Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Optimalizace webového serveru pro výkonové zatížení

Štěpán Staniek

Vedoucí práce: Ing. Pavel Štěpán

12. května 2015

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Pavlu Štěpánovi za předání cenných zkušeností při tvorbě bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Staniek, Štěpán. *Optimalizace webového serveru pro výkonové zatížení.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce je vytvořena za účelem zjištění dostupného výkonu testovaného systému a na něm běžících aplikacích. V obsahu práce jsou diskutovány možnosti testování zátěže webového serveru. Provedeny jsou analýzy požadavků na webovou aplikaci, na jejichž základě jsou vytvořeny testovací scénáře. Z testovacích scénářů jsem navrhl zátěžové testy, které simulují příchozí uživatele na testovaný systém. Testy jsou navrženy tak, aby co nejvěrněji odpovídaly provozu s reálnými uživateli v plném provozu systému. Na základě výsledků z provedených zátěžových testů na testovaném systému je navrženo doporučení pro zadavatele.

Klíčová slova Zátěžový test, test hraniční zátěže, výkonnostní test, JMeter, dedikovaný server, optimalizace webového serveru

Abstract

This bachelor thesis is created in order to identify the available computation power of a tested system and applications running on it. It describes the available possible solutions for a web server load testing. Analysis of requirements for a selected web application was performed and it's results were used to

create multiple testing scenarios. Various load tests that simulate a user behaviour were designed based on testing scenarios. Results of load testing are at the end used to create a set of recommendations to improve the overal system performance.

Keywords Load test, stress test, performance test, JMeter, dedicated server, web server optimalization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tento konkrétní požadavek na provedení zátěžového testování vznikl za situace, kdy jsem se chystal na dokončení bakalářského programu a zároveň jsem pracoval jako webový vývojář v nejmenované firmě. Zde jsem byl členem týmu, který vyvíjel nové webové aplikace pro komerční užití. Proces vývoje webových aplikací byl úspěšně dokončen a začali jsme připravovat proces testování před pilotním provozem webové aplikace. Zde se mi nabídla možnost spojit pracovní úkol a téma pro mou bakalářskou práci. Zátěžové testování výkonu aplikací je nedílnou součástí vývoje softwarových řešení. Testovaný systém testujeme tak, že simulujeme předpokládané chování uživatelů při používání aplikací na tomto sytému. Zjišťujeme, jak se aplikace na testovaném systému chovají na aplikační úrovni, jaké systémové zdroje využívají a jak moc jich potřebují. Dále nás zajímají odezvy aplikací směrem k uživatelům. Pomocí naměřených výsledků jsme schopni vyhodnotit a ověřit si, zda námi vyvinuté webové aplikace splňují očekávané chování nebo nikoliv. Jsme schopni takto odhalit výkonnostní problémy, úzká místa, paměťové úniky, hardwarové nedostatky apod. Mnou navržené zátěžové testy jsou navrženy tak, aby je bylo možné přenášet na nové testované systémy. Výstupem této práce je doporučení pro zadavatele práce, kde bude uvedeno stanovisko, zda konkrétní webový server, na kterém bylo provedeno měření, splňuje požadavky zadavatele na testovaný systém.

Kapitola 1

Cíl práce

Cílem této práce je zjistit jaké výkonové nároky má webová aplikace zadavatele na testovaný systém. Zjištění můžeme nalézt tři. Nedostatečný výkon testovaného systému, který nedokáže efektivně provozovat webovou aplikaci. Dalším zjištěním je přesným opak, a to že webová aplikace využívá pouze zlomek výkonu testovaného systému. Posledním zjištěním by měl být současně konečný stav pro zadavatele. Jedná se o variantu, kdy testovaný systém poskytuje dostatečný výkon na provoz webové aplikace podle jeho požadavků. Výkon testovaného systému je podroben zátěžovým testům, které mají různou úroveň zatížení. Z naměřených výsledků testování lze určit náročnost webové aplikace na testovaný systém. Návrh a realizace zátěžových testů počítá s dalším použitím testů. Testy by měly být v krátkém časovém úseku znovu použitelné pro testování stávajícího, či nového serveru. Výstupem této práce by mělo být doporučení, zda konkrétní webový server, na kterém bylo provedeno měření, je schopen v přijatelné době obsloužit požadovaný počet uživatelů.

Kapitola 2

Popis testované aplikace

Testovaná aplikace je určena pro pacienty s diabetem (cukrovkou), především I. typu, kteří potřebují nebo chtějí lépe kompenzovat svou nemoc a na dálku více komunikovat se svým lékařem. Služba umožňuje intenzivnější monitoring pacientů. Základem je získávání úplných a přesných hodnot hladiny glukózy v krvi. Naměřená data jsou automaticky odesílána do webové aplikace, kde jsou okamžitě přístupná pro oprávněné uživatele. Pacient a lékař tak mohou na dálku konzultovat průběh léčby, přičemž oba mají k dispozici aktuální i dřívější naměřené hodnoty glykemie.

Kapitola 3

Analýza kritických procesů webové aplikace

Pro správné definování našich problémových „kritických“ procesů je potřeba se vžít do role uživatelů, kteří budou aplikaci používat den co den. Jen tak jsme schopni definovat ty správné procesy, u kterých může v běžném provozu dojít k problému. Aplikace je navrhována pro více rolí. Ty, které nás zajímají, jsou především lékař a pacient. Tyto dvě role zastupují v aplikace nejpočetnější skupiny a představují celkově stovky uživatelů, což je požadavek zadavatele. Návrh zátěžového testování jsem ale nestavěl pro tak velký počet uživatelů. Vybereme pouze pravidelnou činnost těchto rolí, kterou musejí provádět vybraný den v aplikaci. Pravidelnou činnost uživatelů provedeme ve vybraném časovém úseku, čímž jsme schopni zjistit chování aplikace v tu nejvytíženější dobu, které nás zajímá.

3.1 Lékaři přistupují do aplikace současně v jeden okamžik

Role lékař představuje kvalifikovanou osobu schopnou provádět kontroly svých pacientů. Všichni lékaři v systému by pro zátěžové testování měli mít ideálně přidělený stejný počet pacientů. Z celkového množství svých pacientů pak lékaři každý den provádějí pravidelné kontroly pouze části svých pacientů. Jedná se především o pacienty s naměřenou kritickou hodnotou, kteří urgentně potřebují kontrolu stavu. Lékař kontroluje každého pacienta jednou za 14 dní. Pokud lékař provede kontrolu všech pacientů s kritickou hodnotou, přechází na kontrolu těch pacientů, kteří byli kontrolování nejpozději. Samotná kontrola pacienta lékařem spočívá v několika krocích. Na profilové stránce pacienta přečte poslední zapsané záznamy o stavu pacienta, které napsal z již dříve provedených kontrol pacienta. Cílem je si připomenout stav daného pacienta. Po přečtení záznamů přechází lékař na kontrolu schématu měření pacienta.

Zde se podívá, jestli pacient provedl všechna měření, které mu lékař stanovil. Dále lékař přechází na kontrolu naměřených dat v podobě grafů. Křivky jsou uvedeny v závislosti na čase, respektive každých sedm, třicet a devadesát dní. Pokud potřebuje lékař načerpat ještě další informace, je mu k dispozici pod křivkami přehled statistických údajů o záznamech monitorování. Nyní by již měl mít lékař k dispozici všechny potřebné informace o stavu pacienta. Na základě těchto nabytých informacích pak napíše nový záznam o aktuálním stavu pacienta a jeho léčbě. Po uložení záznamu v aplikaci přechází lékař na kontrolu dalšího pacienta v pořadí.

3.2 Pacienti prochází svůj profil v aplikaci a komentují naměřené hodnoty vymykající se normálu

Z pohledu chování role pacienta v aplikaci docházíme k následujícímu procesu, jehož kritickým místem je určitě ukládání komentářů. Pacient přichází na přihlašovací stránku aplikace, zde provede zadání přihlašovacích údajů. Po autentizaci pacienta dojde k zobrazení jeho profilové stránky. Zde si může pacient přečíst lékařem napsané záznamy k jeho aktuálnímu stavu a doporučené léčbě. Pokud se to po pacientovi žádá, vkládá pacient komentář o svém aktuálním stavu zdraví a probíhající léčbě. Pacient přichází primárně do aplikace proto, aby zde okomentoval své naměřené hodnoty glykémie. Především jde o hodnoty, které výrazně překračují normální mez a indikují hypo nebo hyperglykémii. Napsaný komentář pacient ukládá do aplikace. Pokud se nachází více takovýchto hodnot, je mu doporučeno je také okomentovat. Také se po pacientovi žádá, aby doplnil časový údaj, kdy provedl měření po jídle. Pro přehled svého zdraví si může pacient zobrazit křivky v časové závislosti. Touto činností pacient zpravidla končí a aplikaci opouští. Předpokládáme, že tuto činnost pacient ideálně provádí jednou za dva dny.

3.3 Pacienti odesílání naměřená data do aplikace jednou za den

Jako poslední kritický proces vidíme v odesílání naměřených dat pacientem do webové aplikace. V krátkém okamžiku zde může dojít k odeslání stovek naměřených hodnot do aplikace. Tento proces by mohl významně zatížit chod webového serveru, na kterém aplikace běží, jelikož dochází k zapisování do MySQL databáze. Proces jsme navrhli tak, že pacienti provádí každý den pět měření glykémie. Bohužel v časové tísni přes den nemají čas odesílat naměřená data do aplikace. Proto až po provedeném posledním pátém měření pacient odesílá všech pět naměřených hodnot do aplikace. Simulujeme tedy nárazovou zátěž, která je na serveru testována.

Kapitola 4

Návrh optimální hardwarové konfigurace serveru na základě informací zjištěných z analýzy webové aplikace

Vývojáři aplikací se snaží dávat velký důraz na líbivý a efektivní design webových aplikací, ale je snadné ignorovat důležitou a velice prostou pravdu, že většina obsahu webových aplikací je statická. To si můžeme ověřit pohledem na velmi široké spektrum internetu, kde statický obsah převažuje. Od firemních webových stránek přes osobní stránky známých osobností až po běžné zpravodajské kanály. Jak vlastně vypadá neefektivní práce se statickým obsahem? Příkladem je vypršení platnosti příchozího požadavku, který ukončil webový server a statický obsah požadované stránky nebyl stažen úplně celý. K načtení stránky tedy můžou chybět obrázky, odstavce či oddíly, které na stránce obvykle zabírají velkou plochu. Načtená stránka bez těchto prvků ztrácí požadovaný vzhled a stává se neefektivní. Pokud bychom předpokládali, že máme dobře kódované stránky, můžeme zajistit, že obsah se načte okamžitě a objemné prvky, jako obrázky, se načtou s mírným zpožděním. Tohle je důležité pro uživatele s pomalým připojením, ovšem tato úprava nemůže být řešením našeho problému. Obrázek je mnohdy klíčovým prvkem, bez kterého význam stránky ztrácí smysl, ať je stránka sebelepší. Abychom tedy pochopili, co je efektivní a co není, je potřeba si stanovit konkrétní výkonnostní cíle.
Dle požadavku zadavatele chceme, aby webová aplikace byla schopna obsloužit během jednoho dne:

- 20 doktorů na webu
- 100 pacientů na webu
- 200 glukometrů zasílá data

Tyto požadavky v našem testování zastupují scénáře, které jsou přesně definovány. Práce jednoho doktora je vyjádřena do 112 HTTP požadavků. Činnost pacienta ve webové aplikaci je zastoupena 10 HTTP požadavky a činnost jednoho glukometru je 5 HTTP požadavků. Přepočtení do HTTP požadavků podle zadaných požadavků zadavatele na testovaný systém je 4240 HTTP požadavků za den. Nyní předpokládáme, že během jedné hodiny dojde k obsloužení 15% veškerého provozu, takže počet HTTP požadavků za hodinu je přibližně 636, což je 0.17 stránky za vteřinu. Průměrná doba načtení jedné stránky by se měla pohybovat do pěti vteřin, aby se splnily požadavky na testovaný systém.
Námi testovaná webová aplikace využívá statický i dynamický obsah. Provedl jsem analýzu vybraných stránek, abych určil, jaký typ obsahu se na stránce vyskytuje.

Název stránky	**css**	**js**	**png**	**gif**	**Doba načtení**	**# pož.**	**Staž. dat**
Monitorace	7	10	17	15	1130 ms	51	725.40KB
Monitorace(filtr.)	7	10	17	15	1465 ms	51	725.54KB
Profil pacienta	11	20	22	22	2689 ms	76	1016.14KB
Prof pac.(odesl. pozn.)	11	22	23	22	3256 ms	80	1.02MB

Tabulka 4.1: Analýza obsahu webových stránek

V tabulce vidíte načtení dvou stránek monitorace a profil pacienta.
Na těchto stránkách jsem provedl akci, což vidíte pod konkrétní stránkou. Provedením akce je myšleno spuštění PHP kódu, na jehož výstupu bude odpověď serveru s obsahem požadované stránky. U stránky monitorace vidíme dobu načtení stránky kratší než v případě vyhledávání uživatele pomocí vyhledávacího pole na stejné stránce. Při vyhledávání uživatele na stránce monitorace jsme vytvořili požadavek na specifická data v databázi systému. Tento požadavek zvýšil dobu, která je potřebná k načtení této stránky. Stejnou reakci jsme zjistili u načtení další stránky profilu pacienta, kde jsme pro porovnání provedli odeslání poznámky napsané lékařem. U těchto čtyř příkladů vidíme, že splňují požadavek zadavatele na počet načtených stránek za den.
Námi pronajatý server má hardwarové řešení postaveno na značkovém serveru Supermicro na nejmodernější platformě x9. Toto řešení využívá čtyř jádrový procesor Intel Xeon E5-2620 v taktu 2.00GHz a 4GB operační paměti.[1]

Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	4
Thread(s) per core:	4
Core(s) per socket:	1
Vendor ID:	GenuineIntel
CPU MHz:	2000.058
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	15360K

Tabulka 4.2: Výpis programu lscpu na testovaném serveru

Kapitola 5

Porovnání hardwarových komponentů pro levnou a značkovou variantu webového serveru

Server je obecně počítač připojený nepřetržitě do sítě s přístupem k internetu. Jeho nekončící provoz je odpovědný za vyřizování příchozích http požadavků od klientů. Jeho odpovědí na požadavek je odeslání cíle specifikovaného URL, což většinou představuje webovou stránku, obrázek, dokument apod. Všechny tyto procesy na serveru vyřizuje webový server. Nejčastějším webovým serverem je Apache http server.[2]
Pro upřesnění je potřeba definovat výše uvedené termíny levný a značkový webový server. Pod termínem levný webový server si představujeme něco mezi klasickým kancelářským počítačem umístěným v rohu zaprášeného „kumbálu“ a mezi dnes již silně rozšířenými domácími NAS servery, které většinou slouží k zálohování fotek z dovolených, ukládání filmů atd. Tato skupina webových serverů sice nepřetržitě poskytuje své služby koncovým klientům, ale jejich dostupnost je velice omezená, stejně jako jejich propustnost. Je to dáno především jejich výkonem a prostředím, ve kterém jsou umístěny. Jejich výkon je pro nabízení komerčních služeb nízký a vyžaduje velké investice do pořízení dostačujícího hardware. Pro komerční účely představují značné riziko při poskytování služeb zákazníkům a nelze je použít pro naše účely nabízení služeb webové aplikace zadavatele.
Značkovou variantou je zde myšlena skupina serverů, která je navržena přesně za účelem provozu webových služeb na internetu. Jejich výkon je stabilní a dokáže obsloužit stovky, tisíce a dnes již opravdu velké množství uživatelů za vteřinu. Jde především o skupinu serverů umístěných v serverové skříni, tzv. „rack“. Zde jsou servery umístěny tak, aby nezabíraly mnoho místa, ale zá-

roveň, aby jejich správa byla velmi praktická. Lze zde totiž velmi efektivně udržovat nízké provozní teploty a tím prodloužit životnost serverů a zároveň je zde téměř bezúdržbové prostředí s nízkým procentem prašnosti. Samozřejmě platí, čím vyšší kvalita serveru, tím vyšší cena. Je tedy na zvážení, zda je potřeba investovat do vlastního hardwaru a celkové správy webového serveru anebo využít služby pronájmu webových serverů. V dnešní době je na tomto trhu mnoho společností, které nabízí pronájem webového serveru a jeho celkovou správu. V níže uvedeném porovnání je i spočteno, že využitím těchto služeb pronájmu lze na provozování webových služeb ušetřit. Navíc se můžeme bezstarostně věnovat práci na projektu.

Kapitola 6

Porovnání pronájmu a nákupu vlastního webového serveru

Pronájem serveru patří mezi stále oblíbenější způsob provozování webových aplikací. Cílem této kapitoly je porovnat celkové investiční a provozní náklady na provoz serveru pořízeného běžným nákupem a formou pronájmu serveru umístěného v datovém centru. Protože morální životnost serveru je 3 roky, jsou celkové náklady porovnány na tuto dobu.[1] Pronájem serverů a jejich umístění v profesionálním datovém centru představuje v dnešní době nejmodernější a nejefektivnější způsob serverových řešení z pohledu kvality poskytovaných služeb i výše celkových nákladů. Hlavním důvodem, proč využít pronájem serveru místo jeho nákupu, je rozložení nákladů na pořízení a provoz serveru do pravidelných měsíčních splátek bez nutnosti jakýchkoliv počátečních investičních nákladů.

Srovnání nákladů na nákup vlastního serveru oproti pronájmu serveru

Pro srovnání nákladů jsem vybral níže uvedenou konfiguraci serveru, viz tabulka 6.1. Tuto konfiguraci jsem zvolil v nabídce pronájmu webového serveru. [2] na tento server jsem si nechal vystavit konečnou cenu za pronájem, viz tabulka 6.2. Pro nákup vlastního hardware jsem našel odpovídající server, jehož specifikace odpovídá naší konfiguraci. [3] Pro zjištění nákladů na provoz vlastního serveru jsem využil nabídky tzn. server housing. Jde o provozování vlastního serveru v prostředí datového centra. U jednoho z poskytovatelů této

[1] http://www.g2server.cz/blog/finance/srovnani-nakladu-nakup-vs-pronajem/

[2] http://www.web4u.cz/cs/serverhosting/dedikovany-server/model/supermicro-x10slm-f

[3] http://www.czc.cz/dell-poweredge-r220-e3-1220v3-8g-2x1tb-h310-1u/153628/produkt

služby jsem vytvořil objednávku na server housing, ze které jsem získal ceny na provoz vlastního serveru, viz 6.2.

Výchozí konfigurace serveru

Název serveru	Intel Xeon E3 4core
Procesor	Xeon E3-1220V3 3,1GHz
Paměť	8 GB DDR3 RAM
Pevné disky	2x 500GB SATA
Software	OS Linux

Tabulka 6.1: Konfigurace serveru

Náklady na nákup serveru a provoz po dobu 36 měsíců

	Nákup	**Pronájem**
Investiční náklady	26151 Kč	0 Kč
Cena serveru	26151 Kč	0 Kč
Provozní náklady	88200 Kč	106920 Kč
Náklady na energie, provoz, chlazení	88200 Kč	0 Kč
Pronájem serveru	0 Kč	106920 Kč
Náklady celkem	**124351 Kč**	**106920 Kč**

Tabulka 6.2: Srovnání nákladů na pronájem a nákup serveru

Pronájem serveru je navíc spojen s dalšími výhodami

- Nulové počáteční pořizovací náklady - služby jsou hrazeny měsíčními splátkami.
- HW servis do 60 min dle zvoleného tarifu (standard 4 hodiny)
- Flexibilita pronájmu - jsme schopni flexibilně reagovat na požadavky našich klientů a zvyšovat výkon pronajatých serverů dle jejich potřeb.
- Připojení do páteřní sítě - v základu 100 Mbps (symetricky), možnost rozšíření až na 1 Gbps.
- V ceně pronájmu jsou zahrnuty i všechny náklady spojené s provozem serveru (energie, pravidelná hardwarová údržba) a konektivita k internetu.

- Bezpečnost - díky maximální péči o bezpečnost serverů ze strany provozovatele data centra je riziko krádeže serveru (a tedy i všech dat), vyhoření serveru, povodně či mechanického poškození sníženo na minimum.
- 99,95% dostupnost garantovaná SLA certifikátem.

Nevýhody nákupu a provozu vlastního serveru

- Vadné komponenty serveru navyšují konečnou cenu.
- Nejistota dostupnosti našich služeb.
- Servisní zásah technika data centra navyšuje celkové náklady.
- Můžeme se stát cílem útoků hackerů, kteří nám budou přetěžovat server.
- Je potřeba mít znalost správy webového serveru.

Kapitola 7

Nainstalování a konfigurace vybrané varianty webového serveru

Na základě provedeného srovnání pronájmu a nákupu vlastního webového serveru jsme se rozhodli pro pronájem webového serveru u společnosti Server4u, na kterém budeme provozovat webovou službu zadavatele. Jedna z výhod pronájmu webového serveru je neustálý dohled a servis ze strany pronajímatele a také přednastavení potřebných služeb pro zprovoznění webové aplikace. Na serveru je již nastavena sada open-source softwaru pro implementaci dynamických webových stránek. To zahrnuje tyto technologie:

- Linux – operační systém
- Apache Server version: Apache/2.2.22 (Debian)– webový server
- MySQL Server version: 5.5.35-0+wheezy1 (Debian) – databázový server
- CGI, neboli PHP, Perl, Python – skriptovací jazyky

Samotnou instalaci webového serveru jsme díky předinstalovaným službám od poskytovatele nemuseli provádět. Návodů na to, jak nainstalovat webový server, najdeme na internetu spoustu a myslím si, že tento popis by zde byl pouze okrajový. My se zaměříme na úpravu konfigurace webových služeb a to webového serveru Apache a databázového serveru MySQL pro provoz webové aplikace.

7.1 Konfigurace Apache z hlediska výkonu

Po instalaci webového serveru Apache je výrobcem přednastaveno standardní nastavení pro běžné využití serveru. Mým úkolem však bylo zaměřit se na optimální nastavení pro výkonové zatížení podle požadavků zadavatele, tedy

stovky uživatelů přistupující denně na webovou aplikaci. [3] Pojďme si uvést hodnoty, na které je potřeba se zaměřit.

MaxRequestsPerChild

[4]Stanovuje limit na počet požadavků, které může jeden proces obsloužit. Jakmile proces dosáhne limitu, končí. Špatným nastavením této hodnoty můžeme způsobit velké úniky zdrojů v podobě paměti a výpočetního času, což je nežádoucí. Toto riziko představují především nízké hodnoty v řádu stovek procesů, kde dochází k velmi rychlému vytváření a ukončování potomků. Tím ubíráme procesorový čas k řešení procesů a dochází tak k prodlení. Pro nekonečný cyklus potomka se nastavuje hodnota 0. Standardně je přednastavena hodnota 10000, která je téměř ve všech případech dostačující. Pro práci pouze se statickým obsahem webu můžeme tuto hodnotu ještě navýšit. Jelikož věříme, že webová aplikace zadavatele je výkonově odladěná a nebude docházet k paměťovým únikům, nastavili jsme hodnotu 0.

StartServers, MinSpareServers a MaxSpareServers

StartServers udává počet vytvořených procesů při startu serveru. Tato hodnota je velmi podobná MinSpareServers, která udává nutné minimum spuštěných, ale nečinných procesů. MaxSpareServers je horní mez počtu nečinných procesů. Tím se proces stává, pokud neobsluhuje žádný proces. Když je nečinných procesů více než stanovený limit, rodič tyto procesy ukončí. Přednastavené hodnoty jsou pro zvolený apache modul odlišné. My využíváme modul prefork a máme nastaveno StartServers na hodnotu 5, MinSpareServers 5 a MaxSpareServers 150.

ServerLimit a MaxClients

Hranice pro tyto hodnoty je závislá na velikosti volné paměti a výpočetního času procesoru. Hodnota MaxClients určuje maximální počet připojených uživatelů. Standardně přednastavená hodnota MaxClients pro modul prefork je 256. Pro naše testování jsme hodnotu snížili na 150, jelikož budeme vytvářet maximálně 150 uživatelů přistupujících na server. Větší počet uživatelů již bude testovat nestandardní reakce testovaného systému. Hodnotu SeverLimit jsme upravili na stejnou hodnotu 150, protože pro modul prefork plní tyto dvě hodnoty stejný význam.

