Mendelova univerzita v Brně
Provozně ekonomická fakulta

Webová architektura v prostředí vysoké zátěže

Diplomová práce

Vedoucí práce:
Ing. Michael Štencl, Ph.D.

Bc. Jakub Škrha

Brno 2012

Text poděkování

Text prohlášení

místo a datum prohlášení ....

Abstract

Citace práce v anglickém jazyce

abstrakt práce v anglickém jazyce

Abstrakt

Citace práce v českém jazyce

abstrakt práce v českém jazyce

Contents

1 Úvod

Něco na téma jak vzniká vysoká zátěž na iternetu, jaký je vliv internetu, roustoucí zájem o internet a stoupající zátěž. Uvidíme, napíšu to jako poslední.

2 Nežádoucí vlivy a důsledky vysoké zátěže

Úspěch internetových a webových projektů je přímo úměrný výši návštěvnosti, používání a registracím a samozřejmě výdělku z aplikace. Obecně se dá předpokládat, že čím je větší návštěvnost projektu, tím jsou větší i zisky. A to už díky reklamní činnosti či placenných služeb. Ovšem zde se dá velice jasně konstatovat, že tyto výdělky nejsou až tak lehce získané. Nejenom, že si musí aplikace získat své uživatele, ale musí řešit problémy s obrovským počtem uživatelů, čili problémy s vysokou zátěží.

Vysoká zátěž může mít ve své podstatě několik nežádoucích vlivů, které můžou mít až katastrofický scénář. Může docházet k takovému zatížení aplikace, že odpovědi na jednotlivé požadavky mohou trvat velice dlouhou dobu. Tím pádem si uživatel může rozmýšlet, zda-li příště navštíví tuto webovou aplikaci, či zkusí některou z jiných možných konkurenčních alternativ. Takto velice nepříznivý scénář může být ještě horším. A to tak, že díky velkému zatížení dojde dokonce k výpadku celé aplikace, a tím pádem už se uživateli nedostane vůbec žádné odpovědi na jeho požadavek. Při takovém scénáři existuje vysoká pravděpodobnost ztráty a poklesu uživatelů, což může znamenat velký pokles zisků pro firmu či společnost.

Po technické stránce se při velké zátěži vytvoří pro každý požadavek celý samostatný proces či vlákno procesu, které si klade nároky na procesor CPU a operační paměť RAM serveru. Vznikají tak i procesy, které musí čekat na přidělení takových prostředků a tím pádem zatížení, které roste a čeká na vykonání může server přetížit až už nebude provozuschopný. Takovéto zatížení je možné pozorovat při výpisu právě běžících procesů a jejich vytížení na RAM či CPU (například příkazem top). Dále je možno pozorovat tzv. "Load Average" (například příkazem uptime), které představuje počet procesů čekajích na přidělení prostředků během jedné, pěti či patnácti minut.(Kyle Rankin, 2010) Takto je možné sledovat jaké jsou nežádoucí vlivy vysoké zátěže na technické úrovni.

Je nutné podotknout, že existuje i jeden skrytý a ne tak viditelný důsledek vysoké zátěže. Tím je fakt, že jakmile se začne zvedat návštěvnost uživatelů, a tím pádem i zátěž aplikace, projektové vedení si klade požadavek, aby tento nárůst uživatelů již zůstal, a naopak se dokonce i zvětšoval. A to vše z toho důvodu, že velký počet uživatelů, znamená velkou zátěž pro aplikaci, ovšem velký finanční přínos pro firmu.

3 Zneužití vysoké zátěže

Z nežádoucích vlivů a důsledků uvedených v předchozí kapitole vyplývá, že rostocí zátěž může způsobit katastrofické scénáře, čili může negativně působit na celou aplikaci. Tento poznatek představuje obrovské riziko při jeho zneužití. Zpravidla může být způsobeno úmyslným či neúmyslným chováním nějaké organizace či jedince. Takovéto zneužití má svoji oporu i v zákonech, kde hrozí až odmětí svobody několika let.

