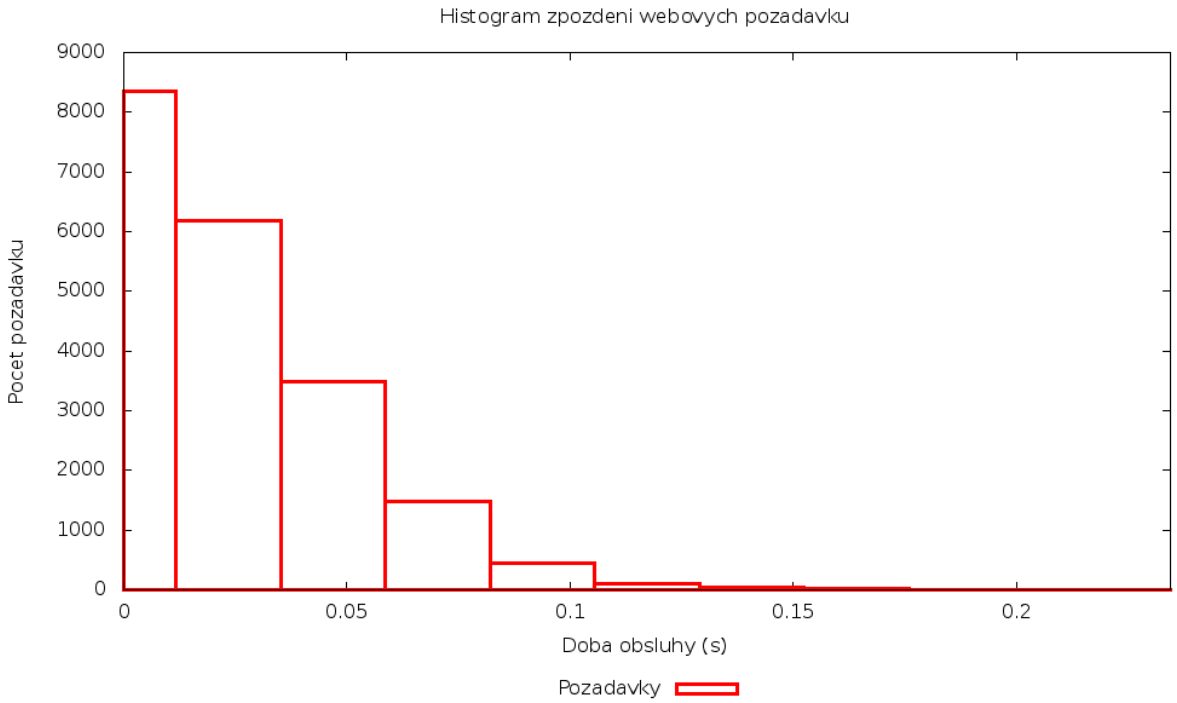


Obrázek 18 - graf reprezentující dobu potřebnou na obsloužení požadavku na webový server



Obrázek 17 - graf ukazuje, že při použití SSD disku se vyřízení požadavků urychluje a nové nemusí být zařazovány do bufferu

Obrázek 19 - histogram ukazující, že při použití rychlé persistentní paměti a zrychlení databáze je většina požadavků obsloužena velice rychle



5.5 Závěr experimentů

Experimenty ukázaly, že systém je nejvíce citlivý na změny parametrů své nejpomalejší komponenty, která byla detekována jako pevný disk (HDD).

V experimentu s SSD diskem je zřetelné významné zrychlení odezvy při obsluze webových požadavků a 8 násobné zvětšení obsloužených dotazů na webové stránky.

6 Shrnutí simulačních experimentů a závěr

Pro sestavení modelu počítačového serveru bylo nutné nastudovat paměťovou hierarchii v počítačích (http://en.wikipedia.org/wiki/Memory_hierarchy).

Dále se seznámit s funkcionalitou webového a databázového serveru. Konkrétněji jak tyto servery používají procesy operačního systému, jak jsou náročné na práci s RAM, zatěžují procesor (respektive jednotlivá jádra) a jak přistupují na disk.

Z analýzy experimentů, detekce slabého místa systému a porovnání současných cen komponent počítače na českých internetových obchodech (http://www.czc.cz/ a http://www.alza.cz/) nám vyšlo, že ekonomicky nejvýhodnějšího zrychlení serveru, který z větší části obsluhuje webové požadavky a používá u toho databázi, je možno dosáhnout při zakoupení SSD disku.

7 Seznam obrázků, grafů a tabulek