Zvolení MPM

Apache pracuje pod mnoha operačními systémy, které vyžadují různý přístup ke zpracování požadavků. Proto byl navržen systém modulů pro konkrétní operační systém, na kterém je Apache nainstalován. Pro unixové verze se nabízí několik modulů: prefork, worker, leader, perchild. Prefork je nejpoužívanější

přístup k vyřizování požadavků. Jeden hlavní proces je nadřízený ostatním. Prefork nepoužívá vlákna, ale procesy. Těch běží více a každý vyřídí jeden požadavek. Podle počtu příchozích požadavků se zvyšuje nebo snižuje počet procesů. Výhoda tohoto řešení je, že chyba jednoho požadavku ukončí pouze jeden proces a nezatěžuje tak ostatní procesy. Modul Worker má také více procesů, ale zde má každý proces více vláken. Je výkonnější než Prefork a je vhodnější pro náročné aplikace. Musíme ale počítat s tím, že chyba jednoho procesu může vést ke zpomalení a pádu více procesů. K tomuto modulu se vážou specifická nastavení hodnot, např. ThreadsPerChild, které vyjadřuje počet vláken na jeden proces. Modul perchild je podobný workeru, ale pracuje s proměnlivým počtem vláken. Modul event je také podobný workeru, ale předává požadavky ve stavu KEEP ALIVE určenému vláknu, pracovní vlákno se tím uvolňuje pro další zpracování.

Navýšení paměti Cache

Použití paměti cache je využíváno v mnoha Apache modulech, které zde udržují záznam o nejčastěji využívaných datech. Používáním paměti cache je velmi užitečné a navýšením této paměti rozhodně přispějete ke zvýšení výkonu. Pokud zjistíte, že část operační paměti je nevyužitá, navyšte paměť cache.

Deaktivace nepoužívaných Apache modulů

Pokud nejsou všechny aktivní moduly používány, je nutné je deaktivovat, neboť i neaktivní modul využívá paměť. Standardně je aktivováno zhruba dvacet modulů.

7.2 Konfigurace MySQL z hlediska výkonu

MySQL je jeden z nejoblíbenějších SQL databázových serverů. Server se snadno aktualizuje i nastavuje a podporuje celou řadu skriptů a programovacích jazyků. Oblast správy databázového serveru MySQL vyžaduje mít přehled o novinkách v aktualizacích MySQL[5]. Na MySQL server jsme použili InnoDB formát úložiště dat, které je od verze MySQL 5.5 standardně přednastaveno a stalo se tak nejčastěji využívaným formátem. InnoDB podporuje rozšířené zamykání záznamů a předně transakční zpracování požadavků. To jej předurčuje k využití v aplikacích, které jsou závislé na naprosto spolehlivé práci s daty, jako jsou například bankovní systémy. Uvedu zde nástroje, které jsme využili pro stanovení optimálních hodnot konfigurace a několik základních hodnot, které by měly být vždy upraveny podle vaší hardwarové konfigurace serveru.

7.2.1 Mysqlreport

Mysqlreport je nástroj pro kontrolu důležitých hodnot MySQL serveru. Jeho výstup je uživatelsky přívětivý a snadno pochopitelný. Obsahuje přehled všech nastavených hodnot a jak jsou ve skutečnosti využívány. Jeho cílem je usnadnit uživatelů práci s analýzou využití MySQL serveru, která by jinak musela být zpracována ručně nebo pomocí jiného nástroje. Výstup tohoto programu `mysqlreport.tex` je přiložen v příloze této práce v adresáři sources.

7.2.2 Tuning primer

Skript Tuning-primer.sh jsem použil pro kontrolu a případné vylepšení nastavení databázového serveru MySQL. Jedná se o nástroj, který si načte stávající nastavení a statistiky MySQL serveru pomocí `SHOW STATUS LIKE` apod. Na základě algoritmu výrobce skriptu pak doporučí vhodné nastavení databázového serveru. Je kompatibilní od verze MySQL 3.23 až po současnou poslední verzi 5.6. Nástroj dokáže navrhnout doporučení například pro následující hodnoty:

- Slow Query Log
- Max Connections
- Worker Threads
- Key Buffer
- Query Cache
- Sort Buffer
- Joins
- Temp Tables
- Table (Open & Definition) Cache
- Table Locking
- Table Scans (read_buffer)
- Innodb Status

7.2.3 Percona Tools for MySQL

Jako další zdroj pro studium a porovnání nastavení MySQL serveru se nabízí nástroj společnosti Percona. Na jejich stránce [4] je potřeba projít sedm kroků,

[4] Percona Tools for MySQL - PERCONA CONFIGURATION WIZARD FOR MYSQL

kde stačí odpovědět na pár otázek, jaká bude role serveru, jaké jsou spuštěny další služby na serveru kromě MySQL databáze, či zda se jedná o replikační server. Výstupem tohoto procesu je obsah konfiguračního souboru pro nastavení databázového serveru podle společnosti Percona.

7.2.4 `innodb_buffer_pool_size`

Je nejdůležitější hodnota, na kterou je se potřeba zaměřit při jakékoliv instalaci InnoDB. Tato hodnota představuje cache paměť, kde se indexují data. Čím vyšší tato hodnota bude, tím menší je pravděpodobnost, že budete muset přistupovat na disk s aktuálním požadavkem a tím snížíte čas operace tohoto procesu. Hodnota by měla být nastavena podle počtu služeb běžících na serveru. Pokud se jedná pouze o databázový stroj, může zde být nastaveno až 80% paměti RAM. My ovšem máme nainstalovány další služby jako Apache a můžeme tak nastavit hodnotu z rozsahu 40 – 60%. Určit konkrétní hodnotu není hned možné, vychází z požadavků běžících aplikací. Na našem testovaném serveru je nastaveno pouze 128MB. Dle doporučení nástroje Tuning primer je bezpečné nastavit hodnotu optimálně na 2/3 z celkové paměti systému. V našem případě máme k dispozici 3,85 GB a dle doporučení bychom měli nastavit hodnotu v rozmezí od 1,5-2,5GB.

7.2.5 `innodb_log_file_size`

Určuje v megabytech velikost jednotlivých souborů protokolu v protokolové skupině. Standardně je pro verzi 5.5 nastaveno 48MB, ale je možno nastavit velikost hodnoty až na 1/N, kde N zastupuje hodnotu `innodb_log_files_in_group` hodnoty `innodb_buffer_pool_size`. Nastavením vysoké hodnoty šetříme diskové operace, avšak zpomalujeme proces obnovení po selhání. Celková velikost souborů protokolů by neměla přesáhnout 4GB na 32 bitových systémech. Na našem testovaném serveru byla ponechána standardně přednastavená hodnota 48MB

7.2.6 `max_connections`

Uvádí nejvyšší počet povolených připojení. Mimo normální stav může nastat stav vyčerpané paměti RAM, pokud jsme zvolili vysokou hodnotu, anebo nám server přestane přijímat požadavky, protože jsme nastavili příliš nízkou hodnotu. Ideální hodnota je tedy někde mezi průměrným počtem požadavků a velikosti paměti RAM. My jsme přednastavili hodnotu 151. Po vyhodnocení nastavení nástrojem Tuning primer bylo zjištěno, že využíváme pouze 10% a měli bychom tedy nastavenou hodnotu `max_connections` snížit, abychom předešli neočekávanému vyčerpání paměti.

7.2.7 `innodb_flush_log_at_trx_commit`

Nastavuje postup po provedení příkazu commit. Hodnota 1, která je přednastavena i na našem testovaném serveru, znamená nejbezpečnější přístup. Jakmile totiž příjme příkaz commit, okamžitě synchronizuje buffer protokolu do souboru protokolu na disku. Nastavením hodnoty 0 nebo 2 představujeme lehce zpožděný zápis na disk. Nula znamená synchronizaci jednou za vteřinu. Nastavení 2 znamená, že commit si vynutí zápis bufferu protokolu do souboru protokolu, ale soubor protokolu není zapsán na disk dříve, než uplyne sekunda.

7.2.8 `innodb_file_per_table`

Pokud je nastaveno na ON, pak říkáme InnoDB, aby ukládalo data a indexy do souborů s koncovkou `.ibd` pro každou tabulku. Výhodou tohoto nastavení je to, že můžeme obnovit data po pádu, úpravě nebo smazání tabulek. Avšak pro databáze s tisíci tabulkami je toto nastavení velmi náročné na výkon serveru a proto se pro vyšší zátěž nedoporučuje. My jsme ukládání dat a indexů pro zátěžové testování nechali vypnuté.

7.2.9 `query_cache_size`

Při nenulovém nastavení udává velikost paměti cache. Doporučené hodnoty pro nastavení jsou v řádech desítek MB, neboť stovky MB bývají již kontraproduktivní z důvodů nutnosti zneplatňování velkého množství záznamů při změnách. Konkrétní hodnota pro jednotlivé servery se může lišit a optimum je třeba stanovit měřením. Já jsem změnil standardně nastavenou hodnotu 0 na 16M. [5]

[5] http://www.zdrojak.cz/clanky/zvysujeme-vykon-mysql-zmenou-konfigurace/

Kapitola 8

Typy zátěže webového serveru

S odkazem na [6] definujeme základní typy testování výkonu

- Performance test - Výkonnostní test
- Load test – Zátěžový test
- Endurance test – Dlouhodobý test
- Stress test – Test hraniční zátěže

Výkonnostní testování zatížení systému definovanou zátěží pro změření jeho chování se používá pro změření reakcí očekávané zátěže nebo porovnání vlastností systému po provedené změně.
Zátěžové testování je prováděno za účelem zjištění, jak se chová testovaný systém pod narůstajícím zatížením. Zatížením se myslí zvyšující se počet přistupujících uživatelů, či počet paralelních požadavků v testovacím scénáři. Zátěžové testování pomáhá určit hranici maximální provozní kapacity systému. To je ovšem velmi obtížné určit předem a hranice se tak stanovuje až po několika provedených testech se zvyšující se zátěží. V průběhu zvyšování zátěže nám častěji přibývají problémy s úzkými místy v systému, která by se projevila až při ojedinělých situacích v provozu systému, kdy je to nežádoucí.
Dlouhodobé testování, jak už je z názvu patrné, vychází hlavně z dlouhodobého testování. Úroveň zátěže nemusí být tak vysoká jako u zátěžového testování. Měřeno je spíše rozložení stejné zátěže do různých časových intervalů. Provádí se často za účelem identifikace vyčerpaných, popř. unikajících zdrojů paměti.
Testování hraniční zátěže je typ testu zatížení s cílem ověření stability systému během nárazového zatížení. Nárazové zatížení by mělo mít nastavené hodnoty za maximální hranicí provozní kapacity systému. Cílem testování je zjistit zda, při nárazové zátěži systém přestane úplně fungovat nebo bude schopen zvládnout dramatický nárůst zatížení.

Kapitola 9

Použité nástroje k zátěžovému testování

Při zátěžovém testování a jeho samotné přípravě jsme použili pouze licenčně svobodné nástroje. Podařilo se mi najít na webové stránce [6] spoustu programů zaměřených na zátěžové testování. Z tohoto důvodu jsme nemuseli pro testování investovat do nákupu licencí komerčních nástrojů. Ze všech nabízených nástrojů jsem musel vybrat ten, který by nejlépe vyhovoval mé potřebě. Nástroj by měl být na platformě nezávislý, abychom jej mohli spouštět nejen na OS Linux, ale i na OS Windows. Velmi důležité je, zda výrobce nástroje aktivně pracuje na jeho údržbě a aktualizacích. Poskytuje tak jistotu, že při možném zjištění nefunkčnosti programu můžeme nahlásit tuto vadu výrobci, který poskytne nějaké řešení. Výhodou je určitě možnost připojit se ke komunitě, která by mi mohla pomoci s implementací mého řešení.

JMeter

Jedná se o nástroj pro měření výkonnosti a pro vytváření zátěže webových aplikací.[7] Lze s ním měřit i výkon SQL databází. Dále se dá použít pro testování JDBC, FTP, LDAP, webservices, JMS, HTTP a generických TCP. Nespornou výhodou JMeteru je, že open-source.[8] JMeter Vám umožní vytvořit specifický test pro vaši aplikaci, který simuluje reálnou zátěž během provozu. Můžete si nastavit, kolik klientů bude posílat požadavky, na jaké objekty a v jakých časových intervalech. JMeter je napsán v jazyce JAVA a disponuje grafickým rozhraním. Jde o velmi populární nástroj pro vytváření zátěžových testů webových aplikací.[7] Jeho dovednosti lze snadno rozšířit pomocí pluginů a komponent, které se také vytvářejí v tomto programovacím jazyce. Díky tomu je možné sestavit i velice složité scénáře, které pracují s mnoha daty a vy-

[6] http://www.opensourcetesting.org/performance.php
[7] http://testovanisoftwaru.cz/automatizovane-testovani/jmeter/

žadují specifické konverze nebo korelace pro vytváření funkčních požadavků. Řada funkcí JMeteru je připravena tak, aby bylo možné je snadno rozšířit nebo zcela nahradit vlastní implementací. Umožňuje spustit test přímo v GUI rozhraní nebo z příkazové řádky. Vykreslování grafů nebo tabulek v GUI často bere mnoho výpočetního výkonu, který pak chybí při vytváření zátěže, a tak může dojít ke zkreslení výsledků. Spuštění testu v konzoli je z tohoto pohledu výhodnější, nabízí statistiky jednotlivých volání, ale postrádá ostatní informace.[8] Tento nástroj jsem si vybral hlavně proto, že platformě nezávislý, jeho použití je pro nekomerční účely zdarma, aktualizovaná podpora a dokumentace, je vhodný pro jednoduché scénáře a řádově stovky virtuálních uživatelů.

Dstat

Je univerzální náhrada za množinu monitorovacích nástrojů jakými jsou: vmstat - virtuální paměť, iostat - vstupně výstupní operace, netstat - síťová komunikace a ifstat - rozhraní. Dstat překonává některé z jejich omezení a přidává některé další funkce, více čítačů a flexibilitu. Dstat je užitečný pro sledování průběhu zátěžového testování. Dstat umožňuje zobrazit všechny systémové prostředky v reálném čase, můžete např. porovnat využití disků v kombinaci s přerušeními od řadiče IDE nebo porovnat čísla šíři pásma sítě přímo s propustností disku (ve stejném intervalu).[9]

Top

Je jeden z nástrojů UNIX-like operačních systému pro monitorování právě běžících procesů a podprocesů. Na jeho výstupu přehledně vidíme seznam spuštěných procesů a statistiky využití systémových prostředků. Oproti nástroji Ps, který také monitoruje procesy, obsahuje Top pro nás lepší výstup. Abychom mohli uložit výstup tohoto nástroje, musel jsem vytvořit skript, který spouští program Top s parametry `n` a `b`. Parametr `n` nastavuje počet iterací než se program ukončí. Já jsem nastavit tento parametr na 1, abych dostal jeden výstup v podobě seznamu procesů. Parametr `b` nám dovoluje výstup programu předat jinému programu nebo uložit do souboru jehož název jsem nastavil jako argument programu.

Rally agile

V dnešní době existuje mnoho různých agilních nástrojů. Všechny vycházejí ze stejných principů: rychlý vývoj, kolektivní práce, spolupráce v týmu, komunikace se zákazníkem a především schopnost přizpůsobit se během celého

[8] http://www.etnetera.cz/773-tech_life/tech_life_140213_aplikace_pro_zatezove_testovani.html

[9] http://dag.wiee.rs/home-made/dstat/

životního cyklu projektu. Agilní metodiky většinou rozdělují jednotlivé požadavky na menší části s minimálním plánováním a dlouhodobé plánování přímo nezahrnují. Rally se v tomto projektu využívá především na definování jednotlivých testů, záznam výsledků provedeného měření, přiložení potřebných souborů, především vizualizovaných grafů v obrázku a samozřejmě zhodnocení naměřených výsledků. Tento nástroj jsem používal, protože byl ve společnosti zadavatele již zaveden a já jsem se jako člen týmu musel přizpůsobit.

Generatedata

Někdy je potřeba vytvořit velké množství ukázkových údajů pro věci, jako je testování softwaru, databáze jmen zákazníků, tvorba realisticky vypadajících maket, a tak dále. U textu máme Lorem Ipsum, ale pro všechno ostatní, co musí obsahovat lidská data, jako jsou jména, adresy, jména měst atd., je obzvlášť těžké vytvořit, protože budete potřebovat polo realisticky vypadající sadu dat. Tento nástroj byl napsán proto, aby tento problém vyřešil. Poskytuje rychlý a jednoduchý způsob, jak vám umožní vytvářet velké objemy vlastních dat v libovolném formátu, jaký budete potřebovat. [10] Nám umožnil rychle vytvořit velké množství vlastních dat pro testování softwaru, databáze a mnoho dalších. Podle naší potřeby je možno si přizpůsobit sadu formulářů, podle kterých si vlastní data vytvoříme. My tento nástroj využijeme pro vytvoření testovacích dat k naplnění naší databáze, jedná se o uživatelské účty, objednávky, monitorování apod.

Vlastní skripty

9.0.10 Skripty na vytvoření souboru pro import do databáze

Pomocí výše uvedeného nástroje Generatedata jsme schopni vytvořit testovací data pro naplnění jednotlivých tabulek databáze. Tabulky jsou v databázi provázány mnoha vztahy, nejčastěji typu 1:N, a je tedy potřeba testovací data vkládat tak, abychom tyto vazby dodrželi. Jelikož se jedná o velké množství vkládaných dat, vznikly níže uvedené skripty. Jejich hlavním úkolem je vytvořit soubor obsahující správně seřazená testovací data. Tento soubor pak nahráváme do databáze. Skripty jsou napsané v jazyce bash. Každý skript obsahuje na začátku ukázku použití.

9.0.11 Vytvoření doktorských účtů

Pomocí tohoto skriptu vytváříme uživatelské účty role `lekar_diab`. Vstupem pro tento skript je soubor s daty uživatelských účtů role `lekar_diab`. Jednotlivým účtům je potřeba nastavit rozsah působnosti a přidat další hodnoty

[10] http://benkeen.github.io/generatedata/

do tabulky `src`. Tento proces za nás nyní provádí skript. Výstupem je soubor obsahující sled SQL příkazů, které můžeme nahrát do databáze.

9.0.12 Vytvoření pacientských účtů

Vytvoření uživatelských účtů role `pacient_diab` vyžaduje jiný postup než v případě role `lekar_diab`, proto vznikl tento skript. Nově přidanému záznamu do uživatelských účtů je v případě role `pacient_diab` potřeba přidat vztahy do dalších tabulek:

- Pacient
- Přistroj
- Objednávka
- Monitorace
- Časy přibližného měření glykémie
- Meze hodnot glykémie

Tyto tabulky proto musejí být na vstupu tohoto skriptu. Logika skriptu je taková, že ze vstupních souborů načítám postupně řádek po řádku. Jeden řádek z každého vstupního souboru pak představuje proces vytvoření uživatelského účtu role `pacient_diab`.

9.0.13 Vytvoření naměřených hodnot pacientů

Cílem tohoto skriptu je nastavit uživatelskému účtu role `pacient_diab` naměřená data glykémie. Vstupní soubor obsahuje náhodně vytvořené hodnoty glykémie v určitý den a čas. Počet řádků ve vstupním souboru odpovídá počtu přiřazených hodnot jednomu účtu. Skript nastavuje naměřená data ze vstupního souboru uživatelskému účtu. Toto dělá pro námi nastavený počet pacientů.

9.0.14 Skripty pro vytvoření testovacích scénářů

Pomocí níže uvedených skriptů jsme schopni vytvořit testovací scénáře pro definovaný počet uživatelů konkrétního testu. Pro každý námi definovaný kritický proces existuje skript na vytvoření testovacích scénářů. Pomocí něj vytváříme scénáře pro konkrétní test. Proces vytvoření skriptů byl následující. Každý kritický proces má definované kroky. Ty představují URL adresy s požadavky na konkrétní stránky webové aplikace zadavatele. Tyto požadavky se implementují do programu JMeter pouze pro jednoho uživatele. Tento návrh testovacího scénáře je spuštěn. Pokud všechny požadavky proběhly úspěšně a

požadované výstupy z provedeného testování byly správně zaznamenány, použijeme tento návrh jako šablonu do skriptu konkrétního testu, pomocí kterého budeme vytvářet stejné testovací skripty pro konkrétní počet uživatelů.

TS 1: Simulace přístupů pouze pro lékaře specialistu

Úkolem tohoto kritického procesu bylo nasimulovat práci specialistů (lékařů) ve webové aplikaci. Předpokládá se, že specialista vykonává práci v aplikaci jednou denně po co nejkratší čas nutný ke kontrole svých pacientů. Postup práce specialisty začíná přihlášením se do webové aplikace. Po přihlášení přechází na stránku monitorací, kde je seznam pacientů, kteří jsou v daném období monitorováni. Ze seznamu pacientů lékař kontroluje ideálně všechny pacienty, kteří vykazují kritické hodnoty. U těchto pacientů se podívá na přehled naměřených hodnot a přečte si komentáře ke kritickým hodnotám. Na závěr kontroly pacienta napíše komentář k jeho stavu. Po kontrole svých pacientů se odhlašuje z aplikace.

TS 2: Simulace přístupu pouze pacientů

Pacient využívá k měření cukru v krvi glukometr, který odesílá naměřené hodnoty do webové aplikace zadavatele a to pomocí WiFi připojení nebo přes Bluetooth do aplikace v mobilním telefonu. Typickou činností pacienta ve webové aplikaci je okomentování svých naměřených hodnot. To se předpokládá pouze u naměřených hodnot, které jsou příliš nízké nebo příliš vysoké. Takové hodnoty se nazývají hypoglykémie nebo hyperglykémie. Dále si projde komentáře vytvořené jeho lékařem a může se podívat na grafy obsahující koncentraci naměřených hodnot v kategoriích hypoglykémie, optimální hladina cukru v krvi a hyperglykémie.

TS 3: Simulace odesílání naměřených hodnot do webové aplikace

Tento kritický proces simuluje odesílání naměřených hodnot glykémie pacienty do aplikace. Stanovili jsme, že pacienti se každý den měří pětkrát a své naměřené hodnoty odesílají až večer po posledním měření, protože je to pro ně pohodlnější.

TS 4: Simulace současného přístupu specialistů, pacientů a odesílání naměřených hodnot

Tento kritický proces kombinuje předchozí tři varianty. Toto testování bylo velice náročné na výkonnost kontroloru, na kterém jsem spouštěl testovací scénáře. Z toho důvodu jsem musel navrhnout testovací scénáře s nižším počtem uživatelů, než jsem si představoval. Délka těchto testů byla ze všech definovaných testů nejdelší. Jednalo se o necelou půlhodinu, při které jsem monitoroval stav testovaného systému.

Kapitola 10

Návrh testovacích scénářů kritických procesů

10.1 Analýza zátěžového testování

Základním kamenem každého projektu je vytvoření analýzy [9]. V našem případě se budeme zabývat analýzou zátěžového testování. Tato část určuje základní atributy výkonnostního měření. Celková analýza zátěžového testování podléhá schválení příslušným řídícím orgánem. Tuto pozici zastupuje Projektový manažer.
Představení webové aplikace, na které se bude zátěžové testování provádět, tedy „Co se bude měřit?“, jsem již provedl v kapitole 2. Tato aplikace běží na nějakém prostředí, takže blíže představíme testovací prostředí. Zamyslíme se nad možnými omezeními tohoto testování a uvedeme možná rizika. Jelikož se jedná o menší projekt testování zátěže, realizačním týmem se stává pouze z mé osoby s dohledem odpovědného programátora, který zná architekturu a vlastnosti webové aplikace.

Přehled analýzy zátěžového testování

Tuto analýzu pro zátěžové testování rozdělujeme do těchto tří částí:

- upřesnění zadání zátěžového testování
- analýza testovaného systému
- návrh realizace zátěžového testování

Jednotlivé aktivity analýzy pro zátěžové testování na sebe navazují a jsou většinou na sobě závislé. Znamená to, že bez znalostí získaných v předchozí aktivitě nemůže analytik zátěžového testování efektivně pokračovat v analytických pracích následujících aktivit. Současně aktivity analýzy odpovídají příslušným kapitolám, které jsou do analýzy zpracovávány.

Upřesnění zadání zátěžového testování

Zadání pro zátěžové testování vychází z požadavků zadavatele práce. Ten chce na trh uvést svou novou webovou aplikaci.
Cílem zátěžového testování je zjistit hranice možností konkrétního webového serveru, na kterém nyní webová služba běží a zjistit, pro kolik uživatelů je schopný zajistit stabilní chod této služby. Vytvořit ukázková data registrovaných uživatelů a jejich naměřených dat. Připravit adresářovou strukturu a do ní uložit tato data tak, aby byla jasně rozpoznatelná a připravená okamžitě k testování. ve zvoleném testovacím nástroji vytvořit testovací scénáře, které simulují běžnou činnost rolí typu pacient, lékař specialista. Vybrat vhodné monitorovací nástroje serveru. Provést zátěžové testování a zpracovat naměřená data. Na základě těchto výstupů zhodnotit stávající hardwarovou a softwarovou konfiguraci a navrhnout doporučenou optimalizaci.