U webových projektů je možné z těch neúmyslných zmínit například nějaké klientské chyby programů, či větší míru indexace robotů jednotlivých vyhledáváčů. Roboti vyhledávačů pravidelně prochází webovou aplikaci a indexují její jednotlivé části, a tyto výsledky zohledňují následně ve svých výsledcích ve vyhledávání. Tito roboti mohou představovat nežádoucí zátěž.

Tím druhým a daleko nebezpečnějším úmyslným způsobem lze mnohdy způsobit daleko větší škody. Toto úmyslné chování už je klasifikováno jako útok na webovou aplikaci za který hrozí postih podle zákona. Tyto útoky jsou nazývány DoS, neboli "Denial of Service". Jejich cílem je ochromit infrastrukturu celé webové architektury přehlcením požadavků na které aplikace bude vytvářet odpovědi. Tyty útoky využívají chyb, nedostatků či nedokonalostí protokolů ICMP, TCP, UDP a jiných protokolů či samotných webových aplikací. Například útok s názvem "Tcp Syn Flood", kdy útočník vytíží aplikaci SYN pakety pro navázání spojení využívá nedokonalosti TCP protokolu. Dalšími útoky využívajích nedokonalostí mohou být "ICMP Flood", "Ping of Death", "Smurf Attack", "IP Spoofing", "Fraggle Attack", "Teardrop", "Application level" aj. Ovšem proti většině těchto útoků už dnes existuje ochrana ve formě servisních instalací jednotlivých systémů či aktualizací programů síťových zařízení a nebo lehkou konfigurací. (Faisal Khan, 2009)

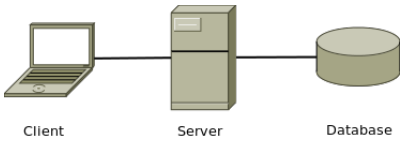
Ovšem co v dnešní době představuje daleko větší nebezpečí, jsou útoky typu DDoS, neboli "Distributed Denial of Service". V tomto případě jako princip obdobný jako u DoS, ovšem s tím, že je úkol distribuovaný. Tedy je spuštěn z několika stanic, několika uživateli a pomocí různých nástrojů. Tento způsob je tedy daleko více organizovaný a daleko více nebezpečný a učinnější. (Faisal Khan, 2009)

Právě v dnešních dnech se stává symbolem boje za svobodu internetu skupina s názvem Anonymous, která využívá útoků DDoS. Při svých útocích využívají například útoky typu "Slowloris", kdy útočník využívá protokolu HTTP na aplikační úrovni a chce celou odpověď na svůj požadavek. Při navázaném spojení ovšem odesílá HTTP hlavičky co nejpomaleji, aby tak co nejvíce prodloužili dobu spojení a získal prostor pro vytvoření dalšího spojení, čili další zátěže. Tato skupina napadá webové aplikace veřejnosti neoblíbených politických stran, vládních organizací, protipirátských asociací a jiných subjektů. Získávají si tím obrovskou podporu ve společnosti i médiích, která s jejich kroky souhlasí. Dokonce pro své útoky využívají i příznivců z řad veřejnosti, kteří nemusí odborníky informačních technologií. Stačí jim si pouze stáhnout upravený program, v určený čas ho spustit a připojit. Útoky probíhají hlášeně či neohlášeně, organizovaně a ditribuovaně.

Otázkou zůstává, kdy jejich konání přeroste z útoků pro dobro společnosti, a stanou se útoky pro vydírání, posílení moci, či za účelem finančního obohacení. V ten moment i společnost, která je v tyto dny podporuje, může pocítit, jak jsou pro ně nebezpeční. I v historii Země nalezneme spoustu skupin, které byly lidmi podporovány a nakonec se z nich stal symbol krutosti, tyranie, úzkosti a neštěstí. Proto je důležité jejich útoky nepodceňovat a umět se bránit. Má práce se nezabývá konkrétním řešením nějakého z typů útoků, ale zabývá se obecně vysokou zátěží, a jak ustát narůst obrovské zátěže a tedy i nějaký útok.(Pavel CČepskyý, 2012)