Specifikace realizačního týmu

Návrh a přípravu zátěžového testování jsem prováděl pouze já a výstupy mé přípravy jsem konzultoval s odpovědným zástupcem vývojového oddělení, který navrhoval další změny nebo mé výstupy akceptoval. Konzultace se prováděla každý týden. Nově vzniklé úkoly k řešení byly zaváděny do agilního nástroje Rally. K těmto úkolům byly přiděleny konkrétní osoby jako řešitelé. Takto definované úkoly v agilním nástroji Rally nám pomohly efektivně rozložit práci mezi volné zdroje a hlavně dodržet harmonogram prací.

Specifikace testovacího prostředí

Služba zadavatele je již nainstalována na pronajatý server vybrané hostingové společnosti. Přístup ke konfiguraci tohoto serveru mají pouze zúčastněné osoby realizačního týmu. Služba je přístupná na internetu, aby ji bylo možno testovat v běžném prostředí a dosáhnout reálných hodnot. Využívání služby je zpřístupněno pouze uživatelům, kteří mají přístupové jméno a heslo. V tomto případě se bude jednat pouze o uměle vygenerované uživatelské účty a nebude se jednat o reálné osoby. Na webovém serveru poběží databáze MySQL s potřebnými daty k testování a monitorovací nástroje. Simulace uživatelů bude spuštěna na mnou poskytnutém laptopu HP 4330s. Tomuto zařízení se v prostředí zátěžových testů říká „Controler“ neboli kontrolor. Webovému serveru, databázovému serveru a obecně všem testovaným službám se jednotně říká „testovaný systém“.

Popis testovaného systému

Při našem testování jsme měli nainstalovanou verzi PHP 5.3.3, Apache 2.2.22, MySQL 5.5.35. Zároveň na serveru nebyly spuštěny žádné další náročné pro-

cesy, například aplikační žurnálovaní, webové služby, zálohování nebo provozní monitoring.

Popis kontroloru

Na tomto počítači jsme spouštěli testovací scénáře programem JMeter ve verzi 2.11. Spuštěn byl na operačním systému Elementary OS Luna, což je linuxová distribuce Ubuntu. Tento počítač využívá procesor Intel Core i5 s taktem 2,3 GHz, k dispozici zde je 4 GB paměti RAM. Z počátku jsem při spouštění méně náročnějších testů využíval program JMeter v grafickém rozhraní. Náročnější testy jsem pak spouštěl přes negrafické rozhraní v terminálovém režimu tak, abych mohl získat maximální výkon. V průběhu testování jsem zde nespouštěl žádné další procesy.

Kritické faktory

Jelikož se zátěžovým testováním nemám doposud žádné zkušenosti, vidím zde určité riziko, že můžu některé postupy vykonávat zdlouhavě z důvodu nedostatku informací, nebo že můžu opomenout kroky před testováním, což způsobí nutnost opakovat testování znovu. Také se obávám, že vytvoření scénáře pro test nebude vždy odpovídat přesnému zadání, protože webová aplikace má svou bezpečnostní politiku, kterou nebudu schopen splnit. Je tedy potom na dohodě, zda lze pro účely testu snížit tyto nároky nebo bude potřeba změnit scénář. Zátěžové testování bude prováděno jako distribuovaný test a zde vidím nejzávažnější kritický faktor. Kontrolorem, generátorem a monitorem se stává pouze jeden počítač, který bude zároveň generovat virtuální uživatele, spouštět skripty a monitorovat testovaný systém.

Specifikace testovacích dat

Přesná specifikace testovacích dat vychází z architektury databáze webové aplikace. Ta je navržena pro uchovávání uživatelských účtů a jejich specifikace rolí, objednávek monitorování nad množinou pacientů, přiřazení přístrojů k jednotlivé monitoraci, přiřazení kontrolních hodnot pro každého pacienta a spoustu dalších tabulek spojených s objednáváním služeb. Důležité je správně evidovat naměřená data z přístrojů od konkrétního pacienta. Tato tabulka bude v našem zátěžovém testování obsahovat nejvíce záznamů, jelikož jsou zde uloženy záznamy naměřených dat všech pacientů. Nad pacientovými záznamy má okamžitý přehled specialista lékař, ke kterému je pacient přiřazen. Z výše uvedeného popisu databáze (jedná se pouze o výčet tabulek databáze) je patrné, že vztahy mezi jednotlivými tabulkami a jejich záznamy jsou klíčové a je potřeba je dodržet. Jelikož jsem simuloval virtuální uživatele a jejich naměřená data, bylo potřeba vybrat vhodný nástroj ke generování takových dat. Následně jsem musel provést zpracování vygenerovaných dat tak, aby je bylo

možné úspěšně nahrát do databáze. Nepředpokládal jsem, že na toto zpracování je k dispozici nějaký nástroj a pro každý testovací scénář jsem musel vytvořit program, do kterého vložím potřebná data. Výstupem programu je soubor, který je možno úspěšně nahrát do databáze. Další podobný program pro zpracování vstupních dat je potřeba pro naměřená data pacientů. Všechny použité soubory ve vztahu k databázi mají koncovku `sql`. Po úspěšném nahrání všech potřebných dat k zátěžovému testování jsem provedl zálohování celé databáze. To výrazně zkrátilo čas práce při další potřebě použít tento typ zátěžového testu a logicky jsme tímto zvýšili efektivitu práce, kterou můžeme investovat jinde.

Specifikace měřených parametrů

Výstup zátěžového testování je tou nejdůležitější hodnotou, kterou máme k dispozici z provedeného testování. Na základě těchto naměřených hodnot jsme mohli stanovit závěr zátěžového testování, a proto bylo velmi důležité vybrat vhodný nástroj, který nám poskytne kvalitní výstup. S tímto výstupem pak provedeme další analýzy a navrhneme doporučení pro optimalizaci. Při každém zátěžovém testu byly měřeny jednotlivé prvky serveru. Základními výkonnostními ukazateli jsou: vytížení CPU, využití operační paměti, práce s disky nebo síťové operace.

10.2 Návrh realizace zátěžových testů

Tato kapitola řeší stanovení postupů, definice testovacích scénářů a organizační záležitosti zátěžového testování.

Definice cyklů zátěžového testování

Definujme si jednotlivé kroky, které povedou k dokončení zátěžového testování.[10] Začal jsem výběrem vhodných nástrojů pro zátěžové testování a monitorování testovaného systému. Jednotlivé nástroje jsem již dříve stručně popsal. Tyto nástroje nám poskytnou dostatečný výstup naměřených dat pro další zpracování, viz Specifikace měřených parametrů. Společně s projektovým manažerem jsme si definovali slovně testovací scénáře, které vycházejí z kritických procesů uživatelů ve webové aplikaci. Konkrétní chování jednotlivých uživatelů již dříve konzultoval projektový manažer se zástupci jednotlivých rolí. Testovací scénáře se tedy velmi blíží reálnému chování uživatelů. Definované testovací scénáře jsem uložil do agilního systému Rally, kde se sdružují také úkoly pro přípravu testovaného systému a definice konkrétních zátěžových testů. Tyto slovně definované testovací scénáře jsem přepsal do vybraného nástroje JMeter k zátěžovému testování. V souvislosti s připravováním testovacích scénářů jde ruku v ruce také příprava ukázkových dat, nad kterými jsem prováděl testy. Jedná se o již dříve zmíněná data v databázi. Bylo tedy

zapotřebí připravit databáze pro jednotlivé testy. Vycházel jsem z požadavků zadavatele, že simulovaných uživatelů budou stovky, bylo proto potřeba vybrat nástroj, který nám dokáže vytvořit velké množství ukázkových dat, která se budou nahrávat do databáze. Takto jsem si připravil databáze, nad kterými jsem spouštěl konkrétní testy. Průběh testování jsem monitoroval vybranými nástroji Dstat a Top, které mi poskytly potřebná data ke zhodnocení provedeného testování.

Definice testovacích scénářů

Jak jsem výše popsal, testovací scénáře vycházejí z kritických procesů webové aplikace. Jedná se o nejčastější činnosti uživatelů, které mají začátek a konec. Není v mých silách a ani není potřebné vytvořit sadu scénářů, kde si každý uživatel dělá, co chce. Takové testování nechceme. Níže uvedené návrhy scénářů vycházejí z konzultací, kterých se účastnili lide zastupující jednotlivé role: pacient a lékař specialista. Pro tyto role byly vytvořeny následující scénáře:

TS1: Chceme znát, kolik diabetologů souběžně může kontrolovat své pacienty

Definice činnosti lékaře
Lékař zadá adresu pro přihlášení do systému aplikace Diabetes. Vyplní pole uživatel a heslo a přihlásí se do systému. Je přesměrován na osobní stránku. Po této fázi lékař přechází k samotné práci na pacientech. Jeho práci vyjadřují tyto činnosti:

1. Lékař klikne na záložku Monitorace, odpověď serveru je odeslání obsahu stránky Monitorace.
2. Lékař vybere daného pacienta pro kontrolu naměřených hodnot kliknutím na políčko Hodnotit a lékaři se zobrazí profilová stránka vybraného pacienta.
3. Lékař si přečte poslední záznamy pacienta, které zde napsal on, či jeho kolegové, aby si připomněl stav pacienta.
4. Zkontroluje schéma měření pacienta, zda zapsal všechny hodnoty.
5. Prohlédne si souhrnný graf měření za 30 dní.
6. Klikne na tlačítko „7 dní“ pro zobrazení souhrnného grafu měření za sedm dní a prohlédne si graf měření.
7. Klikne na tlačítko „90 dní“ pro zobrazení souhrnného grafu měření za devadesát dní a prohlédne si graf měření.
8. Prohlédne si detailní graf měření za 90 dní.

9. Prohlédne si detailní graf měření za 30 dní.
10. Prohlédne si detailní graf měření za 7 dní.
11. Prohlédne si delta graf měření za 7 dní.
12. Prohlédne si delta graf měření za 30 dní.
13. Prohlédne si delta graf měření za 90 dní.
14. Pro detailnější přehled pacientova měření si lékař prohlédne „Statistické údaje o záznamech monitorace“.
15. Nyní již má dostatečný přehled o pacientově stavu a do elementu „Příběh“ napíše poznámku s hodnocením pacientova stavu a léčby.
16. Po napsání poznámky lékař klikne na tlačítko „vložit poznámku“, ta se zapíše do databáze a lékaři se načte znovu profil pacienta.
17. Lékař pokračuje krokem č. 1.

Tyto činnosti jsem převedl do podoby URL odkazů na konkrétní stránku. V níže uvedeném seznamu neuvádím doménu na webovou aplikaci zadavatele, která je samozřejmě součástí v reálném testování. Některé kroky definují činnost osoby na stejné stránce a nejedná se o přechod na novou stránku. Tato činnost je definována jako časová konstanta. Následuje seznam odkazů, který slouží jako šablona testovacího scénáře pro jednoho uživatele.

1. `/cs/monitorace/monitorace-glykemie`
2. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni-zaznamu`
3. `Kontrola schématu, časová konstanta 15s.`
4. `Prohlížení souhrného grafu, časový konstanta 15s.`
5. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/`
6. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/7_days/`
7. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/90_days/`
8. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/90_days/gr_time`

9. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/30_days/gr_time`
10. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/7_days/gr_time`
11. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/7_days/gr_delta`
12. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/30_days/gr_delta`
13. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta] /prohlizeni_mon/90_days/gr_delta`
14. `Prohlížení statistických údajů o~záznamech monitorace, časová konstanta 15s.`
15. `Psaní hodnocení, časová konstanta 60s.`
16. `Odeslání hodnocení /cs/monitorace/monitorace-glykemie /[pořadové číslo pacienta]/prohlizeni_mon/90_days/gr_delta`

Výše uvedený seznam odkazů a časových konstant jsem implementoval do programu JMeter.

TS2: Chceme znát, kolik pacientů může prohlížet svá data souběžně

Pacient vstupuje do webové aplikace zhruba jednou za dva dny, aby komunikoval s lékařem a uvedl okolnosti měření extrémních hodnot. Pacient se také podívá na grafy svých naměřených hodnot, kde může vyčíst, jaká je jeho kompenzace diabetu.

Definice činnosti pacienta

Pacient zadá adresu pro přihlášení do webové aplikace. Vyplní pole uživatel a heslo a přihlásí se do systému. Je přesměrován na osobní stránku. Pacient na svém profilu ve webové aplikaci provádí následující kroky:

1. Načtení profilové stránky.
2. Přečte si komentář od lékaře.
3. Klikne na vybraný řádek s extrémní hodnotou a v zobrazeném formuláři.
4. Napíše novou poznámku vysvětlující okolnosti měření.
5. Novou poznámku uloží, klinutím na tlačítko „Odeslat“.

6. Klikne na jinou hodnotu v tabulce u měření po jídle a v zobrazeném formuláři uvede, kolik minut po jídle to bylo.

7. Změna počtu minut po měření, prodleva 5s.

8. Uloží provedené změny.

9. Napíše nový vzkaz lékaři.

10. Psaní příběhu, prodleva 15s.

11. Pacient si zobrazí časový graf.

12. Pacient si zobrazí časový graf v období 90 dní.

Jednotlivé kroky jsem zase převedl do podoby URL odkazů. Takto vytvořený seznam odkazů zase použiji jako šablonu scénáře pro jednoho uživatele v programu JMeter. V níže uvedeném seznamu neuvádím doménu na webovou aplikaci zadavatele, která je samozřejmě součástí v reálném testování.

1. `/cs/monitorace/monitorace-glykemie`

2. `Čtení komentářů lékaře, časová konstanta 15s.`

3. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta]/editovat-glykemii/[ID měření pacienta]`

4. `Psaní poznámky k~měření, časová konstanta 10s.`

5. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta]/editovat-glykemii/[ID měření pacienta]`

6. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta]/editovat-glykemii/[ID měření pacienta]`

7. `Změna počtu minut po měření, delay 5s.`

8. `/cs/monitorace/monitorace-glykemie/[pořadové číslo pacienta]/editovat-glykemii/[ID měření pacienta]`

9. `Psaní příběhu, Delay 15s.`

10. `/cs/monitorace/monitorace-glykemie`

TS3: Chceme znát, kolik pacientů může do aplikace zasílat souběžně data z glukometrů

Definice činnosti pacienta
Pacient každý den zasílá naměřená data do webové aplikace. Pacient měří podle předepsaného schématu měření a data posílá hromadně, např. ve večerních hodinách. Test simuluje večerní období, kdy všichni pacienti posílají svá denní měření glykemií. Denní měření obsahuje pět hodnot. Naměřené hodnoty jsou odesílány jako HTTP POST data z mobilní aplikace pro Android. Pacient tedy provádí následující činnost:

1. Odeslání naměřené hodnoty. (5x)

URL podoba scénáře:

1. `/cs/surveillance-data-glycemia` (5x)

TS4: Chceme znát, kolik diabetologů může prohlížet své pacienty a kolik pacientů může prohlížet svá data a zasílat data, tedy všechno souběžně

Diabetologové pracují v ordinačních hodinách. Diabetici průběžně posílají glykemie během celého dne a pokud naměří extrémní glykemii, přihlašují se do systému, aby zapsali poznámku k měření. K souběhu všech tří aktivit tedy může dojít v dopoledních i odpoledních hodinách pracovního dne. Odhad situace – lékařů je 10 x méně než pacientů (lékař se stará o 10 diabetiků). Tento testovací scénář kombinuje jednotlivé výše uvedené scénáře. Je zapotřebí vyladit jednotlivé databáze pro každý testovací scénář tak, aby odpovídal požadavkům tohoto scénáře.

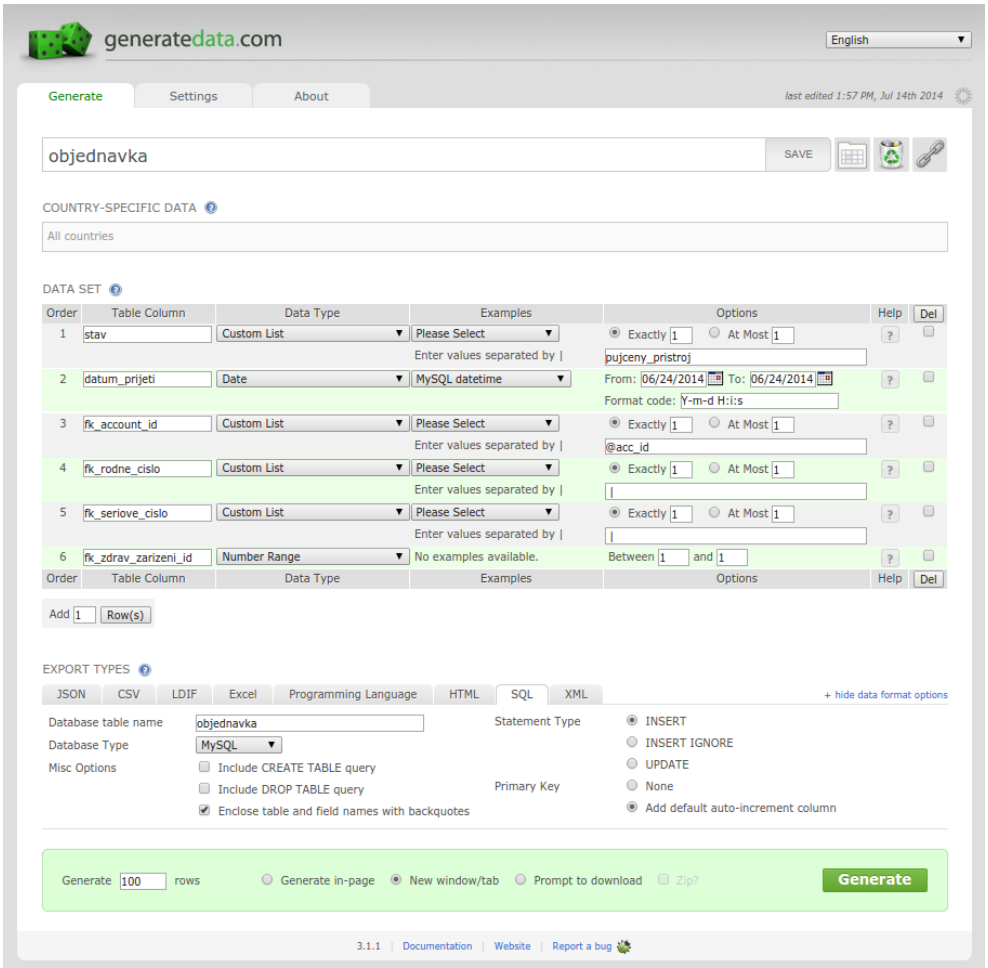
Kapitola 11

Příprava dat do testovaného systému

Abychom mohli provést zátěžové testy na testovaném systému, musíme vytvořit ukázková data, nad kterými budeme testování provádět. Jedná se o naplnění databáze, která obsahuje data potřebná pro testovaný systém. K našim testům je zapotřebí vytvořit ukázkové uživatelské účty pro role lékař a pacient. Pacientským účtům jsou následně přidělena data o naměřených hodnotách. Tímto naplněním databáze chceme dosáhnout stejného obsahu dat jako při reálném provozu testovaného systému.

Vytvoření vstupních souborů

Jedná se o soubory vytvořené programem Generatedata. Pro každou tabulku databáze je vytvořen formulář v tomto programu. Formulář obsahuje atributy např. jméno, příjmení, město, věk nebo telefon. Je zde možnost nastavení relativní četnosti generovaných polí dle zvoleného regionu. Tím nám při větším množství dat vzniknou duplicity, což se blíží stavu databáze při normálním používání. Podle definice testu vytvoříme požadovaný počet záznamů a ten uložíme do souboru s koncovkou `sql`. Například pro tabulku objednávka je vytvořen formulář, viz obrázek 11.1.
Vygenerované záznamy jsem ručně uložil do souboru `insert_objednavka.sql` v adresáři konkrétního testu. Žádný záznam v souboru není stejný. Vytvořený formulář lze uložit v programu Generatedata, pouze pokud je nainstalován na lokální stanici. Vstupní soubory se vytváří pro konkrétní databázi, která je použita u jednoho či více testů.



Obrázek 11.1: Formulář pro vytvoření dat tabulky objednávky.

Vytvoření doktorských účtů

Skript `create_doctor_insert.sh` má za úkol vytvořit soubor, který importuji do databáze. Skript obsahuje na začátku nápovědu. Je zde uveden přesný typ a počet vstupních souborů, který je nezbytný pro správné vytvoření výstupního souboru. Skript ověří správný počet vstupních souborů a existenci souborů, jinak vykáže chybu. Po ověření skript načte z každého souboru postupně první až poslední řádek. Série stejných řádků obsahuje data pro vytvoření jednoho uživatelského účtu. Vstupní soubory obsahují příkazy insert s vygenerovanými daty z programu Generatedata. Abychom dodrželi vztahy mezi jednotlivými tabulkami v databázi, je potřeba uložit si hodnotu naposledy vloženého záznamu do tabulky. K tomu jsem využil funkci `last_insert_id()`, kterou jsem uložil do uživatelské proměnné. Tuto proměnou následně použí-

vám při vložení dalšího záznamu do tabulky. Takto upravené příkazy insert jsou nahrány do výstupního souboru, pojmenovaného podle posledního názvu souboru při spuštění skriptu. Vytvořený výstupní soubor jsem uložil do adresářové struktury tak, aby jej bylo možné rychle najít a použít.

Spuštění skriptu z adresáře specifického testu vypadá následovně:

```
../../Create_scripts/create_doctor_insert.sh Inserts/
insert_account_doctor.sql Inserts/output_doctor.sql
```

1. `Inserts/insert_account_doctor.sql` je soubor s daty uživatelských účtů role doktor
2. `Inserts/output_doctor.sql` je výstupní soubor, který se nahrává do databáze

Vytvoření pacientských účtů

Skript `create_pacient_insert.sh` má za cíl vytvořit soubor s pacientskými účty, který se nahrává do databáze testovaného systému. Tak jako u předchozího skriptu je zde nápověda na začátku skriptu. Ošetřuje se pouze počet a existence vstupních souborů. Proces vytvoření výstupního souboru je v tomto případě složitější, protože uživatelský účet pacient má záznamy v několika na sobě závislých tabulkách. Jedná se o záznamy do tabulek: pacient, přístroj, objednávka, monitorace a `glyc_params`. ve skriptu jsou v cyklu načteny série stejných řádků ze vstupních souborů. Tato série obsahuje ukázková data pro jednoho uživatele a ty vkládám do výstupního souboru. Vytvořený výstupní soubor je přesunut do adresáře specifického testu.

Spuštění skriptu z adresáře specifického testu vypadá následovně:

```
../../Create_scripts/create_patient_insert.sh Inserts/
insert_account_patient.sql Inserts/insert_pacient.sql
Inserts/insert_pristroj.sql Inserts/insert_objednavka.sql
Inserts/insert_monitorace.sql Inserts/insert_glyc_params.sql
Inserts/output_patients
```

1. `Inserts/insert_account_patient.sql` obsahuje příkazy vytvoření uživatelských účtů role pacient
2. `Inserts/insert_pacient.sql` obsahuje příkazy vytvoření pacientů
3. `Inserts/insert_pristroj.sql` obsahuje příkazy vytvoření přístrojů
4. `Inserts/insert_objednavka.sql` obsahuje příkazy vytvoření objednávek

5. `Inserts/insert_monitorace.sql` obsahuje příkazy vytvoření monitorace
6. `Inserts/insert_glyc_params.sql` obsahuje příkazy vytvoření parametrů glykémie pacienta
7. `Inserts/output_patients` je výstupní soubor pro nahrání do databáze k vytvoření uživatelských účtů pacientů.

Vytvoření naměřených hodnot

Skript `create_glyc_data.sh` má za cíl vytvořit soubor obsahující naměřená data pro každého pacienta v systému. Na začátku skriptu se nachází jednoduchá nápověda jak skript spustit a popis vstupních souborů. Ošetřuje se pouze počet a existence vstupních souborů. Mezi vstupními soubory je `sql` soubor s předem vytvořenými naměřenými daty. Skript provádí jednoduchou operaci nahrazení hodnoty cizího klíče `fk_monitorace` a `fk_account_id` za hodnoty na vstupu programu, které odpovídají pořadovému číslu pacienta a identifikačnímu číslu prvního uživatelského účtu pacienta v systému. Příklad spuštění skriptu vypadá následovně:

```
../../Create_scripts/create_glyc_data.sh Inserts/glyc_data.sql
100 104 Inserts/output_glyc_data
```

1. `Inserts/glyc_data.sql` je soubor obsahující ukázková data – šablona, vytvořená programem Generatedata
2. `100` je číslo určující počet pacientů
3. `104` je identifikační číslo prvního uživatelského účtu role pacient
4. `Inserts/output_glyc_data` je výstupní soubor obsahující naměřená data pro daný počet pacientů

Vložení výstupních souborů do databáze

Výstupní soubory, námi pojmenované `output_[funkce souboru]`, vygenerované pomocí skriptu, se vloží do databáze testovaného systému:

1. Vložení uživatelských účtů doktorů
   `mysql -p test1 < output_doctor.sql`
2. Vložení uživatelských účtů pacientů
   `mysql -p test1 < output_patients.sql`

3. Vložení naměřených hodnot pacientů
   `mysql -p test1 < output_glyc_data.sql`

Toto pořadí je důležité a musí být dodrženo.

Ověření validních dat v databázi

Tento proces není nutný, ale doporučený. Je potřeba základní znalost sql. Je doporučeno zkontrolovat množství námi nahraných dat. Tyto údaje porovnáme s údaji v popisu test case, pro který byla data připravena. Konkrétně počet uživatelských účtů pacientů, doktorů, počet objednávek, který by měl odpovídat počtu pacientů. U monitorací musí být nastaven termín ukončení monitorace pozdější než v den testování, jinak není monitorace aktivní.

Vytvoření zálohy databáze

Po úspěšném vložení a ověření dat do databáze jsem vytvořil zálohu specifické databáze. Zálohu jsem uložil do adresáře pro konkrétní databázi. Adresáře jsou číslovány od jedničky. V adresáři se nachází soubor „README.ini“, který obsahuje popis dané databáze. V popisu konkrétního testu se nachází odkaz na specifickou databázi. Pro ukázku přikládám příkaz pro vytvoření zálohy specifického testu TC332 (jde o pojmenování podle stávajícího systému):

```
mysqldump -p test1 > ../../db/db1/backup_TC332.sql
```

Při opakování zátěžového testu se jednoduše provede obnova dat databáze ze zálohy. K tomu použijte následující příkaz:

```
mysqldump -p test1 < ../../db/db1/backup_TC332.sql
```

Odpadá tak složitější a časově náročný postup nahrávání výstupních souborů `output_[funkce souboru]` do databáze.

Kapitola 12

Definování a vyhodnocení zátěžových testů

Jednotlivé testy vycházejí ze tří kritických procesů webové aplikace. Každý kritický proces nazýváme testovací scénář. Ten má definované kroky, které tvoří šablonu testovacího scénáře pro program JMeter. Každý zátěžový test má svůj adresář pojmenovaný podle systému uvedeného v Rally. Ten pojmenovává test jako „test case“ a jeho hodnota je TC + číslo, které označuje celkové pořadové číslo test case v systému Rally. Jednotlivé testy se liší počtem uživatelů, množstvím dat v databázi anebo časovou prodlevou mezi spuštěním scénářů. Všechny zátěžové testy jsou monitorovány a jejich průběh je zaznamenáván. Každý test byl spuštěn minimálně třikrát. První test pro každý scénář byl navržen s nižším počtem uživatelů a celkovým objemem dat. Měl za úkol zjistit chování testovaného systému. Na základě jeho výsledku jsem definoval další testy. Tyto testy jsou popsány níže u jednotlivých testovacích scénářů. Ke každému testu je vytvořena jedna databáze, jedna sada testovacích skriptů, konfigurace webového a databázového serveru, která je pro toto zadání u všech testů stejná.

Výstupní soubory

Výstupem každého provedeného testu je sada žurnálovacích souborů z programu JMeter. Sada znamená, že každý testovací skript má dva žurnálovací soubory. Jeden obsahuje data programu JMeter oddělená čárkou a v druhém souboru jsou data uložena ve formátu XML. Tento XML výstup lze pomocí nástrojů JAnalyser, BlazeMeter Plugin nebo Loadosophia [11] vizualizovat do grafů a analyzovat. Nástroj Loadosophia použiji pro analýzu mých výstupních souborů. Další výstupy máme z nástrojů Dstat, který zaznamenává stav systémových prostředků. Výstup z nástroje Top ukládám v průběhu spuštěného testu do výstupního souboru. Ten slouží pro určení běžících procesů a

v případě zjištění kritických hodnot lze zjistit, který proces mohl způsobit tento problém. Ke všem níže definovaným testům existují tyto výstupní soubory a jsou uloženy v příloze práce v adresáři `pf_tests`. K vybraným testům přiložím do této práce grafy z analyzačního nástroje Loadometer. Grafy využití systémových prostředků jsou vytvořené, z výstupních dat programu Dstat, skriptem `graph_all.sh`, který využívá program Gnuplot.

TS1: Chceme znát, kolik diabetologů souběžně může kontrolovat své pacienty

Pro tento scénář jsem provedl celkem šest testů. Testy pro tento scénář se liší počtem uživatelských účtů lékařů a počtem pacientů, kdy jeden lékař má na starosti vždy deset pacientů. Rozdílnou hodnotou je také délka období již naměřených dat pacientů v tabulce `glyc_data`. U konkrétních testů jsou uvedeny přesné hodnoty, které zde byly použity. Testy jsem vytvořil pomocí skriptu `create_test_plan_US648.sh`. Skript obsahuje šablonu vytvořenou v programu JMeter podle přesně definovaných kroků testovacího scénáře č. 1 v URL adresách. Šablona je upravena tak, aby bylo možné vytvářet modifikované testy podle zadaného počtu lékařů. Na začátku skriptu je nápověda, která obsahuje funkci skriptu, ukázku spuštění skriptu a popis všech vstupů do skriptu. Skript ověřuje počet vstupních parametrů a existenci vstupních souborů. Po ověření dochází k postupnému načítání uživatelských účtů lékařů. Provede se vytěžení přihlašovacích údajů, které se vloží v podobě proměnné na místo, kde dochází k přihlašování uživatele do testovacího systému v šabloně skriptu. Po procesu přihlášení lékaře nastává cyklus kontroly pacientů sériově. Pro detailní popis je skript přiložen v příloze. Předpokládá se znalost skriptovacího jazyka BASH.

Test 1

- 10 uživatelských účtů lékař
- 100 uživatelských účtů pacient
- Délka období 6 měsíců odpovídá 91500 záznamů

Test 2

- 10 uživatelských účtů lékař
- 100 uživatelských účtů pacient
- Délka období 12 měsíců odpovídá 183000 záznamů

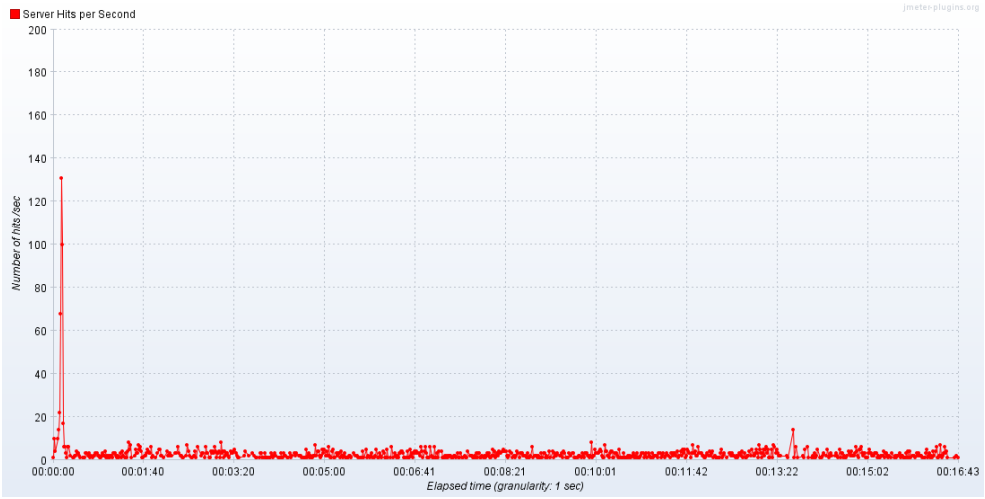
Test 3

- 15 uživatelských účtů lékař
- 150 uživatelských účtů pacient
- Délka období 6 měsíců odpovídá 137250 záznamů

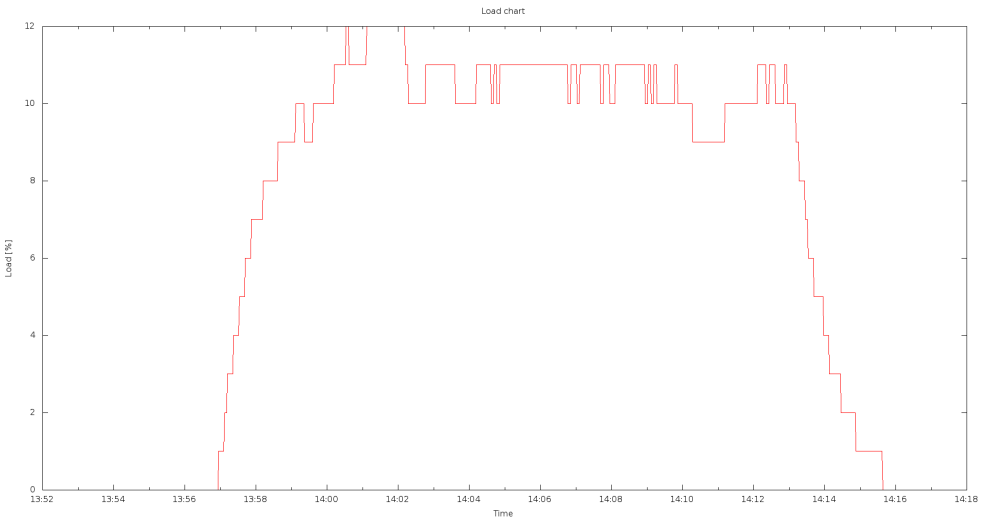
Test 4

- 15 uživatelských účtů lékař
- 150 uživatelských účtů pacient
- Délka období 12 měsíců odpovídá 274500 záznamů

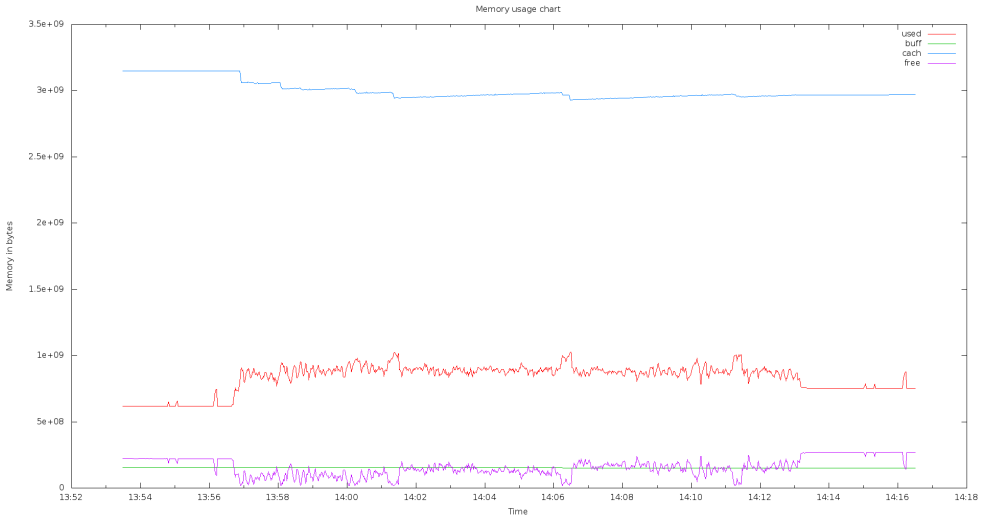
Hodnocení testu č. 4
Zátěžový test dokázal využít zhruba 12% testovaného systému. Jelikož jsem spustil všechny testovací skripty najednou, zátěž vznikla nárazově, a to rovnou na hranici 10%, kde zůstala až do ukončení testu. Na obrázku 12.1 vidíme v první minutě testu vysoký počet příchozích požadavků za vteřinu. Další příchozí požadavky se periodicky opakují ve vlnách s maximem 10 „hitů“ za vteřinu. Velmi podobné chování můžeme vidět u využití operační paměti na obrázku 12.3 a také u vstupně výstupních operací na disku viz obrázek 12.4. Chování testovaného systému hodnotím jako stabilní. Průměrná doba odezvy překročila hranici pěti vteřin. Pokud však uvážíme, že při délce zátěžového testu, zhruba šestnáct minut a počtu příchozích HTTP požadavků viz tabulka 12.1, je hodnota doby odezvy dostačující. Upozornil bych na velmi nízkou hodnotu volné operační paměti, která již před spuštěním testu byla pod hranicí 250MB. Po spuštění zátěžového testu došlo okamžitě k jejímu vyčerpání. Zde je potřeba navýšit operační paměť, abychom předešli časově náročným operacím „swap“.



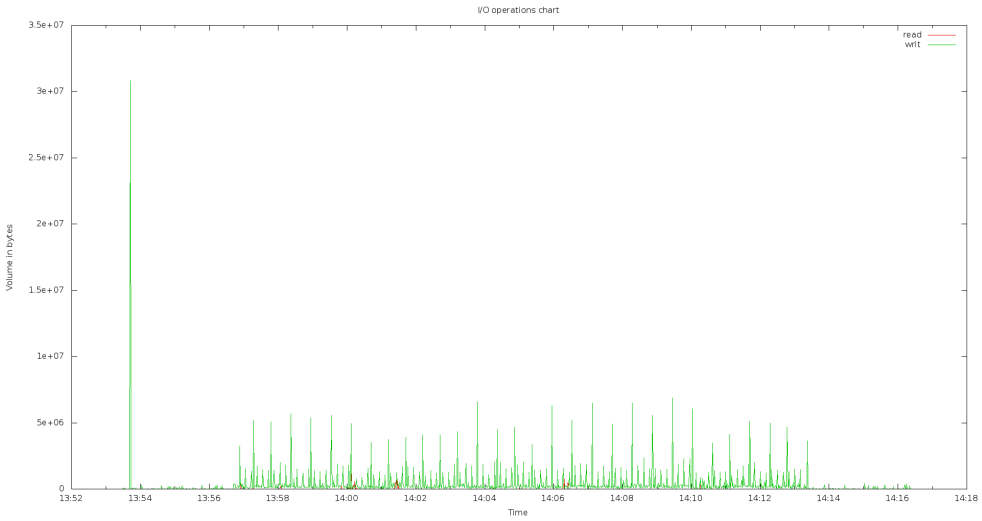
Obrázek 12.1: Test 4: Počet požadavků vytvořený virtuálními uživateli za vteřinu



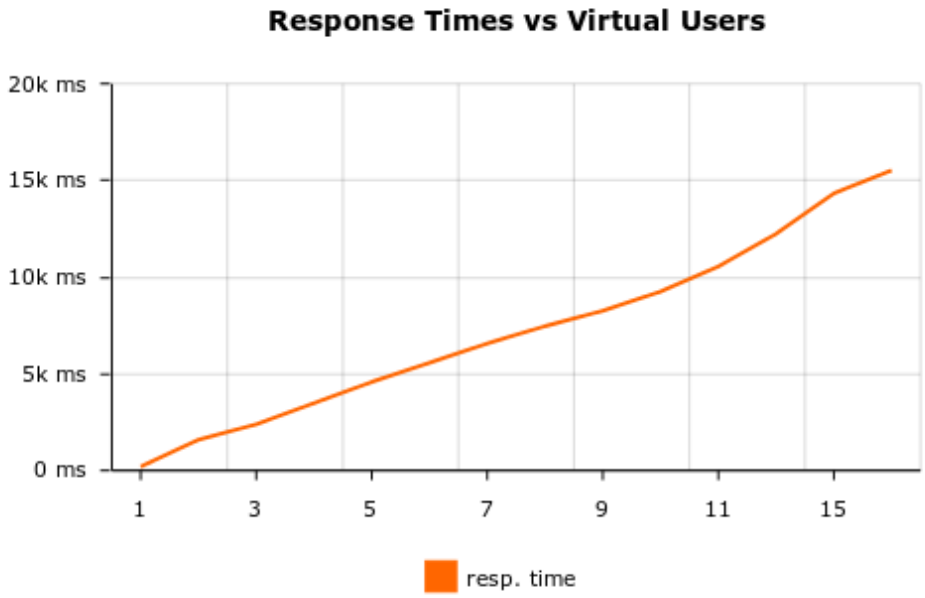
Obrázek 12.2: Test 4: Graf zátěže testovaného serveru



Obrázek 12.3: Test 4: Graf využití operační paměti



Obrázek 12.4: Test 4: Graf vstupně výstupních operacích na disku



Obrázek 12.5: Test 4: Graf závislosti virtuálních uživatelů na době odezvy požadavku

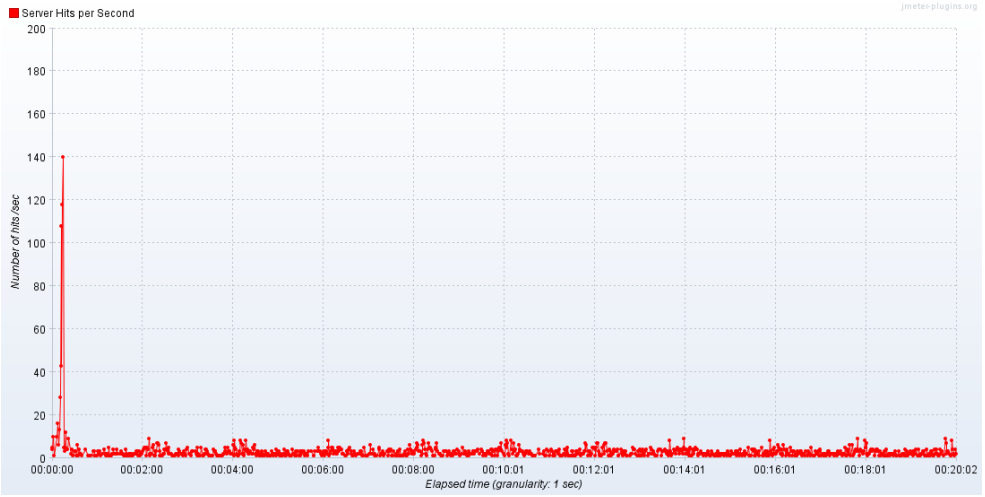
Test 5

- 20 uživatelských účtů lékař
- 200 uživatelských účtů pacient
- Délka období 6 měsíců odpovídá 183000 záznamů

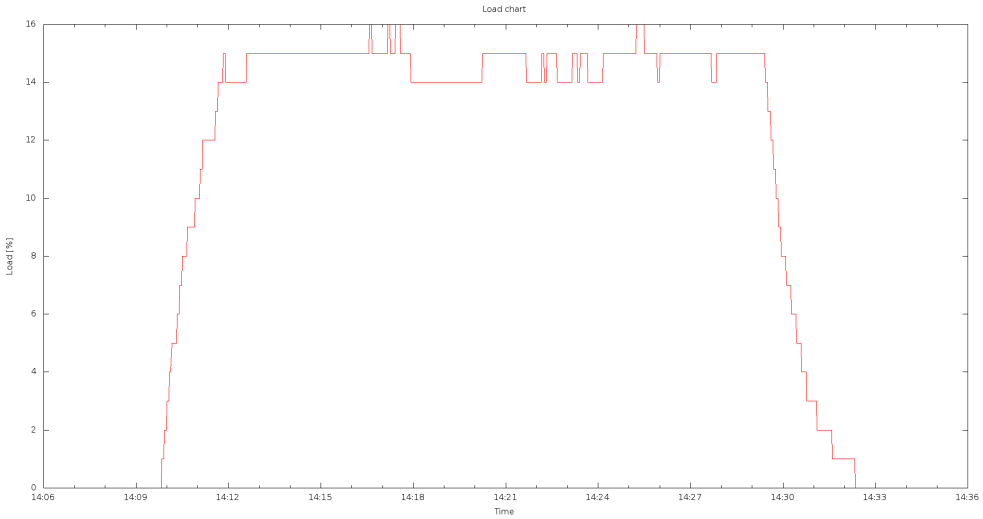
Test 6

- 20 uživatelských účtů lékař
- 200 uživatelských účtů pacient
- Délka období 6 měsíců odpovídá 366000 záznamů

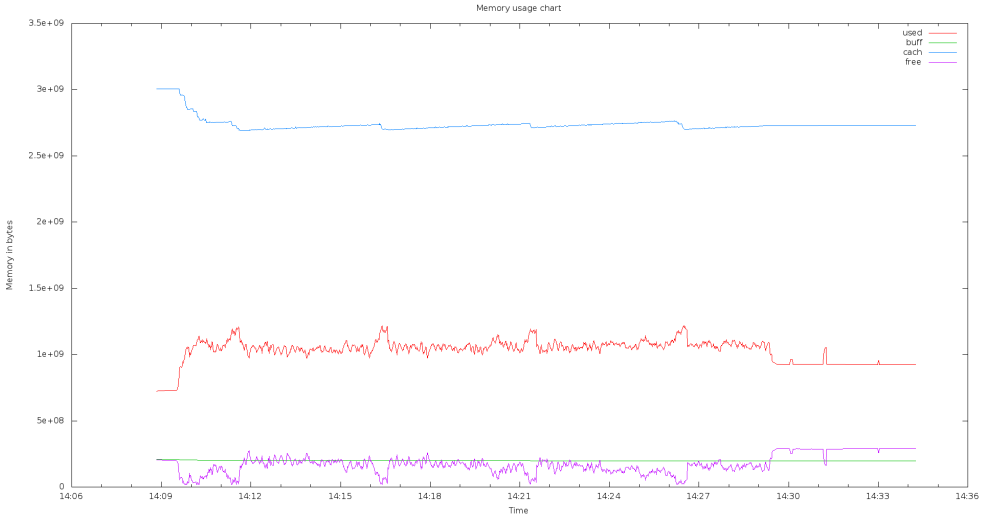
Hodnocení testu č. 6
Tento zátěžový test se velice podobá předchozímu zátěžovému testu č. 4. Jeho grafy jsou obdobné s nepatrně vyššími hodnotami zátěže a příchozích požadavků za vteřinu. Opakuje se zde problém s nízkou hodnotou volné paměti. Test má vyšší hodnotu propustnosti a průměrná doba odezvy je o to nižší než u testu č. 4. Testovaný systém hodnotím jako stabilní s upozorněním na nízkou hodnotu volné paměti.