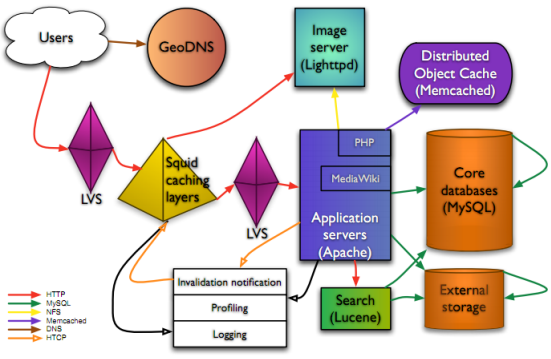
4 Tří a vícevrstvá architektura webových systémů

Webová architektura je ve svém základu třívrstvá. První vrstvu tvoří klient, neboli uživatel se svým Hardware a Software, a svými aplikačními požadavky. Druhou vrstvu tvoří aplikační server, který zpracovává požadavky aplikace, tedy požadavky klienta, zpracuje tento požadavek, vytvoří odpověď a zašle zpět klientovi. Ovšem k tomu, aby mohl tuto odpověď vytvořit, potřebuje i data aplikace, která jsou uloženy v perzistentní databázi, která tvoří třetí a poslední vrstvu třívrstvé architektury. Každá z vrstev, tedy prezentační, aplikační i datová má své místo a svou správu v aplikaci.(Jaroslav Zendulka, 2005)



Obr. 1: Tří vrstvá architektura webových aplikací

V architektuře webových aplikací s vysokou zátěží už je potřeba jiného přístupu. V tomto případě se dá říci, že je třívrstvá architektura nedostačující. Je potřeba počítat se síťovými prvky pro load balancing, s více aplikačními stroji, s databázovými replikacemi, s DNS řešením pro geografické rozdělení zátěže, s CDN pro rozdělení zátěže přidělování obsahu, s vrstvami pro cache aplikace a s dalšími vrstvami pro backendové či frontendové aplikace a služby. V tomto případě neexstuje žádné jasně dané a pevné řešení, každá aplikace si s sebou nese své individuální a charakteristické řešení a strategii, i když některé osvědčené postupy se opakují. Tyto strategie už nesou název vícevrstvá architektura.



Obr. 2: Webová architektura společnosti Wikimedia provozující Wikipedia.org

Nutkno podoknout, že webové architektury využívají nejčastěji ke své komunikaci mezi klientem a architekturou protokol HTTP, který využívá portu číslo 80 a protokolu TCP pro komunikaci. Proto je celá má odborná studie založena na práci s tímto protokolem.

5 Aplikační vrstva

Aplikační vrstva představuje jádro webové architektury. Jejím účelem je příjmout a zpracovat klientův požadavek, vytvořit odpověď a tuto odpověď zaslat nazpět klientovi. Na aplikační vrstvu jsou tak kladeny úkoly celé režie procesu tvorby odpovědi, a tím pádem má velkou zoodpovědnost a mnohdy i největší zátěž.

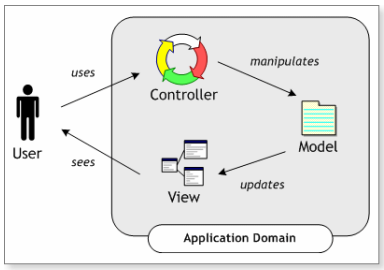
5.1 Webová aplikační architektura MVC

V dnešní moderní aplikační vrstvě se používá aplikační architektura návrhového vzoru MVC pro přehlednější a rychlejší způsob tvorby aplikace. Tato zkratka vychází z tří slov Model, View a Controller, které představují tři základní vrstvy aplikační architektury. Často bývá označovám i jako MVC framework, který rozděluje aplikaci do tří modulů. (Rudolf Pecinovskýy, 2007)

Controller je prvotní inicializační vrstva každého požadavku. Zpracovává příchozí data, parametry a atributy dané akce od uživatele, provádí jejich kontrolu a formátování. Stará se i o zabezpečení dané konkrétní akce vrstvy Controller. Často spolupracuje s vrstvou Model, které předává požadavky na data aplikace, a tyto data dále zpracovává pro předání do vrstvy View. (Rudolf Pecinovskýy, 2007)