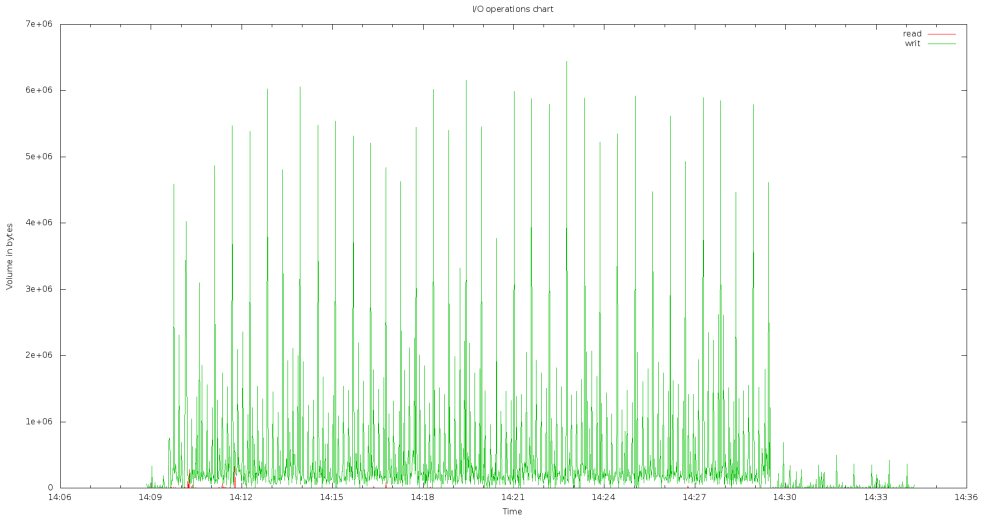
Obrázek 12.6: Test 6: Počet požadavků vytvořený virtuálními uživateli za vteřinu



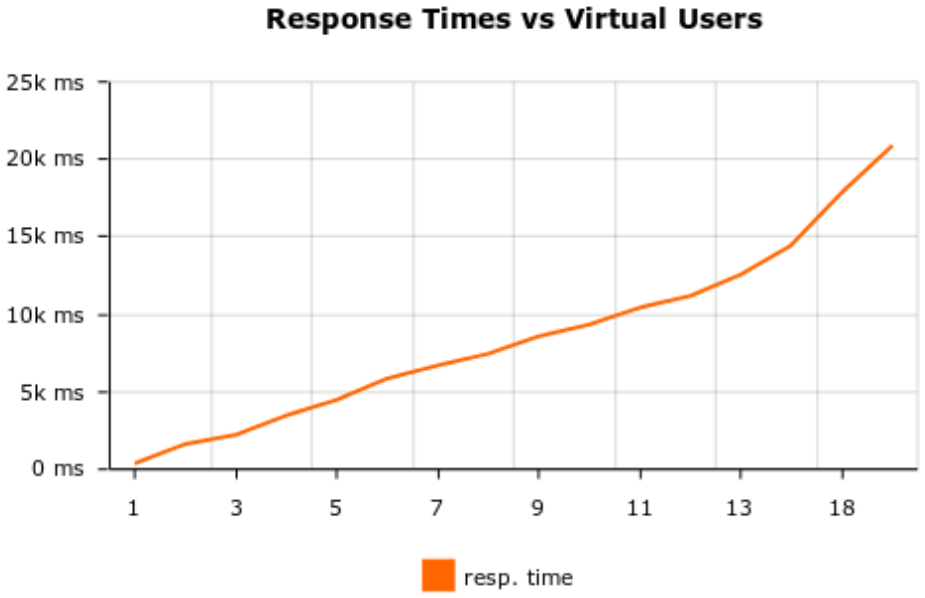
Obrázek 12.7: Test 6: Graf zátěže testovaného serveru



Obrázek 12.8: Test 6: Graf využití operační paměti



Obrázek 12.9: Test 6: Graf vstupně výstupních operacích na disku



Obrázek 12.10: Test 6: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Shrnutí testovacího scénáře č. 1

Provedené zátěžové testování nevykázalo žádnou kritickou hodnotu, kterou bychom neočekávali nebo nedokázali vysvětlit. Zátěž na testovaný systém po spuštění testu vždy prudce vzroste až do určité úrovně zátěže, která je po dobu testu stejná. Potenciální problém vidím s využitím volné paměti. Ta je již před spuštěným testem na velmi nízkých hodnotách. Doporučení pro zadavatele je navýšit volnou paměť. Toho můžeme docílit snížením velikosti paměti pro cache v nastavení serveru. V níže uvedené tabulce 12.1 můžete vidět naměřené hodnoty testů prvního scénáře. Jedná se o průměr doby odezvy požadavků, procento neúspěšně vyřízených požadavků a propustnost. Propustnost definujeme jako počet vytvořených požadavků za vteřinu.

	počet pož.	**průměr[ms]**	**nevyřízeno[%]**	**propust.[pož./s]**
Test 1	1120	3 206	0,12	1,63
Test 2	1120	2 279	0,15	1,80
Test 3	1680	4 727	0,42	1,93
Test 4	1680	5 786	0,32	1,70
Test 5	2240	7 840	0,31	1,87
Test 6	2240	5 432	0,15	2,37

Tabulka 12.1: Naměřené výsledky pro testovací scénář č. 1

TS2: Chceme znát, kolik pacientů může prohlížet svá data souběžně

Pro tento testovací scénář bylo provedeno 7 testů. První test jsem zvolil s nižším počtem pacientů a následně jsem u dalších testů zvyšoval jejich počet. Platí předpoklad, že jeden lékař má na starosti přesně deset pacientů. Testy se dále liší počtem již uložených záznamů naměřených hodnot v tabulce `glyc_data`. Jednotlivé testovací skripty byly vytvořeny pomocí skriptu `create_test_plan_US742.sh`. Na začátku skriptu je nápověda jak spustit skript a popis jednotlivých vstupních parametrů. Vstupními soubory jsou uživatelské účty pacientů, které jsou již vytvořeny v databázi a pacienti, kterým je nastavena monitorace. Program skriptu obsahuje šablonu pro program JMeter, vytvořenou dle zadání testovacího scénáře. Ze vstupních souborů vytěžuji přihlašovací údaje uživatelských účtů a jméno a příjmení pacienta, pro kterého se zrovna vytváří testovací scénář.

Test 1

- 1 lékař
- 10 pacientů

- Délka období 6 měsíců odpovídá 9150 záznamů

Test 2

- 5 lékařů
- 50 pacientů
- Délka období 6 měsíců odpovídá 45750 záznamů

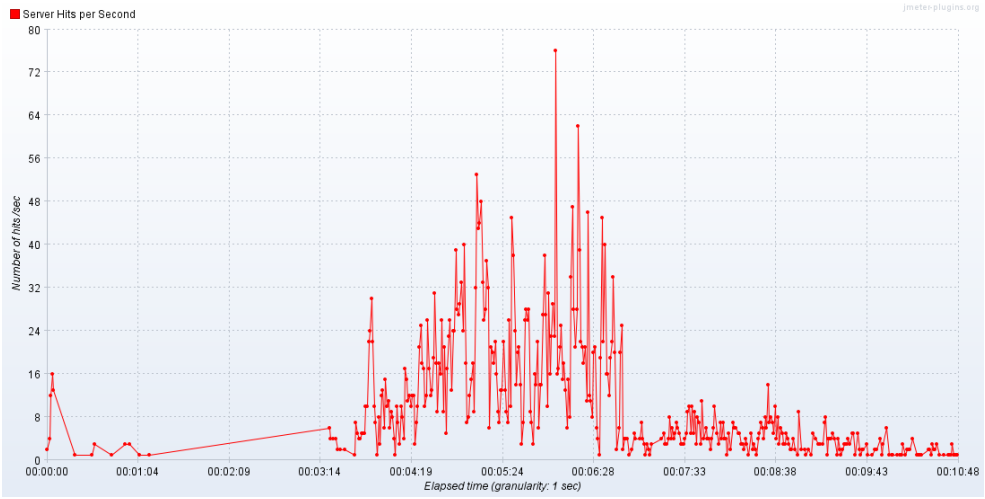
Test 3

- 5 lékařů
- 50 pacientů
- Délka období 12 měsíců odpovídá 91500 záznamů

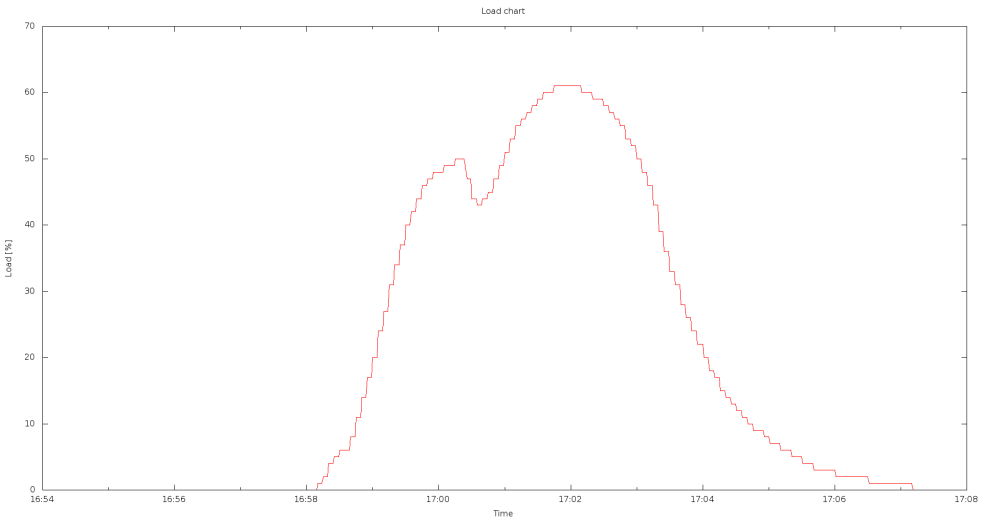
Test 4

- 8 lékařů
- 80 pacientů
- Délka období 6 měsíců odpovídá 73200 záznamů

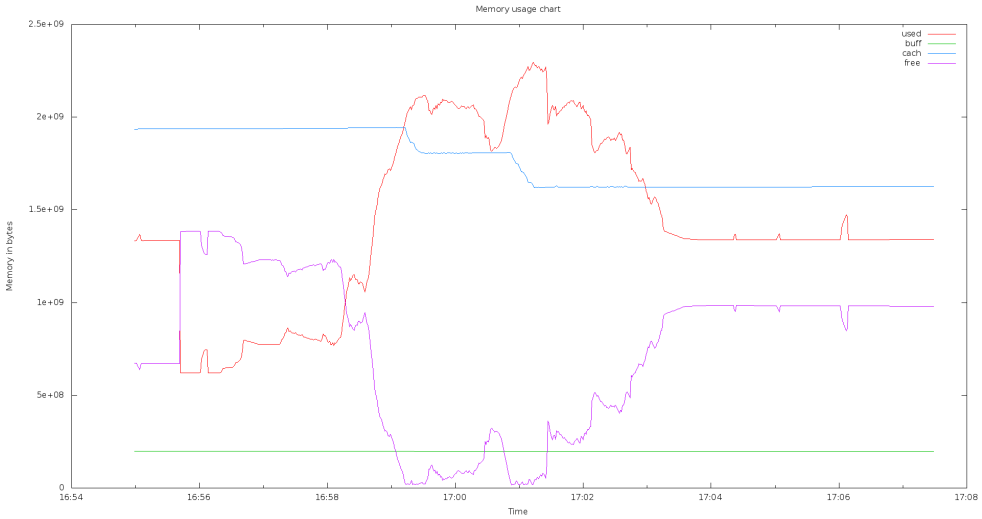
Hodnocení testu č. 4
Zátěžové testování jsem spustil až po čtyřech minutách monitorování testovaného systému. Chtěl jsem zachytit chování systému bez zátěže. Po spuštění zátěžového testu vidíme pozvolný nárůst požadavků za vteřinu, což je přímo úměrné zatížení testovaného systému viz 12.12. Spuštěním osmdesáti testovacích skriptů jsem způsobil okamžité vyčerpání zhruba 1,4GB volné paměti. Její nedostatek se projevil na snížení paměti cache, což může nepříznivě ovlivnit dobu odezvy načítání stránek. U tohoto testu jsem dosáhl k nevyřízení 6% HTTP požadavků, což představuje 48 chyb. Největší problém vidím v příliš vysoké době odezvy, viz tabulka 12.2. Tuto hodnotu považuji za velmi nedostatečnou a výkon testovaného systému za nízký. Provedený test simuluje práci téměř poloviny počtu pacientů dle požadavků na aplikaci za den. Jedná se o test hraniční zátěže, jelikož jsem vytvořil příliš vysokou zátěž,
na kterou již server nedokázal v přijatelné době reagovat. Pozitivní ovšem je, že po skončení generování zátěže testovaný systém dokázal zpracovat téměř všechny příchozí požadavky. Maximální využití výkonu testovaného systému dosahuje 60% a zásadní problém vidím v nedostatku volné operační paměti.



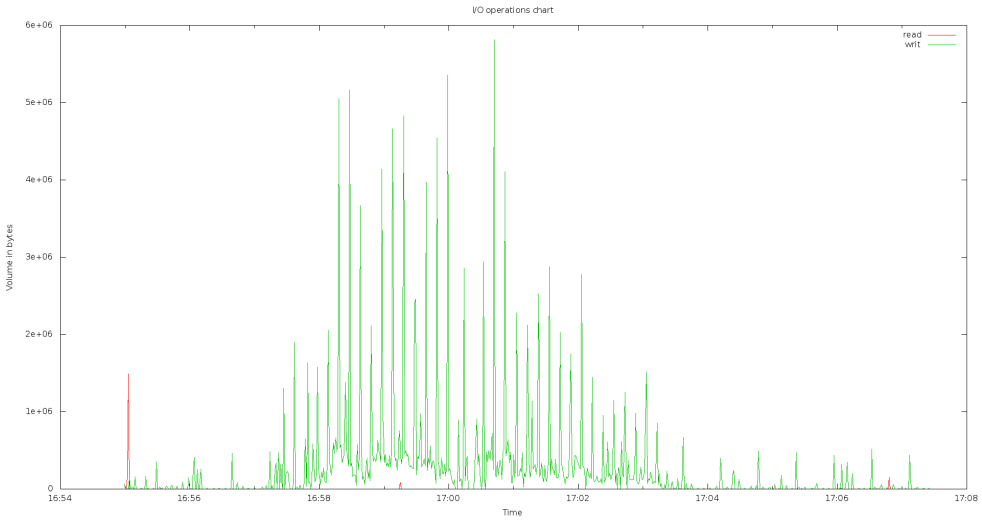
Obrázek 12.11: Test 4: Počet požadavků vytvořený virtuálními uživateli za vteřinu



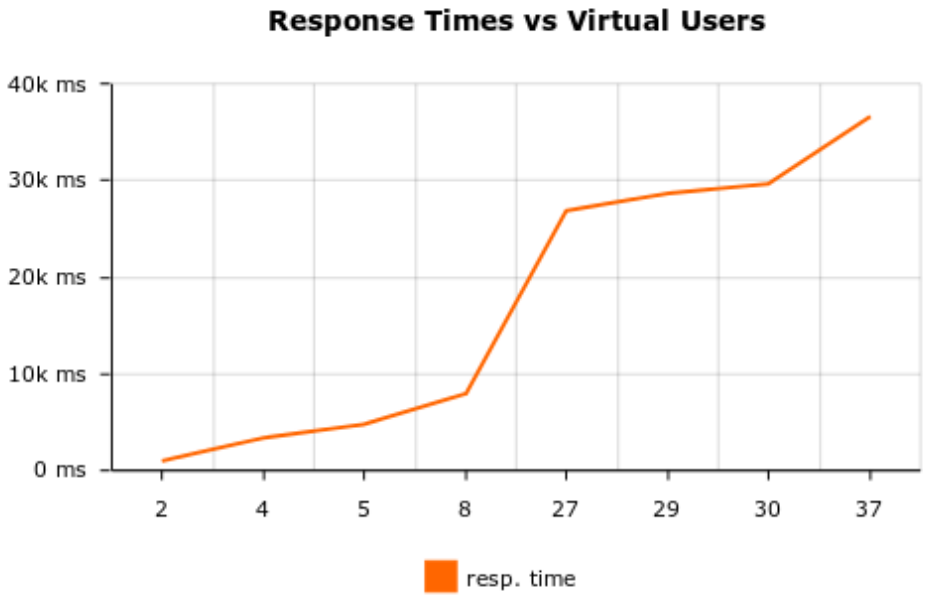
Obrázek 12.12: Test 4: Graf zátěže testovaného serveru



Obrázek 12.13: Test 4: Graf využití operační paměti



Obrázek 12.14: Test 4: Graf vstupně výstupních operacích na disku



Obrázek 12.15: Test 4: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Test 5

- 8 lékařů
- 80 pacientů
- Délka období 12 měsíců odpovídá 146400 záznamů

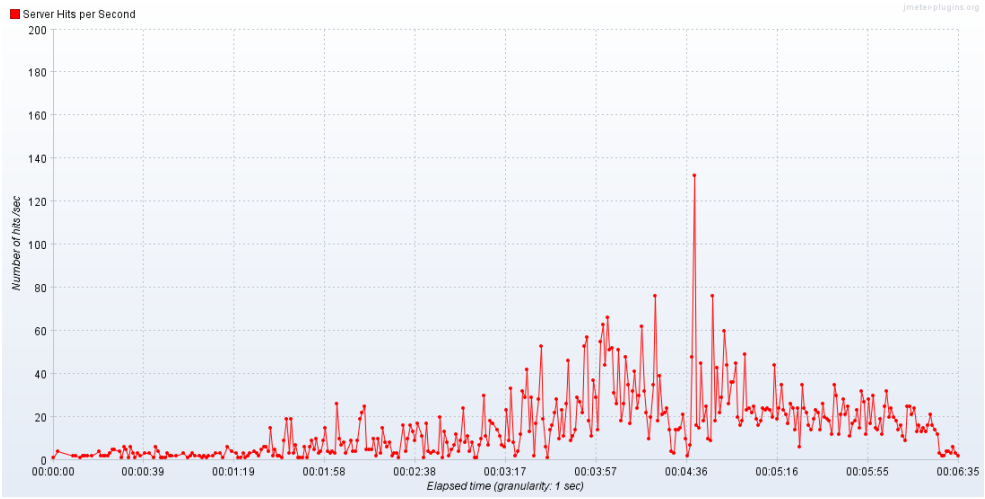
Test 6

- 10 lékařů
- 100 pacientů
- Délka období 6 měsíců odpovídá 91500 záznamů

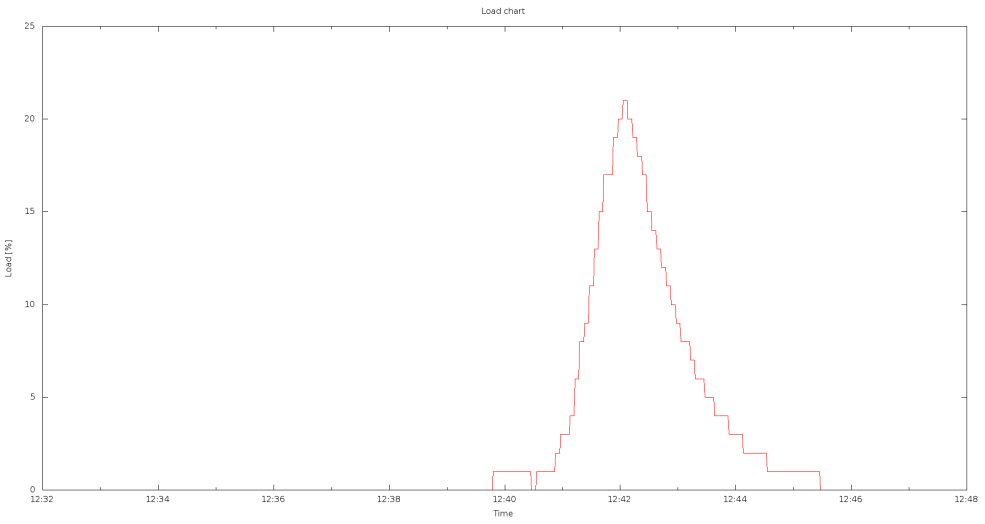
Test 7

- 10 lékařů
- 100 pacientů
- Délka období 12 měsíců odpovídá 183000 záznamů

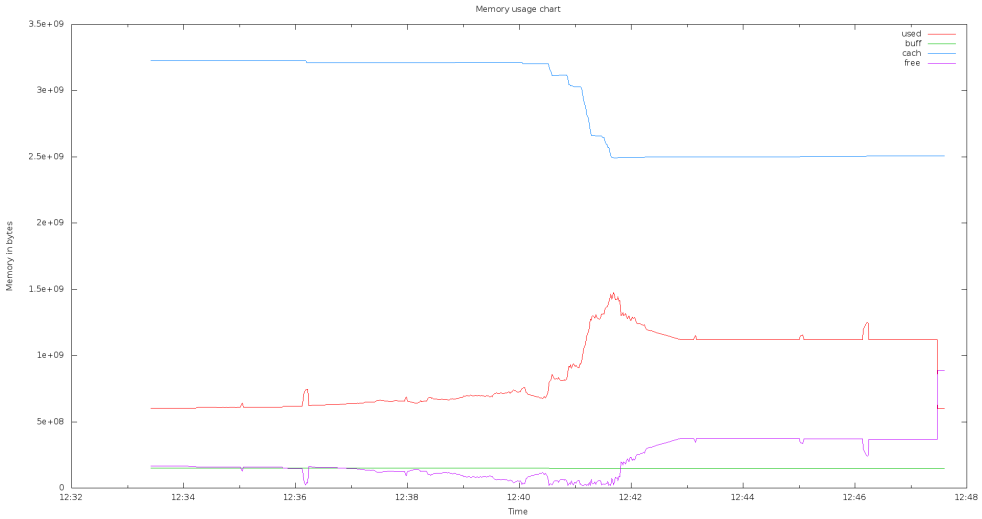
Hodnocení testu č. 7
Provedený test má povahu testu hraniční zátěže, která se u zde projevila ve vysokém procentu nevyřízených požadavků. Jedná se o 85% viz 12.2. Hodnotit výsledky provedeného testu nemá smysl. Zajímá nás pouze chování testovaného systému k této situaci. Z pohledu na obrázek 12.18 je patrné, že již před spuštěním testu bylo k dispozici velmi málo volné paměti. Po spuštění testu a vyčerpání volné paměti došlo k alokaci volné paměti z prostoru vyhrazeného pro paměť cache. Tento proces se provedl několikrát, až do hranice 2,5GB pro paměť cache. Téměř okamžitě poté co došlo k vyčerpání poslední volné paměti a nebylo možné uvolnit další paměť z prostoru cache paměti, lze vidět strmý pokles zatížení testovaného systému, uvolnění zhruba 1GB paměti a ukončení činnosti zápisu na disk. Došlo nejspíše k řízenému odmítnutí čekajících HTTP požadavků na server z důvodu nedostatku volné paměti. Opět tedy narážím na problém s velikostí operační paměti. Chování testovaného systému hodnotím dobře. Určitě není žádoucí bezdůvodně odmítat příchozí HTTP požadavky, ale v této situaci to bylo vhodné řešení, jak zachovat testovaný systém i nadále v provozu.



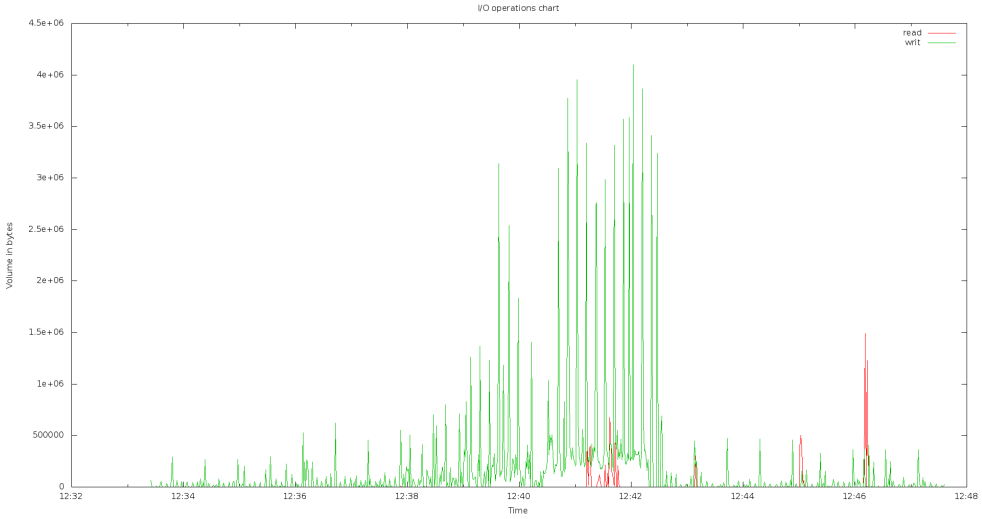
Obrázek 12.16: Test 7: Počet požadavků vytvořený virtuálními uživateli za vteřinu



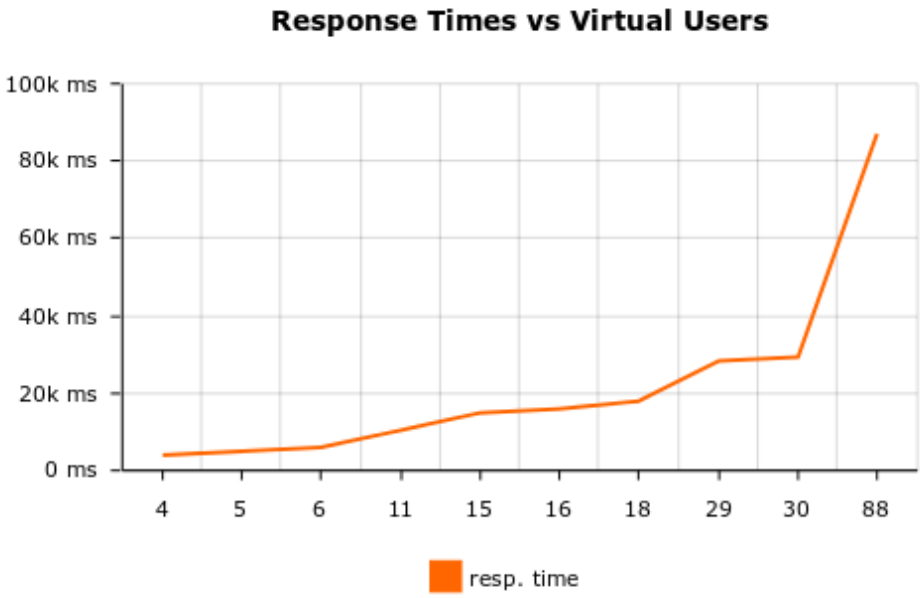
Obrázek 12.17: Test 7: Graf zátěže testovaného serveru



Obrázek 12.18: Test 7: Graf využití operační paměti



Obrázek 12.19: Test 7: Graf vstupně výstupních operacích na disku



Obrázek 12.20: Test 7: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Shrnutí testovacího scénáře č. 2

Mnou navržené testy tohoto testovacího scénáře byli velmi náročné pro námi testovaný systém. Pouze první provedený test s definovanými 100 pacienty v aplikaci má přijatelné výstupy. Dokázal obsloužit 98 požadavků za zhruba 5 minut, což je 1176 požadavků za hodinu. Tato výsledná hodnota je téměř dvanácti násobkem denního požadavku na aplikaci. Další provedené testy jsou zbytečně náročné. Jako možné doporučení pro snížení doby odezvy webové aplikace vidím v implementaci AJAX řešení. Webová stránka aplikace by po prvním požadavku uživatele načetla všechny potřebné objekty stránky a při přechodu na další stránky v aplikaci by stahovala pouze obsah této stránky, protože ostatní objekty by již měla načtené. V níže uvedené tabulce 12.2 můžete vidět naměřené hodnoty testů druhého scénáře. Jedná se o průměr doby odezvy požadavků, procento neúspěšně vyřízených požadavků a propustnost. Propustnost definujeme jako počet vytvořených požadavků za sekundu.

	počet pož.	**průměr[ms]**	**nevyřízeno[%]**	**propust.[pož./s]**
Test 1	100	2800	2	1
Test 2	500	15065	7	2
Test 3	500	16329	0	2
Test 4	800	26593	6	2
Test 5	800	28493	7	2
Test 6	1000	24593	56	2
Test 7	1000	21059	85	2

Tabulka 12.2: Naměřené výsledky pro testovací scénář č. 2

TS3: Chceme znát, kolik pacientů může do aplikace zasílat souběžně data z glukometrů

K tomuto scénáři bylo definováno 14 testů. V průběhu testování jsme navrhovali nové a odlišné testy z důvodu velmi krátkého časového intervalu, kdy byl test aktivní. Pro první tři testy je nastavena prodleva mezi odesíláním naměřených hodnot do systému 10 sekund. Tato hodnota byla po konzultaci upravena, jelikož neodpovídala reálnému intervalu. Měřicí přístroje dokáží naměřené hodnoty pacientů odesílat v mnohem kratším intervalu, a to zhruba do 2 sekund. Další testy již mají nastavenou prodlevu 2 sekundy. Byly vytvořeny a provedeny další čtyři testy s kratší prodlevou. Dosud provedené testy spouštěly konkrétní počet definovaných scénářů najednou. Upravil jsem tedy spouštěcí skript tak, aby spustil pouze určitý počet (dále skupinu) scénářů paralelně a vyčkal na dokončení naposledy spuštěného scénáře. Po jeho dokončení spustil další skupinu. Tímto rozdělením scénářů do skupin jsme dosáhli delšího intervalu testu. Dalším rozdílem od předchozích testů je periodicky se opakující zátěž na testovaném systému. Vytvořil jsem tak dlouhodobý

zátěžový test. U jednotlivých testů rozlišujeme počet spuštěných testů paralelně, počet pacientů, kde platí, že jeden lékař má na starosti 10 pacientů, velikost databáze, nad kterou je prováděno testování, a časovou prodlevou mezi naměřenými hodnotami odeslanými do aplikace. Testy jsem vytvářel pomocí skriptu `create_test_plan_US743.sh`, který obsahuje ukázku spuštění skriptu s popisem vstupních parametrů. Jeden výstupní scénář pro program JMeter obsahuje pět požadavků, jejichž součástí jsou naměřené hodnoty pro jednoho pacienta. Testovací skripty se spouští pomocí spouštěcího skriptu, který je vždy uložen v první úrovní adresáře testu. Název spouštěcího skriptu není pro všechny testy stejný.

Test 1

- Spuštěno 10 testů
- 10 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 9150 záznamů
- Prodleva mezi požadavky je 10 sekund

Test 2

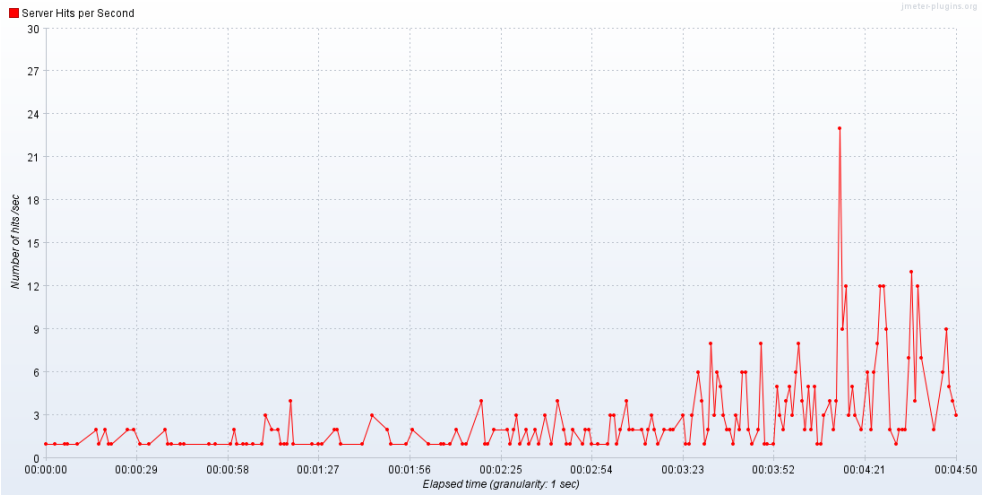
- Spuštěno 50 testů
- 50 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 45750 záznamů
- Prodleva mezi požadavky je 10 sekund

Test 3

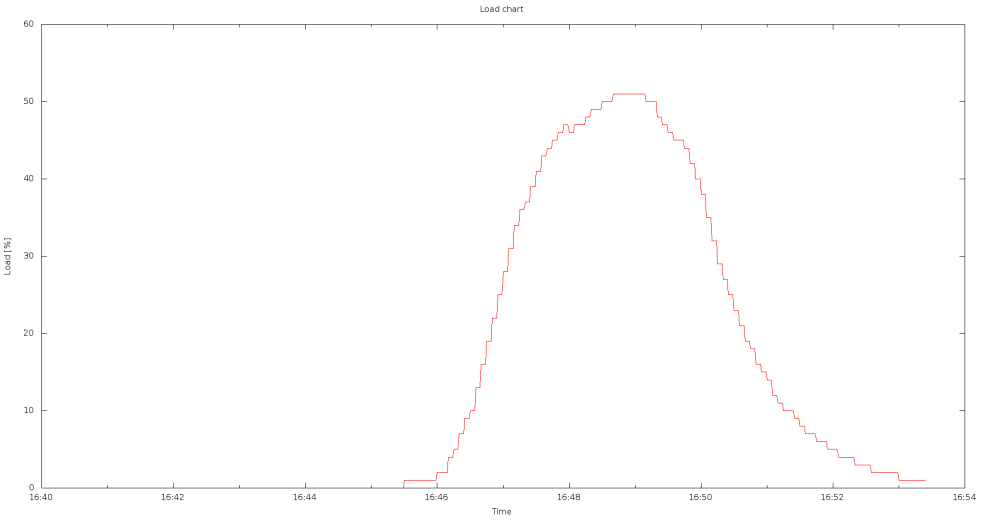
- Spuštěno 100 testů
- 100 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 91500 záznamů
- Prodleva mezi požadavky je 10 sekund

Hodnocení testu č. 3
Testovací skript po spuštění generuje HTTP POST požadavky s hodnotami naměřených dat pacientů. Počáteční i středová část testu probíhá dobře, bez výrazného nárůstu zátěže na testovaný systém, viz obrázek 12.22. Změna nastává v posledních třech minutách, kdy zatížení testovaného systému prudce roste až do hodnoty 50%. S touto reakcí současně začala ubývat volná paměť a zvýšil se objem dat pro zápis na disk. Délka zátěžového testu je zhruba

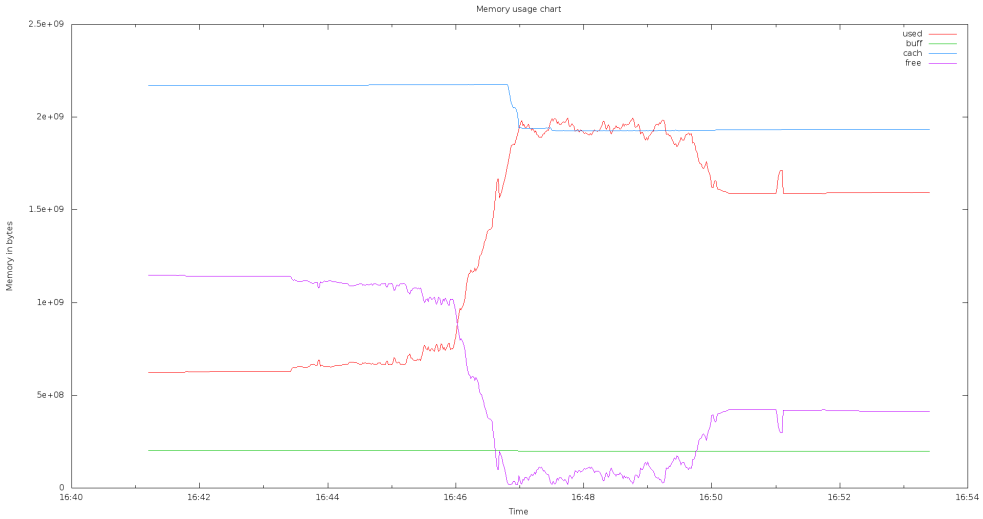
5 minut a za tuto dobu bylo odesláno ze 100 zařízení 500 naměřených hodnot pacientů. Tento test za velmi krátký časový úsek vygeneroval polovinu požadované denní dávky pro testovaný systém. Doba odezvy požadavků přesahuje hranici 6,5 vteřin, ale s ohledem na krátký časový úsek, ve kterém byl test spuštěn, hodnotím test jako úspěšný s doporučením navýšení volné paměti.



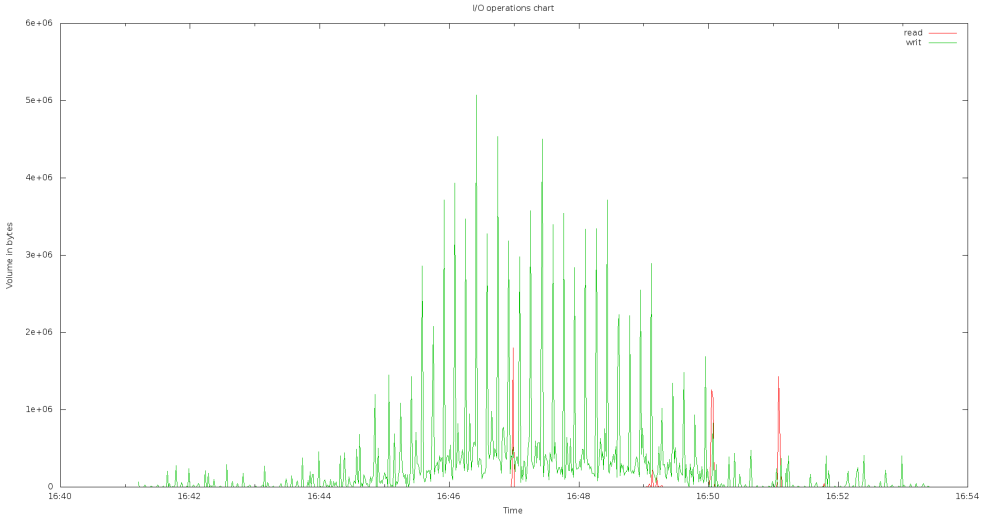
Obrázek 12.21: Test 3: Počet požadavků vytvořený virtuálními uživateli za vteřinu



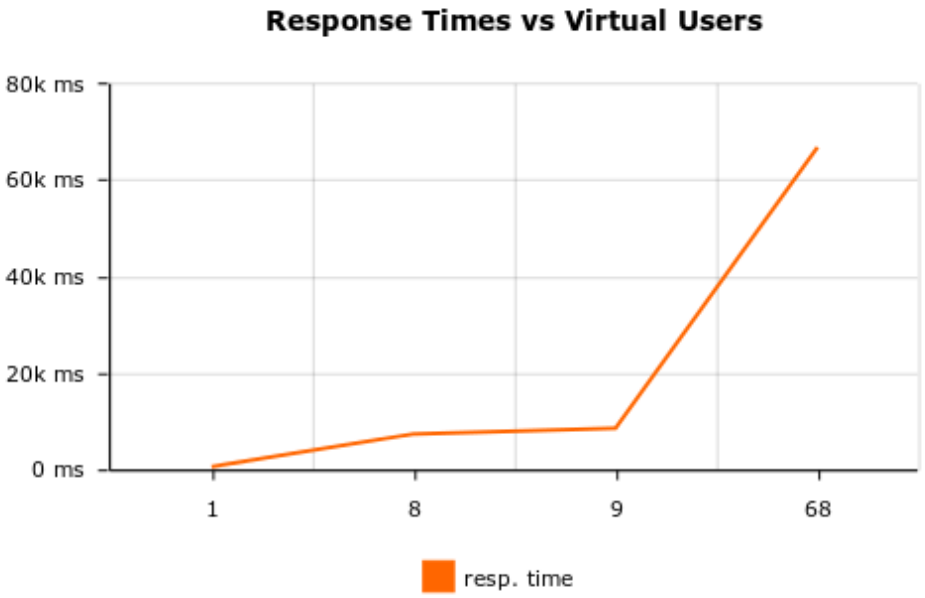
Obrázek 12.22: Test 3: Graf zátěže testovaného serveru



Obrázek 12.23: Test 3: Graf využití operační paměti



Obrázek 12.24: Test 3: Graf vstupně výstupních operacích na disku



Obrázek 12.25: Test 3: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Test 4

- Spuštěno 10 testů
- 10 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 9150 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Test 5

- Spuštěno 50 testů
- 50 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 45750 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Test 6

- Spuštěno 100 testů
- 100 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 91500 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Test 7

- Spuštěno 135 testů
- 135 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 137250 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Test 8

- Spuštěn 1 test
- 100 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 91500 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Test 9

- Spuštěno 10 testů
- 100 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 91500 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Délka testu je zhruba 165 sekund

Test 10

- Spuštěno 15 testů
- 150 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 137250 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Délka testu je zhruba 165 sekund

Test 11

- Spuštěno 20 testů
- 200 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 183000 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

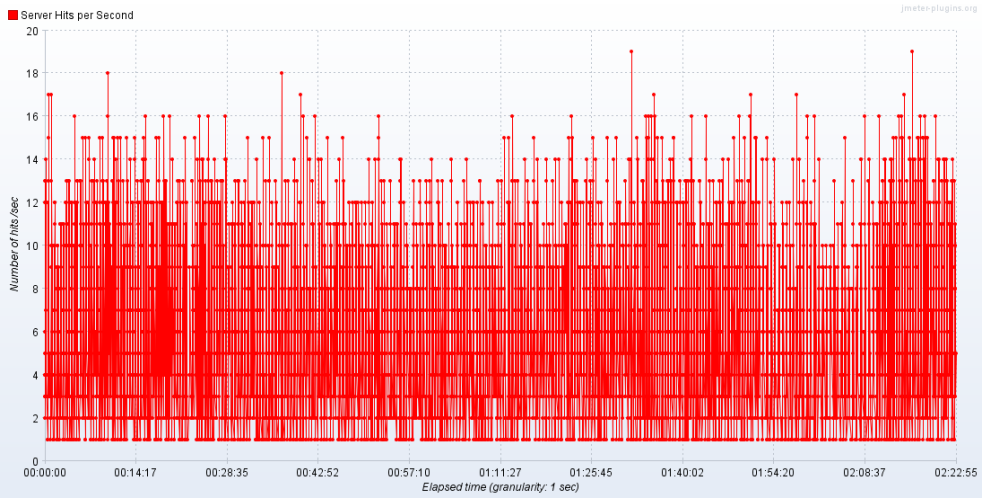
Délka testu je zhruba 300 sekund

Test 12

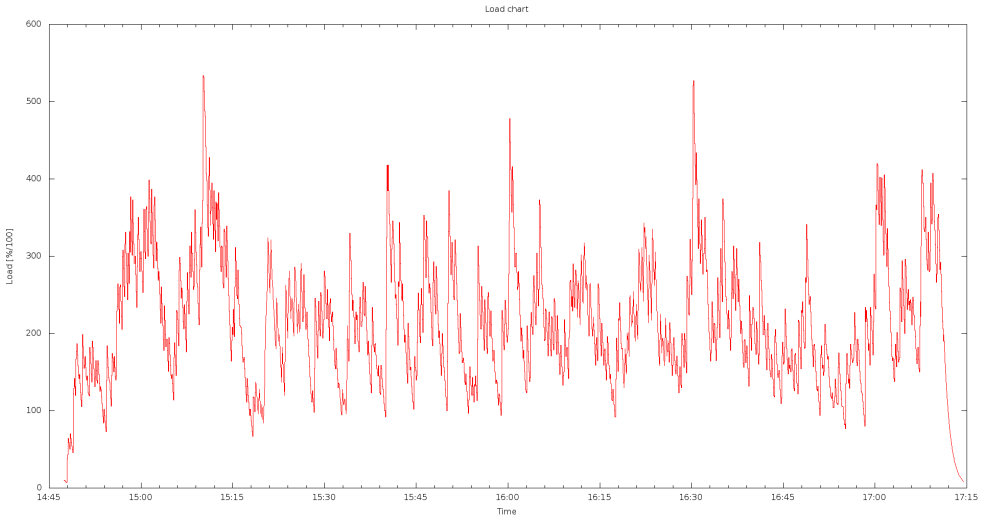
Test byl spuštěn 30 krát, abychom dosáhli dlouhodobého zátěžového testu.

- puštěno 20 testů
- 200 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 183000 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

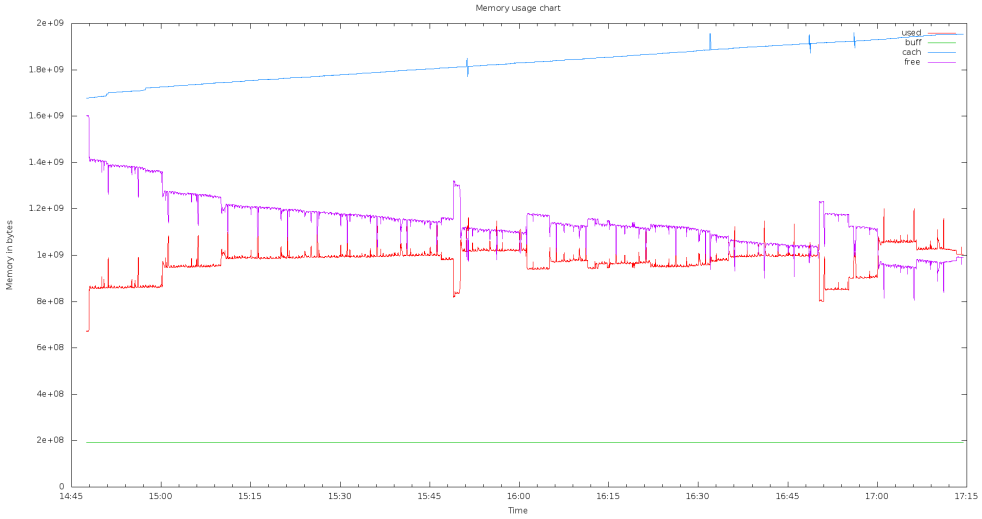
Hodnocení testu č. 12
Hlavním cílem tohoto testu bylo zjistit chování webového serveru při dlouhodobém zatížení. Testování probíhalo tak, že se 200 pacientů, kteří chtějí odesílat svá data z glukometru, rozdělí do 10 skupin. Skupiny se pouští jedna za druhou tak, aby jsem server zatěžoval stabilně. Po celou dobu 2,5 hodinového testu se mi podařilo testovaný systém udržovat pod zátěží 20 „hitů“ za vteřinu a procentuálně jsem využil do 6% výkonu testovaného systému. Test od svého spuštění využil přes 600MB volné paměti, které i v tom nejvytíženějším úseku bylo pořád dostatek, viz obrázek 12.28. Jemné odchylky v tomto grafu jsou spojeny se zvýšeným zápisem dat na disk, viz obrázek 12.29. Počet testovaných glukometrů u tohoto testu přesně odpovídá požadavku zátěže denní dávky na testovaný systém. S průměrnou dobou odezvy 754ms viz tabulka 12.3, hodnotím provedený test jako úspěšný. V průběhu zátěžového testu jsem nezaznamenal žádné nestandardní chování testovaného systému a troufám si tvrdit, že tato zátěž byla pro testovaný systém nízká.



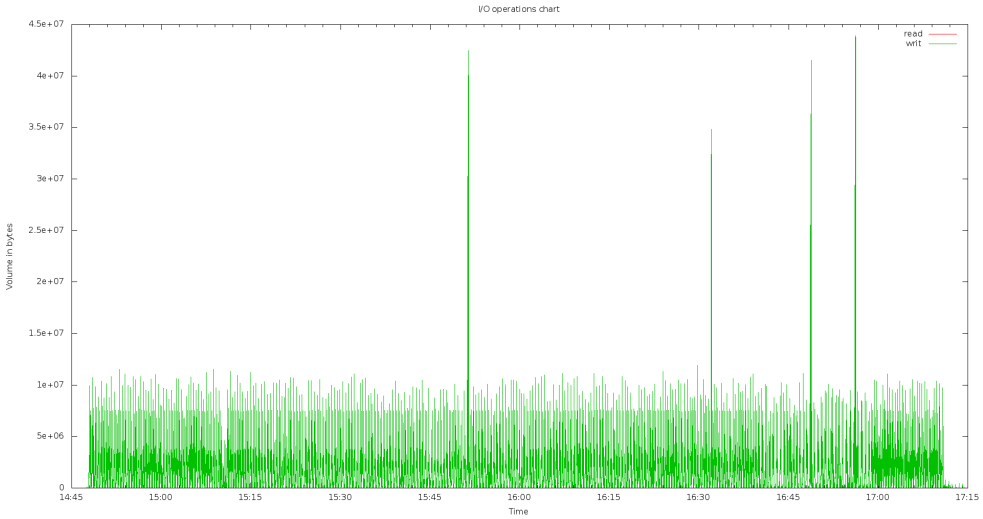
Obrázek 12.26: Test 12: Počet požadavků vytvořený virtuálními uživateli za vteřinu



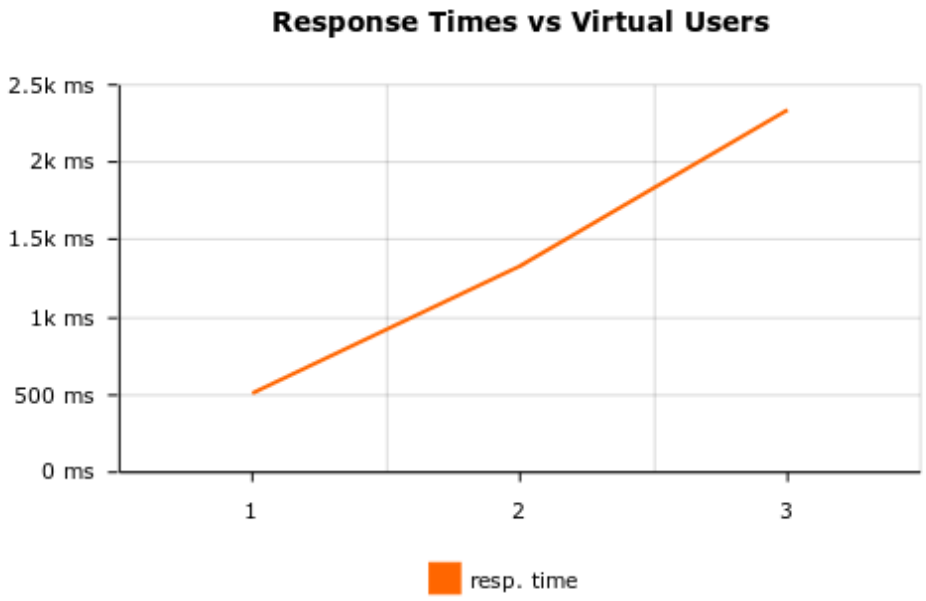
Obrázek 12.27: Test 12: Graf zátěže testovaného serveru



Obrázek 12.28: Test 12: Graf využití operační paměti



Obrázek 12.29: Test 12: Graf vstupně výstupních operacích na disku



Obrázek 12.30: Test 12: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Test 13

Test byl spuštěn 10 krát, abychom dosáhli dlouhodobého zátěžového testu.