Model má za úkol přistupovat k datovým úložištím, a to ať už k perzistentní databázi nebo souborovému systému, cache či jiným typům úložišť. Zapouzdřuje tak datovou logiku frameworku. Často se jedná o soubor dalších návrhových vzorů, kde se může vyskytnout přepravka (Crate) či jejich kolekce pro přenášení dat, zástupce (Proxy) pro přístup k implemtacím nad přepravkami, příkaz (Command) pro vykonání nějaké akce či příkazu, strategie (Strategy) pro určení nějaké konkrétního algoritmu ze skupiny algoritmů nad určitou úlohou, a mnohé další z návrhových vzorů. Modelová vrstva bývá označována za nejsložitější vrstvu, a právě proto je potřeba dodržovat techniky OOP včetně návrhových vzorů pro další možnou rozšiřitelnost a pro přehlednost. (Rudolf Pecinovskýy, 2007)

View klade důraz na presentační úroveň, tedy na grafickou a jinou interakci s uživatelem. Zpracovává tak výsledek práce vrstvy Controller nad vrstvou Model a zobrazuje výsledek určitých operací. Často využívá nějakých šablonovacích přístupů. (Rudolf Pecinovskýy, 2007)



Obr. 3: Návrhový vzor MVC a jeho životní cyklus

5.2 Optimalizace aplikační vrstvy

Optimalizace na úrovni aplikační vrstvy může mít několik způsobů a přístupů. Tato činnost se týká převážně programátorů a softwarových inženýrů, kteří mají za úkol vývoj a údržbu aplikační vrstvy. K tomu, aby se dali identifikovat problematické části pro optimalizaci slouží tzv. profilery. Ty mají obecně za úkol vyprofilovat jednotlivé funkce, metody, procedury, dotazy a příkazy, které se na dané vrstvě, již je profiler určen, vyskytují, a určit jejich dobu trvání, počet volání, čas spuštění, závislosti a další parametry. Profilování, neboli určení kandidátů pro optimalizaci, je prvním a nejdůležitějším krokem pro optimalizaci aplikační vrstvy webové architektury. Další kroky se týkají především těchto oblastí:

- Výběr nejoptimálnějšího algoritmu pro danou úlohu
- Výběr nejrychlejšího MVC frameworku
- Vytváření cache souborů aplikace
- Způsob překladu a vykonání zdrojových souborů
- Přidání další vrstvy architektury - aplikační cache

5.3 Druhy aplikačních vrstev

Existuje celá škála různých programovacích jazyků a webových serverů pro implementaci aplikace. Každý z nich má své výhody a nevýhody, specifická řešení a přístupy. Uvádím zde krátký seznam těch v praxi nejběžněji se vyskytujících:

- Webový server Apache2 s programovacím jazykem PHP
- Java Servlets, Java Spring Source
- C# s technologií .NET
- Ruby on Rails
- Python a Django
- a mnohé další

5.4 Ajax a webové služby

Ajax, neboli Asynchronous JavaScript and XML, se dnes stává nedílnou součástí při vývoji webových aplikací. Aplikace tak dostávájí interaktivnější charakter bez nutnosti znovuzaslání celého požadavku webové aplikaci. Celý tento přístup probíhá nejvíce na straně klienta. Je použito javascriptu pro programovou implementaci, který má přístup ke stromu objektů dokumentu zvaného DOM, neboli Document Object Model. Do aplikační vrstvy jsem se rozhodl přidat AJAX z toho důvodu, že používá objekt XMLHttpRequest pro komunikaci s aplikačním serverem. Tyto

aplikační požadavky jsou nazývány webovými službami pro Ajax. Tyto požadavky jsou vykonávány na aplikační vrstvě a představují potenciální zátěž, která musí být i v některých případech optimalizována.(Brett McLaughlin, 2005)

6 Dabázová vrstva

Úkolem databázové vrstvy ve webové architektuře je zajišťovat datové služby a uchovávat tak aplikační data perzistentní. V oblasti webových architektur se nejčastěji vyskytují relační databázové systémy, a proto i má práce je soustředěna na tento typ databázových systémů. Databázový systém obecně tvoří databáze, jakožto skupina strukturovaných homogenních souborů, a SŘBD, neboli Systém řízení báze dat, jakožto integrovaný softwarový prostředek řídící bázi dat.