- Spuštěno 50 testů
- 50 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 45750 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

Délka testu je zhruba 480 sekund.

Test 14

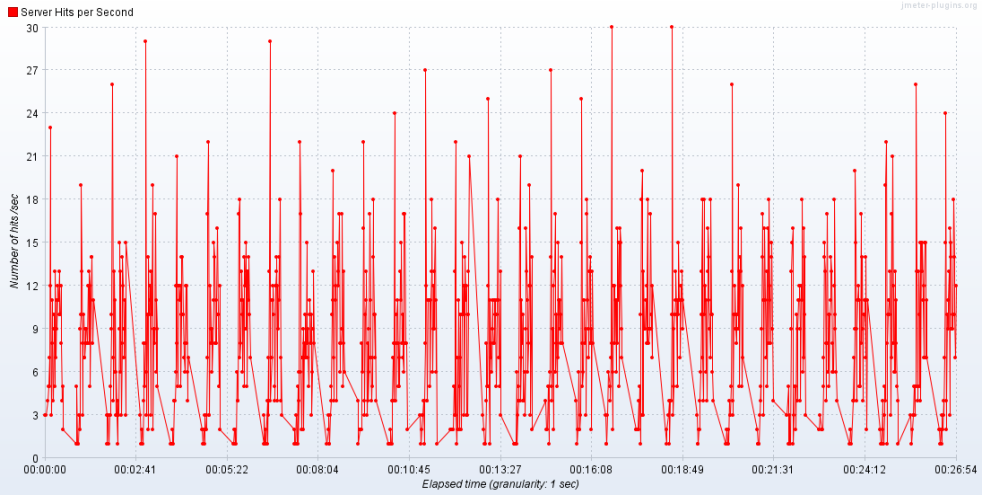
Test byl spuštěn 30 krát, abychom dosáhli dlouhodobého zátěžového testu.

- Spuštěno 50 testů
- 50 pacientů odesílá svá data
- Délka období 6 měsíců odpovídá 45750 záznamů
- Prodleva mezi požadavky jsou 2 sekundy

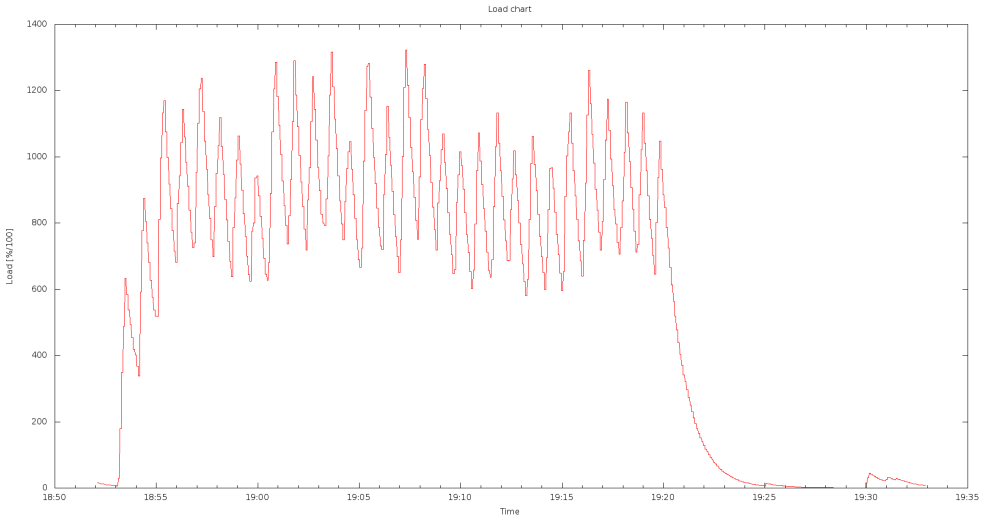
Délka testu je zhruba 1560 sekund.

Hodnocení testu č. 14
Cílem tohoto testu je zjistit chování webového serveru při dlouhodobém zatížení. Testování probíhá spuštěním 50 testovacích skriptů souběžně. Po ukončení posledního spuštěného skriptu, se automaticky spouští znovu stejná série 50 testovacích skriptů. Toto se opakuje 30 krát, tak abychom nasimulovali stabilní zátěž na testovaný systém. Na obrázku 12.31 lze vidět 30 téměř identických period, kde jedna perioda vyjadřuje spuštění 50 testovacích skriptů. Spouštění těchto skriptů ve 30 periodách lze také vidět na obrázku 12.34. Vytvořená zátěž na testovaný systém byla téměř u všech period stejná, a to zhruba 10%. Práce systému s volnou pamětí vypadá dobře. Spuštěním první série dojde ke snížení zhruba 400MB volné paměti. Další snížení o zhruba 200MB nastává se spuštěním další série skriptů. Snižování se zastaví po třetí sérii na hodnotě 400MB volné paměti a na této úrovni zůstává až do konce celého testu. V nejvytíženějším úseku testu je stále k dispozici přes 200MB volné paměti. Na obrázku 12.34 vidíme nečekaný nárůst zápisu dat na disk. K tomu došlo po spuštění čtvrté série skriptů. Pro zjištění co způsobilo tuto odchylku jsem nahlédnul do výstupu z programu Top, kde jsem hledal proces nebo procesy, které mohly způsobit tento problém. Bohužel jsem nenašel nic co by to způsobilo. Jelikož tato odchylka vznikla pouze jednou, považuji ji za anomálii. Výsledky zátěžového testu hodnotím kladně, stejně jako chování

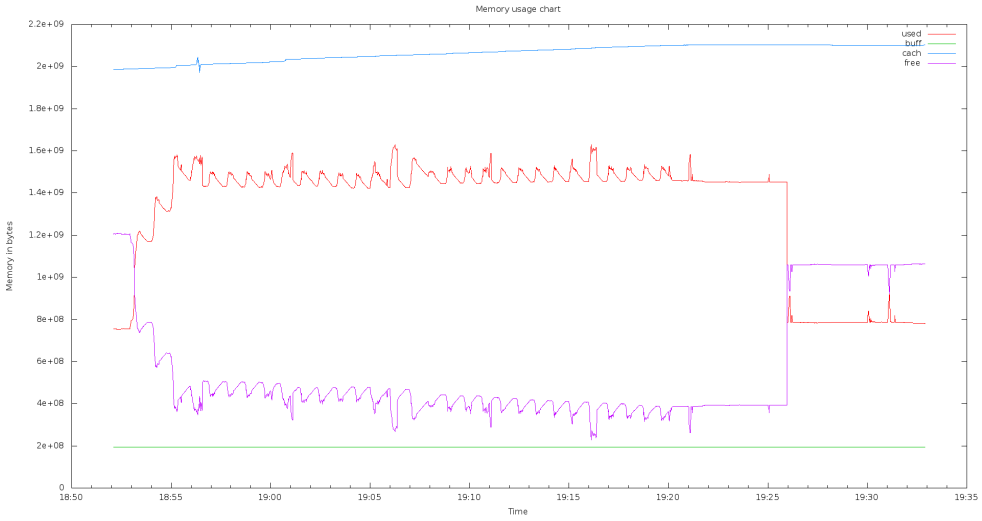
testovaného systému. Ten za necelých 30 minut testu dokázal obsloužit 1500 glukometrů s přijatelnou dobou odezvy 2,4 vteřiny.



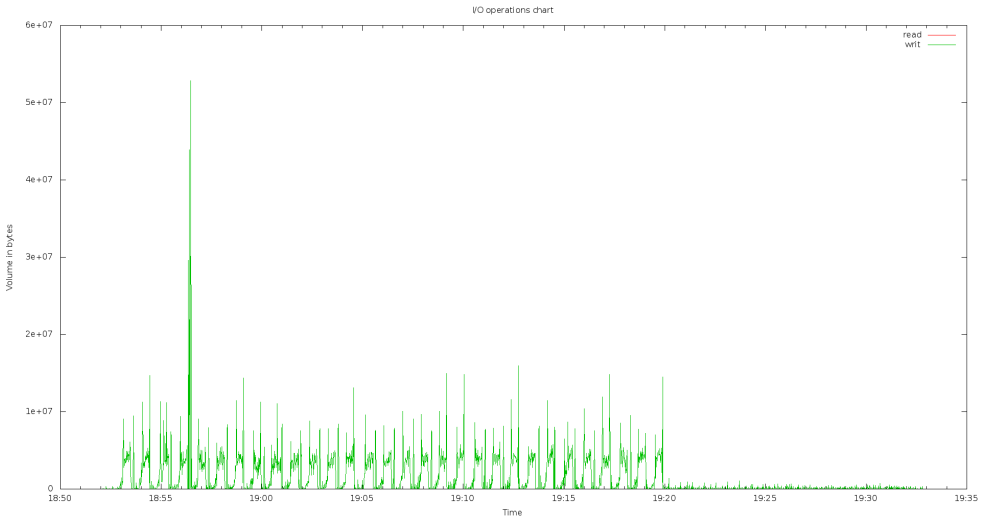
Obrázek 12.31: Test 14: Počet požadavků vytvořený virtuálními uživateli za vteřinu



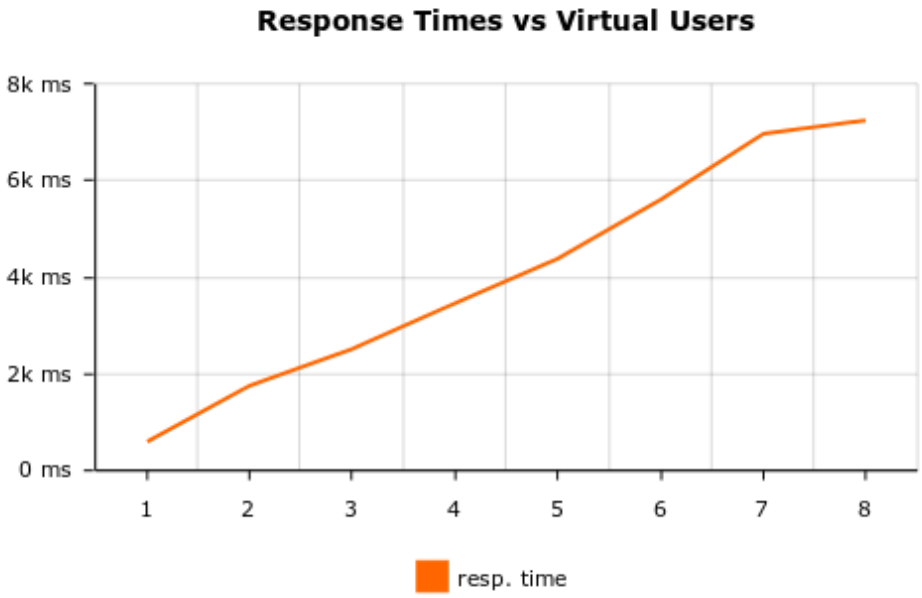
Obrázek 12.32: Test 14: Graf zátěže testovaného serveru



Obrázek 12.33: Test 14: Graf využití operační paměti



Obrázek 12.34: Test 14: Graf vstupně výstupních operacích na disku



Obrázek 12.35: Test 14: Graf závislosti virtuálních uživatelů na době odezvy požadavku

Shrnutí testovacího scénáře č. 3

Naměřené hodnoty tohoto scénáře splňuji naše očekávání. Chování testovaného systému hodnotím jako dobré a stabilní, jelikož jsme dokázali využít pouze jednotky až maximálně dvě desítky procent testovaného sytému. V níže uvedené tabulce 12.3 můžete vidět naměřené hodnoty testů třetího scénáře. Jedná se o průměr doby odezvy požadavků, procento neúspěšně vyřízených požadavků a propustnost. Propustnost definujeme jako počet vytvořených požadavků za sekundu. V některých případech jsme naměřili počet požadavků za minutu.

	# pož.	**průměr[ms]**	**nevyřízeno[%]**	**propust.[pož./s]**
Test 1	50	406	0	1
Test 2	250	1043	10	3
Test 3	500	6693	1	3
Test 4	50	436	0	4
Test 5	250	1986	0	8
Test 6	500	7531	2	4
Test 7	675	13677	4	2
Test 8	500 [11]	348	0	22,8/min
Test 9	500	318	0	3
Test 10	750	623	0	3,3
Test 11	1000	651	14	3
Test 12	30000	754	14	3,5
Test 13	2500	2747	0	5
Test 14	7500	2439	0	5

Tabulka 12.3: Naměřené výsledky pro testovací scénář č. 3

TS4: Chceme znát, kolik diabetologů může prohlížet své pacienty a kolik pacientů může prohlížet svá data a zasílat data, tedy všechno souběžně

Tento testovací scénář zahrnuje předchozí tři testovací scénáře a vytváří tak zátěžové testy kombinující práci lékařů, činnost pacientů a odesílání dat. Jejich kombinací se snažíme docílit vytížení širšího spektra testovaného systému. Ze všech dříve provedených testů se níže uvedené testy nejvíce blíží běžnému provozu na testovaném systému. Obsahuje stejné množství záznamů jako předchozí testy, ale počet příchozích požadavků na testovaný systém je třikrát větší. V případě posledního testu jsme zátěž dvou testovacích scénářů opakovali, abychom dosáhli delšího intervalu zátěžového testu. Pro spouštění

[11] Test byl spuštěn pouze jednou.

testů existuje spouštěcí skript, který se nachází v první úrovni adresáře daného testu. Ten je nastaven tak, aby sám spustil potřebné scénáře. Spuštění testu tohoto scénáře je tedy úplně stejné jako u předchozích scénářů.

Test 1

- 10 pacientů
- 1 lékař
- Délka období 6 měsíců odpovídá 9150 záznamů

Test 2

- 20 pacientů
- 2 lékaři
- Délka období 6 měsíců odpovídá 18300 záznamů

Test 3

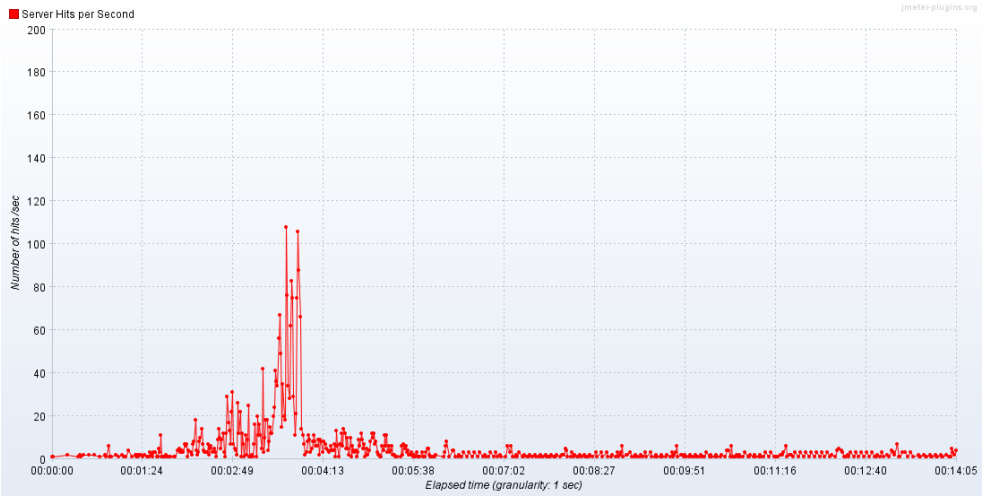
- 30 pacientů
- 3 lékaři
- Délka období 6 měsíců odpovídá 27450 záznamů

Test 4

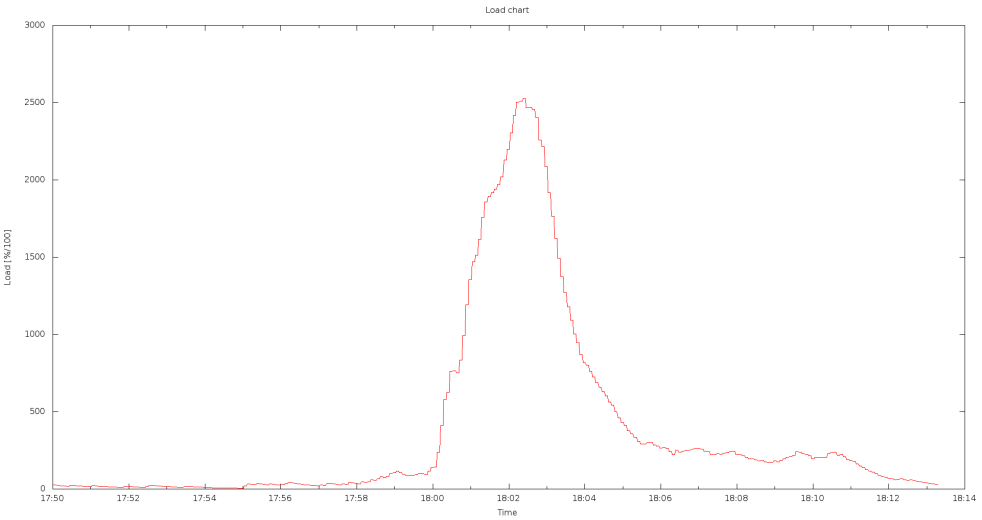
- 40 pacientů
- 4 lékaři
- Délka období 6 měsíců odpovídá 36600 záznamů

Hodnocení testu č. 4
Tento test kombinuje práci 4 lékařů, kteří kontrolují pacienty a zároveň 40 pacientů, kteří posílají svá data do webové aplikace a dalších 40 pacientů, kteří si procházejí své profilové stránky. Po třech minutách od spuštění testu lze vidět na obrázku 12.37 dosažení maximálního zatížení 25% testovaného systému. Hodnota zatížení by nejspíš pokračovala i výše, ale došlo k vyčerpání volné paměti, těsně po přijetí velkého objemu dat na serveru, z glukometrů. Jelikož tento testovací scénář je kombinací tří předchozích testovacích scénářů, můžeme rozdělit výsledky testu pro jednotlivé testovací scénáře č. 1, 2 a 3.
Ke scénáři č. 1 přikládám obrázek 12.41, kde vidíme v počáteční části testu vysoký počet „hitů“ za sekundu způsobený souběžnou prací 40 lékařů v aplikaci. Průměrná doba odezvy požadavku u tohoto scénáře jsou 3 vteřiny. Další

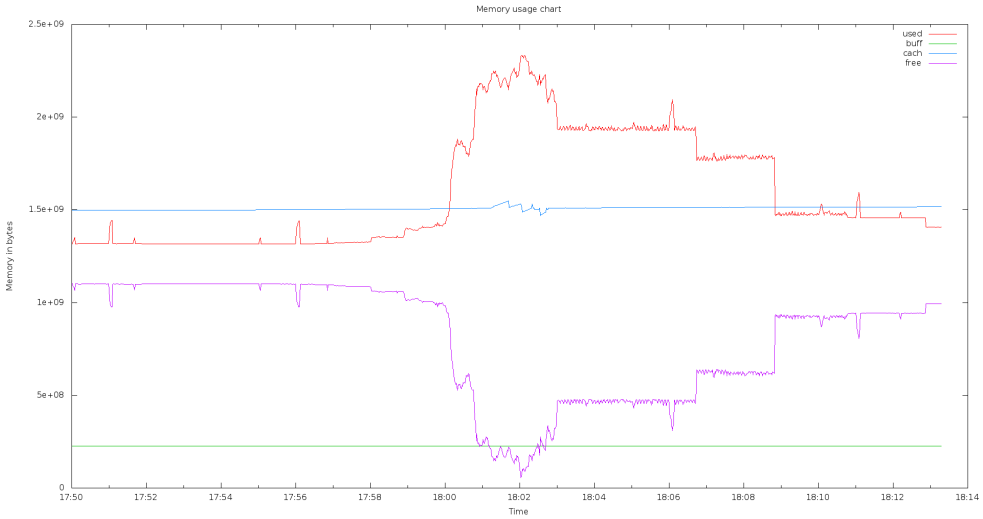
výstupy tohoto scénáře naleznete v příloze v adresáři src/tests. Obrázek 12.42 zobrazuje počet „hitů“ za vteřinu pro scénář č. 2. Po počáteční nečinnosti, která byla nejspíše způsobena přetíženým generátorem požadavků, dochází ke značnému nárůstu požadavků na serveru, který dokáže vyřizovat požadavky s 15 vteřinovou odezvou. Druhá polovina průběhu testu je již s maximem 18 „hitů“ za vteřinu mnohem klidnější. Průběh testu scénáře č. 3 je klidný s postupným zvyšováním zátěže na testovaný systém, viz 12.43



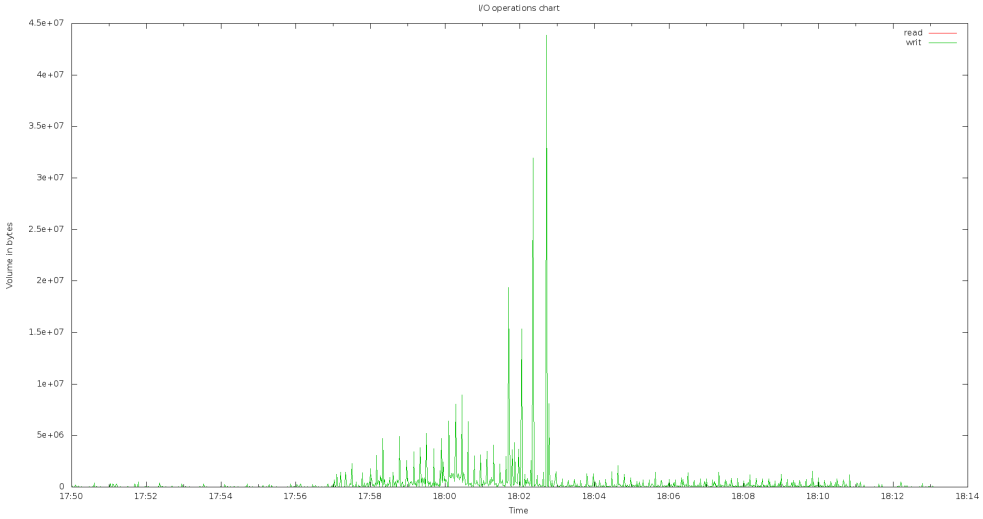
Obrázek 12.36: Test 4: Počet požadavků vytvořený virtuálními uživateli za vteřinu



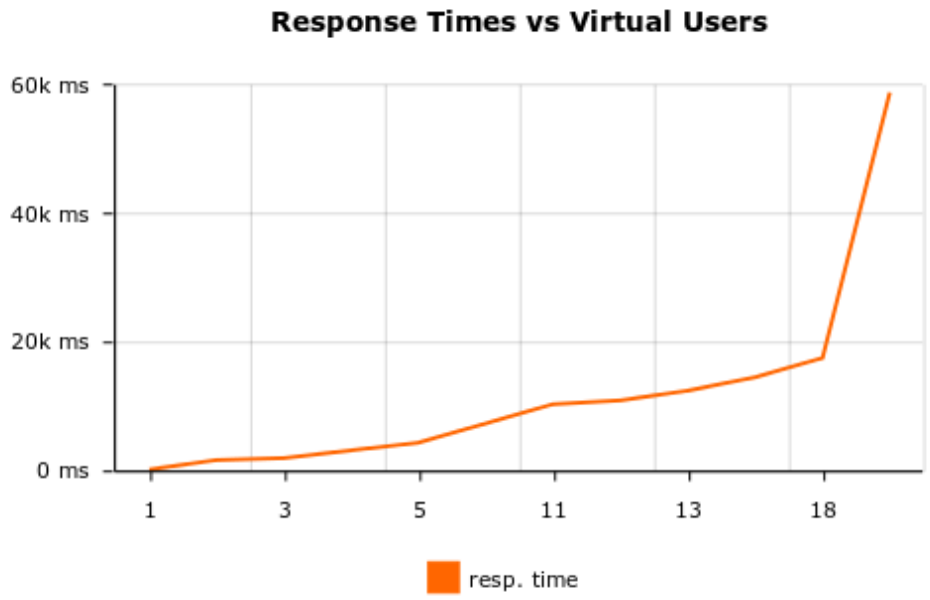
Obrázek 12.37: Test 4: Graf zátěže testovaného serveru