6.1 Optimalizace SQL dotazů

Optimalizace dotazů SQL je nedílnou součásti procesu práce s databázovým systém v prostředí vysoké zátěže. Je totiž důležité nejenom si umět získat potřebná data, ale je potřeba zvážit i za jakou cenu tyto data prostřednictvím databázavého systému získáváme. Hovoříme-li o webových architekturách s vysokou zátěží, je tento proces optimalizace velice důležitý. Každá operace, každý dotaz, každá akce potřebuje ke své realizaci určité hardwarové a systémové prostředky, a v prostředí vysoké zátěže je důležité ušetřit co nejvíce těchto prostředků.

K tomu, abychom mohli vůbec přistoupit k optimalizaci SQL dotazů, je potřeba určit a identifikovat, které tyto dotazy jsou opravdu náročné na prostředky a čas, neboli mají vysokou cenu. K tomu slouží tzv profilery (viz. kapitola 5.2). Profilery mohou být určeny pro aplikační vrstvu, kde profilují nejenom zdrojové kódy aplikace, ale samozřejmě i databázové dotazy, které jsou z této aplikační úrovně spuštěny. Tímto způsobem je možné získat přehled všech operací, které probíhají na aplikační i databázové vrstvě, poněvadž tyto vrstvy spolu neúzce souvisí a spolupracují. Další možností je použít profiler určený přímo k databázové vrstvě. Takový profiler pak profiluje pouze databázovou vrstvu, jednotlivé databázové dotazy, jejich cenu, dobu trvání, a jiné další statistiky.

Každý SQL dotaz má nějaký svůj exukuční plán. Databázový systém po obdržení SQL dotazu vybírá z několika možných exekučních plánů ten nejoptimálnější, který je po té v databázi proveden. Při výběru exekučního plánu je brán v potaz výběr indexu a způsob skenu tabulek, vybraná spojení, aj. Exukuční plán je možné zobrazovat v mnoha databázových systémech pomocí EXPLAIN a identifikovat tak místa exekučního plánu, která mohou být kandidátem pro optimalizaci.(Bohdan Blaha, 2007)

Pro optimalizaci SQL dotazů je možné určit několik základních oblastí, na které je možné se zaměřit při konkrétní optimalizace určitého SQL dotazu:

- Normalizovaný databázový návrh
- Vnořené SQL dotazy
- Indexace, výběr indexu a způsob prohledávání
- Výběr druhu a pořadí spojení

- Způsob používání podmínek, klauzulí a operátorů

6.2 Indexace

Indexace je důležitá a nejefektivnější optimalizace dotazů SQL. Při průchodu dat tabulkou má databáze na výběr několik možností prohledání. První možností je prohledat všechny řádky tabulky podle sql podmínek. To je nazýváno obecně Full Table Scan, nebo také Sequence Scan, neboli sekvenční prohledávání. Další možností je pužití některého z indexů pro přístup k hodnotám namapovaných na jejich ROWID, které ukazuje na fyzické uložení. Toto prohledání bývá nazýváno jako Index Range Scan, nebo jen obecně Index Scan, neboli indexační prohledávání. Samozřejmě prohledávání tabulek pomocí indexace je výrazně rychlejší a tím pádem důležité pro opimalizaci SQL. (Bohdan Blaha, 2007)

Indexy jsou fyzicky i logicky uloženy v asociativních tabulkách, a díky tomu tak i odděleny od datových tabulek. Čili při smazání indexů se nesmiží ani nijak neovlivňí datové tabulky. Pouze se může zpomolit přístup k datům, který byl rychlejší pomocí těchto indexů. Tabulky s indexy jsou samozřejmě uloženy na disku, poněvadž jejich velikost je obrovská a nevešly by se do operační paměti RAM. Operační paměť a přístup k ní je daleko rychlejší než přístup k datům uložených na disku, a proto je potřeba volit nějaký vhodný algoritmus prohledání a přístupu k indexům, a od toho se odvyjí i název a druh používaných indexů. V každém databázovém systému samozřejmě naleznete některé typické a některé atypické druhy indexů. (Bohdan Blaha, 2007) Zde je krátký výběr možných indexů:

- B-tree - pro přístup pomocí Root-Node-List
- Bitmap - pro výčtové sloupce
- R-tree - typ indexu optimalizovaný pro geometrická data.
- GiST - zobecněný vyhledávací strom
- a další

6.3 Partitioning

Partitioning, který je občas do češtiny překládán jako segmentace, občas jako škálování, slouží v relačních databázových systémech k rozdělování tabulek a indexů do menších částí a komponent. Díky tomu je pak činnost databáze rychlejší a snadnější. Při této segmentaci tak může dojít k rozdělení tabulek i na více pevných disků či serverů. Tyto segmenty jsou na sobě nezávisle, ale přitom je k nim možné přistupovat přes tabulku, pro kterou byla segmentace vytvořena. Databázová tabulka a její vlastnosti, jako například referenční integrita nebo žádná redundance, jsou stále zachovány a fungují přes všechny její segmenty. Dokonce i když dojde k selhání či výpadku jednoho ze segmentů, ostatní jsou stále přístupné a je možné s

nimi pracovat. Partitioning je možné provádět na několika úrovních a podle různých klíčů. U segmentovaných tabulek je tak důležité si rozmyslet jakou strategii si zvolit.

Při vertikální segmentaci dojde k segmentaci podle definovaných sloupců databázové tabulky. Klíčem při tomto rozdělení je určení sloupečků, které se nepoužívají ve where klauzuli, nebo jsou prázdné či zřídka používané.

Častěji používáným přístupem je horizontální segmentace tabulek, čili segmentace podle řádků. Zde se segmentují řádky, podle určité hodnoty databázového sloupce. To do jakého segmentu bude řádek tabulky vložen rozhoduje nějaký interval, či hodnota výčtu a nebo nějaká funkce.

Další možností je aplikační úroveň segmentace, která se ne často objevuje v souvislosti s Partitioning. Nejedná se totiž o segmentaci určité databázové tabulky, ale o segmentaci databáze. Část tabulek je umístěna na jednom serveru, část na dalším serveru, a tak dále.

Partitiong je důležitým nástrojem při optimalizaci databázové vrstvy v architektuře webových systémů v prostředí vysoké zátěže. Dá se totiž předpokládat, že se zvětšující se zátěží roste i počet záznamů tabulek, a tak se doba přístupu zvětšuje a prostředky zatěžují ještě víc. Tyto problémy dokáže vyřešit partitiong. (Eli White, 2009)

6.4 Replikace

Replikací rozumíme technologii, kdy je možné nasadit více databázových serverů v rámci jedné databáze. Jedná se tak o sdílení dat mezi více hardwarovými, softwarovými a jinými prostředky a jejich přenositelnost. Účelem replikace je tak dosáhnout vysoké dostupnosti databázového systému a škálování výkonu pro optimalizaci v prostředí vysoké zátěže. Obecně existují dvě základní varianty databázových replikací od kterých se odvyjí jejich další využití. Samozřejmě v závislosti na konkrétním databázovém systému pak existují další členění a nastavení.

Replikace varianty master-slave je podporována ve většině databázových systémech. Jedná se o jednodušší tzv. jednosměrnou replikaci. V této variantě je určen autonomní prvek, jedna replikace, která akceptuje a zpracovává požadavky na změny. Takováto replikace nese název master. Prvek s názvem slave je věrnou kopií autonomního prvku master. Slouží pouze ke čtení a může jich být více pro jeden master. Jakmile master obdrží a zpracuje požadavek na změnu, tak jej po dokončení přenese na ostatní slave replikace.

Replikace typu master-master bývá označována jako obousměrná. To znamená, že jsou v rámci jednoho databázového systému minimálně dvě replikace typu master, které akceptují všechny druhy požadavků na změny i čtení a přenáší je vzájemně mezi sebou. Z této vlastnosti vyplývá, že může dojít ke kolizím, kdy například dvě replikace master zapisují do stejné tabulky. Takovéto kolize jsou nevyhnutelné, a je potřeba je řešit.

Způsob přenosu mezi jednotlivými replikacemi může být synchronní či asynchronní. U synchronního přenosu se čeká až se změny provedou na všechny os-

tatní repliky. Takovýto proces je časově náročný, ovšem na druhou stranu je celý databázový systém konzistentní jako celek. U asynchronního přenosu se nečeká na dokončení přenosu mezi ostatními replikacemi. Díky tomu je celý databázový systém rychlejší, ovšem může dojít k nekonzistenci, kdy na ostatní replikace ještě nejsou přenesena všecha data.

Administrace, nástroje a konfigurace replikací jsou zabudované v téměř každém databázovém systému. Je důležité ale poznamenat, že tyto nástroje nejsou mnohdy dostačujícími řešeními pro architektury v prostředí vysoké zátěže a je proto nutné používání jiných doplňkových nástrojů. Také je více než důležité říct, že v prostředí vysoké zátěže se webová architektura bez databázových replikací jen těžko obejde.(Tomaášš Vondra, 2011)

6.5 Druhy relačních databází

V dnešní době existuje několik druhů relačních databázových systémů. Každý z nich má své klady a zápory, ovšem princip a způsob práce těchto databází je v základu podobný. Uvádím zde přehled těch v praxi se běžně vyskytujících:

- Oracle
- MySQL
- PostgreSQL
- MSSQL
- Firebird
- a mnoho dalších

7 Webová cache

Proč webovou cache, proč se používá a k čemu je dobrá. Urychlení odezvy, odlehčení jiným požadavkům na aplikaci, apod.

7.1 Druhy cache

Popsat jednotlivé druhy browser, proxy, reverse proxy a aplikační cache

7.2 Typy obsahu

Popsat typy obsahu pro cache. Dynamický vs. statický

7.3 Proxy cache a cache prohlížeče

Klientská část, kde a kdo ji instaluje a kdy a jak se používá.

7.4 Reverzní proxy cache

Serverová část, kde a kdo ji instaluje a kdy a jak se používá.

7.4.1 Nginx

Popis NGINX a proč jsem ho vybral.

7.5 Aplikační a distribuovaná cache

Popis aplikačních a distribuovaných cache, jaký je jejich smysl a kde je jejich použití

7.5.1 Memcached

Popis Memcached a proč jsem ji vybral

8 Další vrstvy aplikace

K čemu jsou další vrstvy

8.1 CDN

Content delivery network obrázky a stream.

8.2 NoSQL Databáze

K čemu slouží a kde najdou své uplatnění.

8.3 Vyhledávání

Z vyhledávání se také dělá další vrstva.

9 Virtualizace

Projekty dnes neběží vždy na jednom serveru, ale na více virtualizovaných serverech. Proč tomu tak je.

10 Load balancing

Nevím jestli k této kapitole se vůbec dostanu, uvidíme. Každopádně serverů bývá vždy několik a jak zajišťovat toto rozložení zátěže.

11 Cloud Computing

Budoucnost projektů, startupů, vše řešeno cloudem. AWS

12 Praktická část s experimenty a výsledky

Úvod do toho, že se budu praktickou částí snažit dosáhnout nasymolování vytížené webové architektury a optimalizovat jednotlivé vrstvy v rámci možností.

12.1 Aplikace a její vrstvy

Představení aplikace, její síťové schéma, jednotlivé vrstvy s popisem, domény, apod.

12.2 Testovací nástroje

Popis toho co sleduji testovacími nástroji

12.2.1 XHProf

K profilování

12.2.2 Siege

Pro generování zátěže

12.2.3 PostgreSQL Explain

Vysvětlení sql dotazů

12.3 Aplikační vrstva PHP

Ve své aplikační vrstvě jsem zvolil pro implementaci programovací jazyk PHP běžící na webovém serveru Apache.

Webový server Apache je jedním z nejrozšířenějších a nejpopulárnějším webovým serverem na internetu. Byl implementován v roce 1996 v jazyce C++. Jeho instalace, konfigurace a administrace není nikterak složitá. Na spoustě webových hostingů je dostupný v základní konfiguraci. Je to volně použitelný produkt, který obsahuje spoustu různých přídavných módů. Z těchto důvodů jsem ho vybral pro praktickou část své diplomové práce.

Programovací jazyk PHP se stal jedním z nejpoužívanějších programovacích jazyků pro svoji srozumitelnost, přenositelnost a jednoduchost. Je to dynamicky typovaný programovací jazyk, čili i z těchto důvodů je hodně ohebný. Plně podporuje OOP přístup, čili je možné vyžívat těchto technik včetně návrhových vzorů, které jsou pro složité webové aplikace v prostředí vysoké zátěže velice důležité.

12.3.1 Optimalizace PHP pomocí APC

Jak se chovala aplikace bez APC a co dosahnu APC

12.3.2 Dosažené výsledky

Toto bude vždy na konci každého experimentu, grafy, časy, screeny, apod.

12.4 Databázová vrstvy

Tady se budeme snažit optimalizovat databázi.

12.4.1 Optimalizace databáze

Co jsem použil pro optimalizaci

12.4.2 Dosažené výsledky

Toto bude vždy na konci každého experimentu, grafy, časy, screeny, apod.

12.5 Aplikační cache

Tady se budeme snažit optimalizovat databázy.

12.5.1 Optimalizace aplikace pomocí Memcached

Co jsem udělal s Memcached, jednotlivé vrstvy modelu, popis toho čeho chci dosáhnout.

12.5.2 Diagram tříd pro aplikaci s podporou Memcached

Diagram tříd mé aplikace

12.5.3 Dosažené výsledky

Toto bude vždy na konci každého experimentu, grafy, časy, screeny, apod.

12.6 Reverzní proxy cache

Tady se budeme snažit použít reverzní proxy cache.

12.6.1 Nasazení a konfigurace NGINX s Memcached

Jak jsem co dělal s NGINX, problémy, řešení, návrh architektury a jak to ovlivňuje aplikační vrstvu

12.6.2 Význam Ajax a webových služeb pro NGINX

Ajax komunikuje s NGINX, proč a jak.

12.6.3 Dosažené výsledky

Toto bude vždy na konci každého experimentu, grafy, časy, screeny, apod.

13 Diskuze

Diskutovaná řešení, jak je možné je kombinovat, apod.

14 Závěr

Závěr ve smyslu nákladů a přínosů, kdy je lepší co. Uvidíme, napíšu jako poslední.

15 Literatura

Kyle Rankin *Linux Journal: Hack and / - Linux Troubleshooting, Part I: High Load* [online]. Dostupné z: http://www.linuxjournal.com/magazine/hack-and-linux-troubleshooting-part-i-high-load.

Faisal Khan *DOS ATTACKS: Dos Attacks Overview - What are DoS attacks* [online]. Dostupné z: http://dos-attacks.com/what-are-dos-attacks/.

Pavel Čepský *Lupa.cz: Útoky jménem Anonymous: Jak se rodí hackeři?* [online]. Dostupné z: http://www.lupa.cz/clanky/utoky-jmenem-anonymous-jak-se-rodi-hackeri/.

Doc.Ing.Jaroslav Zendulka,CSc. *VUT-FIT: 10. Architektura klient/server a třívrstvá architektura* [online]. Dostupné z: http://www.fit.vutbr.cz/study/courses/DSI/public/pdf/nove/10_clsrv.pdf.

Brett McLaughlin *Ibm develper works: Mastering Ajax* [online]. Dostupné z: http://www.ibm.com/developerworks/web/library/wa-ajaxintro1/index.html.

Rudolf Pecinovský *Návrhové vzory : 33 vzorových postupů pro objektové programování* 1. vyd. Brno: Computer Press, 2007. 527 s. ISBN 978-80-251-1582-4.

Bohdan Blaha *SQL Optimalizace v Oracle* Praha: Unicorn College, 2010. 47 s. Dostupné z: http://www.unicorncollege.cz/katalog-bakalarskych-praci/bohdan-blaha/attachments/Blaha_Bohdan_-_Optimalizace_SQL_dotaz%C5%AF_v_datab%C3%A1zi_Oracle.pdf.

Eli White *Habits of Highly Scalable Web Applications* DCPHP Conference 2009.

Tomáš Vondra *Replikace v PostgreSQL* CSPUG Konference 2011.