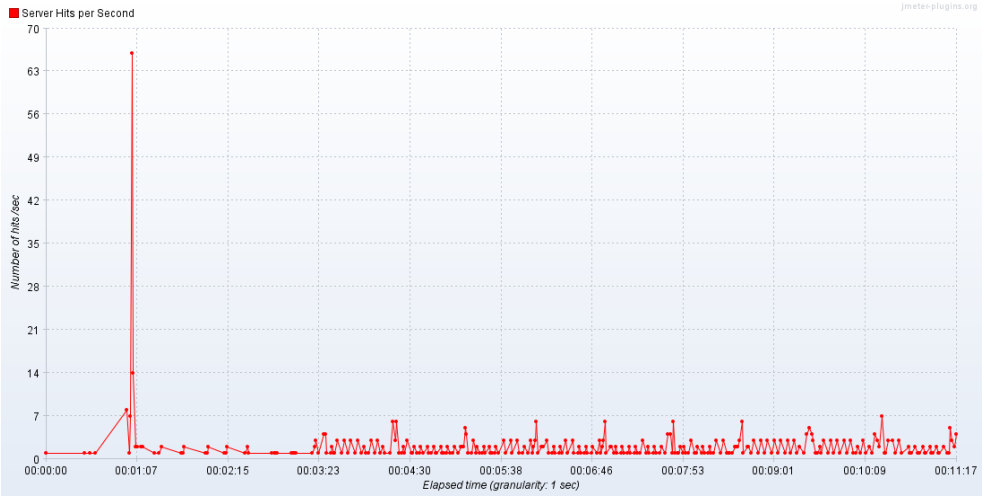
Obrázek 12.38: Test 4: Graf využití operační paměti



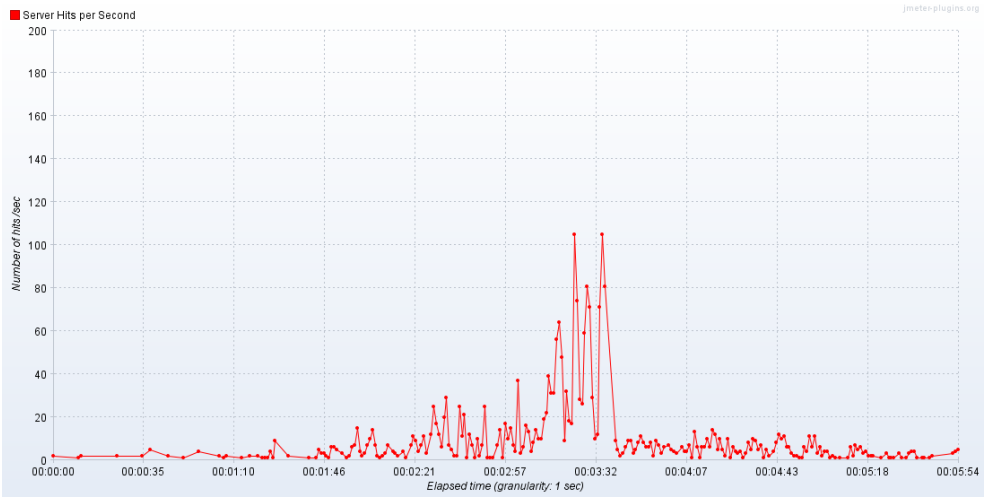
Obrázek 12.39: Test 4: Graf vstupně výstupních operacích na disku



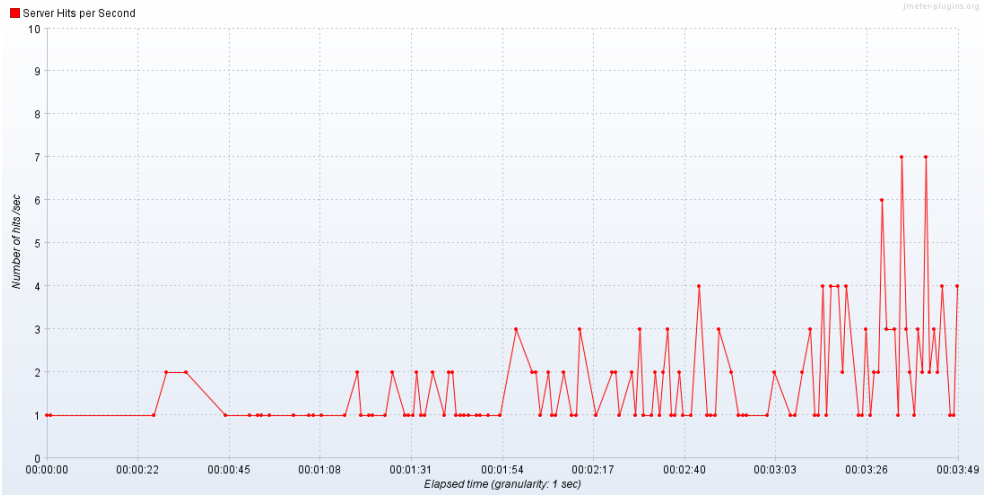
Obrázek 12.40: Test 4: Graf závislosti virtuálních uživatelů na době odezvy požadavku



Obrázek 12.41: Test 4: Scénář 1: Počet požadavků vytvořený virtuálními uživateli za vteřinu



Obrázek 12.42: Test 4: Scénář 2: Počet požadavků vytvořený virtuálními uživateli za vteřinu

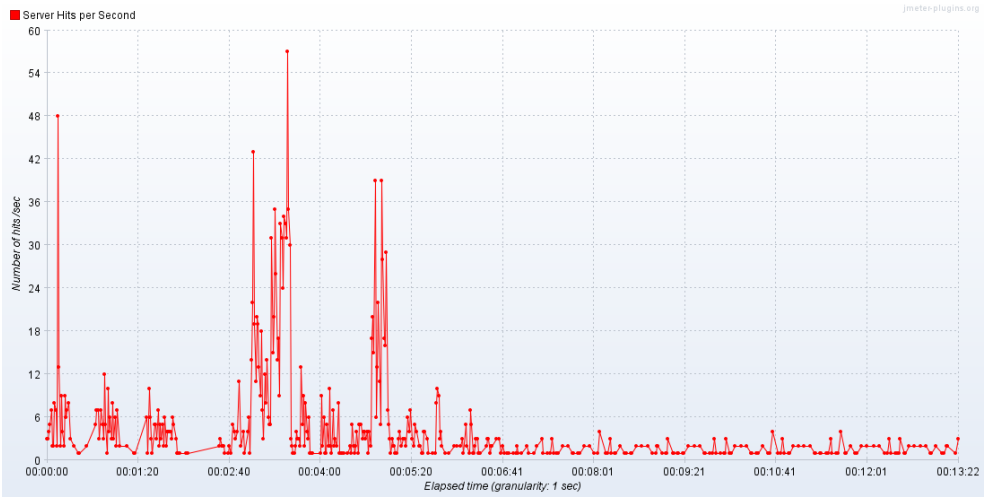


Obrázek 12.43: Test 4: Scénář 3: Počet požadavků vytvořený virtuálními uživateli za vteřinu

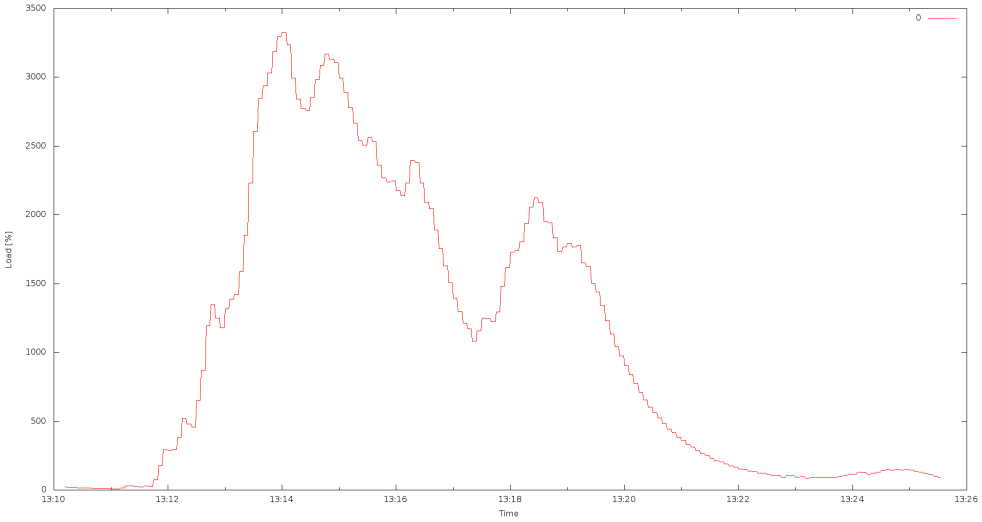
Test 5

- 20 pacientů
- 2 lékaři
- Délka období 6 měsíců odpovídá 18300 záznamů

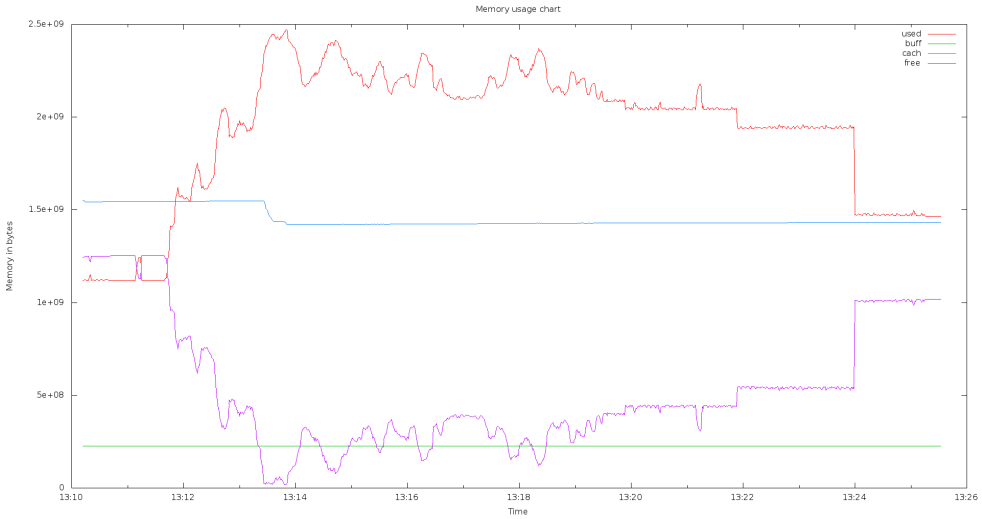
Hodnocení testu č. 5
Délka spuštěného testu je 13 minut, což odpovídá nejdéle běžícímu testovacímu scénáři č. 1. Testovací scénář č. 2 byl spuštěn celkem pětkrát tak, abychom dosáhli delší zátěže po dobu 4 minut. Pětkrát byl také spuštěn testovací scénář č. 3. Testovací scénář č. 2 vykazuje nejdelší dobu odpovědí na požadavky, a to průměrně přes 12 sekund. Způsobeno je to především prudkým nárůstem příchozích požadavků na server, kterému dochází volná paměť. Z grafu zátěže testovaného serveru lze vidět prvních 6 minut vysoké zatížení až 35%. Před spuštěním testu bylo k dispozici zhruba 1,25GB operační paměti, kterou testovaný systém vyčerpal již po prvních dvou minutách testu, viz obrázek 12.46. Testovací scénář č. 3 je testem hraniční zátěže viz obrázek 12.49. Ze tří provedených testů nám vyšla průměrná hodnota propustnosti 2,57 požadavků za vteřinu, viz tabulka 12.4. Tento testovací scénář tvoří kombinace tří testovacích scénářů: práce lékaře, práce pacienta a odesílání hodnot z glukometrů. Scénář č. 1 vykazuje po spuštění až 42 příchozích požadavků na server za vteřinu, viz 12.50. Vzápětí se počet příchozích požadavků na server sníží na hranici jednoho požadavku za vteřinu po dobu pěti minut. Po této odmlce začne opět přijímat více požadavků za vteřinu. U grafu scénáře č. 2, obrázek 12.51 a scénáře č. 3, obrázek 12.52, vidíme opakované spuštění testovacích skriptů. U scénáře č. 2 vidíme nestabilitu příchozích požadavků serverem.



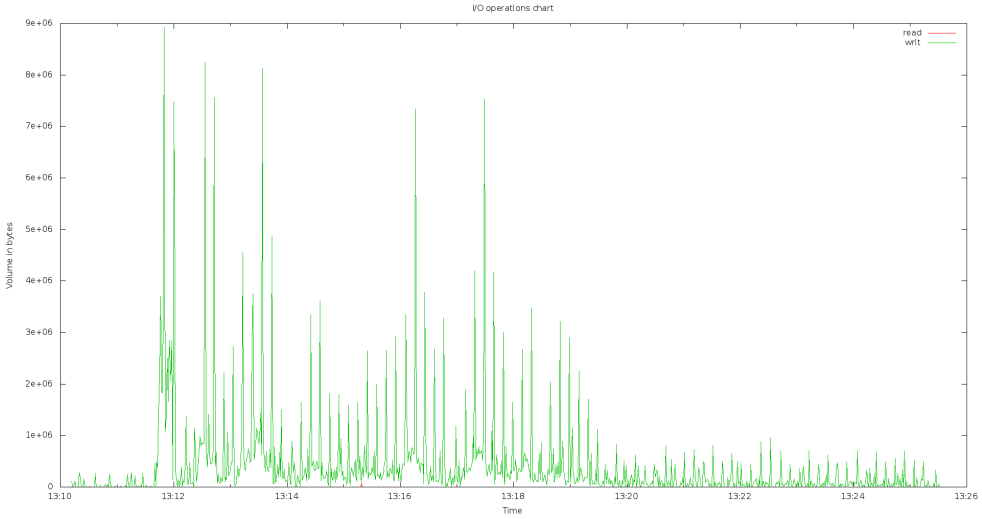
Obrázek 12.44: Test 5: Počet požadavků vytvořený virtuálními uživateli za vteřinu



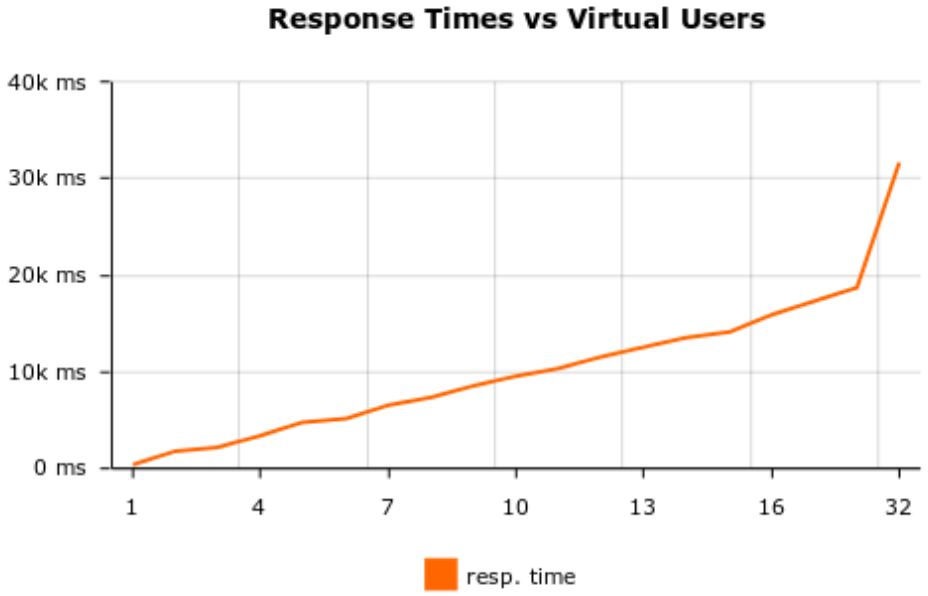
Obrázek 12.45: Test 5: Graf zátěže testovaného serveru v procentech



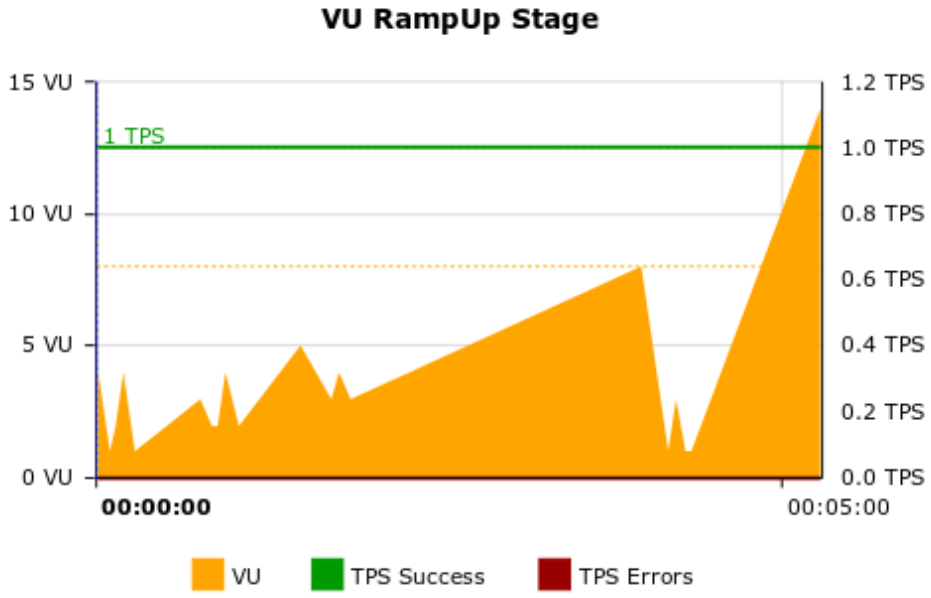
Obrázek 12.46: Test 5: Graf využití operační paměti



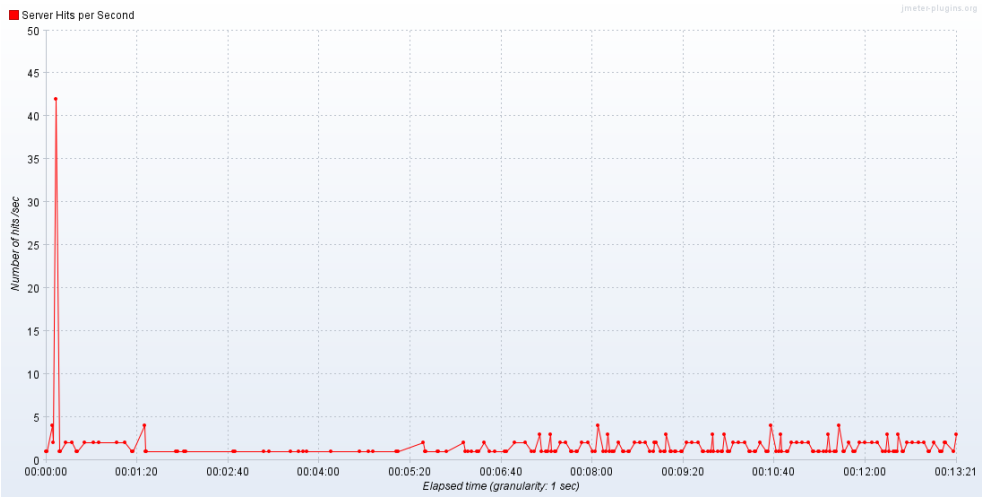
Obrázek 12.47: Test 5: Graf vstupně výstupních operacích na disku



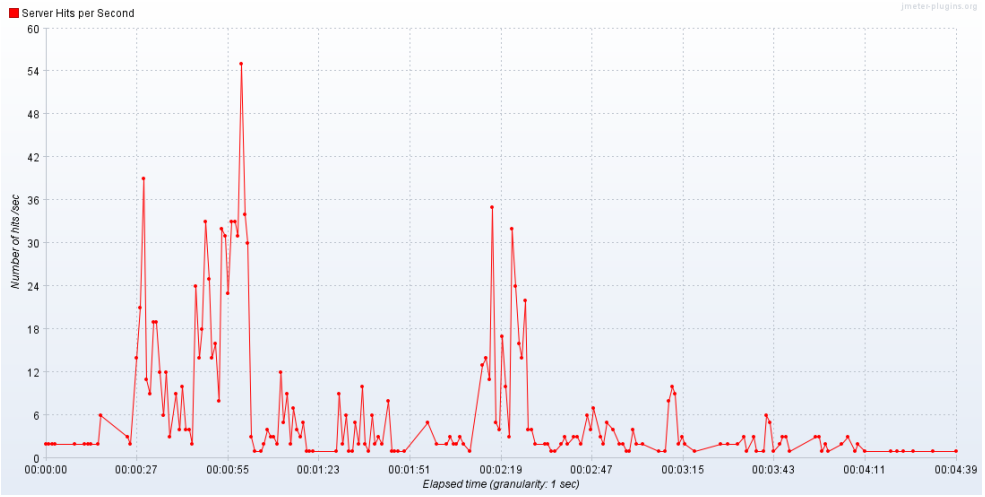
Obrázek 12.48: Test 5: Graf závislosti virtuálních uživatelů na době odezvy požadavku



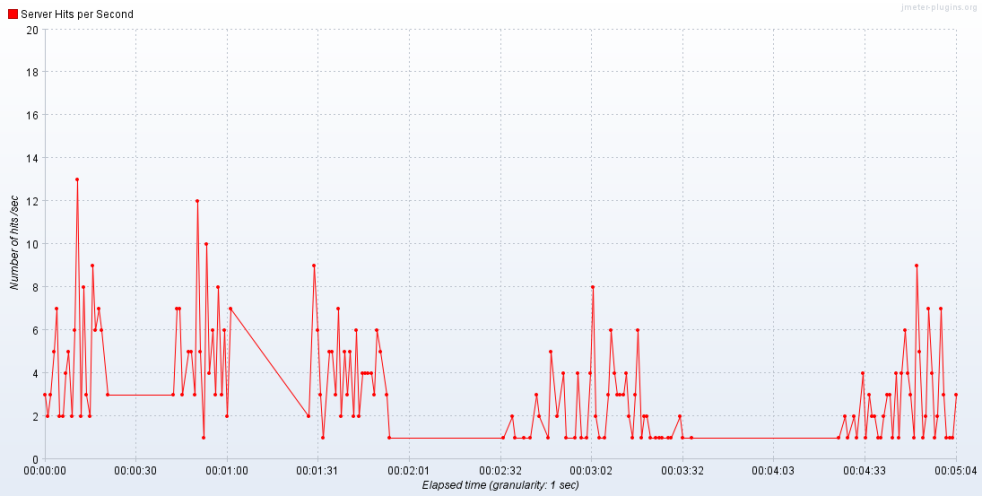
Obrázek 12.49: Test hraniční zátěže testovacího scénáře 3



Obrázek 12.50: Test 5: Scénář 1: Počet požadavků vytvořený virtuálními uživateli za vteřinu



Obrázek 12.51: Test 5: Scénář 2: Počet požadavků vytvořený virtuálními uživateli za vteřinu



Obrázek 12.52: Test 5: Scénář 3: Počet požadavků vytvořený virtuálními uživateli za vteřinu

Shrnutí testovacího scénáře č. 4

Rozložení definovaných testů pro tento scénář jsem zvolil vhodné, protože jsem dokázal naměřit jak velmi nízké hodnoty zatížení, tak i hodnoty překračující přijatelnou hranici odezvy. U těchto testu, které nesplňují přijatelné hranice, je patrný nedostatek volné paměti. Pokud bychom navýšili velikost operační paměti, troufám si říct, že bychom posunuli tuto hranici dále a nynější nepřijatelné výsledky testů by byly rázem akceptovatelné. Ovšem podle stanovených požadavků na aplikaci je tato konfigurace dostačující. V tabulce 12.4 můžete vidět naměřené hodnoty testů čtvrtého scénáře. Jedná se o průměr doby odezvy požadavků, procento neúspěšně vyřízených požadavků a propustnost. Propustnost definujeme jako počet vytvořených HTTP požadavků virtuálními uživateli za vteřinu. V některých případech jsme naměřili počet požadavků za minutu.

	# požad.	**průměr. čas[ms]**	**nevyřízeno[%]**	**propust.[pož./s]**
Test 1	262	1 763,67	0,76	28/min
Test 2	524	3 416,67	0,38	55,70/min
Test 3	786	5 060,00	0,25	1,30
Test 4	1048	8 224,67	1,65	1,50
Test 5	1724	8 758,67	0,80	2,57

Tabulka 12.4: Naměřené výsledky pro testovací scénář č. 4

Závěr

Práce v oblasti informačních technologií mě vždy zajímala a možnost rozšířit si znalosti o realizaci zátěžových testů jsem vřele uvítal. Samotnému provedení zátěžových testů předcházelo dlouhé období, kdy jsem čerpal mnoho informací k tomuto tématu. Postupnými kroky jsem společně s Projektovým manažerem vytvářel celý model této práce. Musím vyzdvihnout konzultace s Projektovým manažerem, které mi pomáhaly najít ten správný směr, jakým se tato práce měla ubírat. Doufám, že jsem alespoň částečně splnil jeho očekávání a mnou odvedená práce bude pro něj přínosem. Tato práce mi poskytla téměř roční zkušenost se zátěžovým testováním webových aplikací. Za tuto dobu jsem zjistil, jak velký význam má provedení zátěžových testů, a tímto bych jej chtěl doporučit všem, kteří o nich zatím jen uvažují. V praxi jsem vytvořil čtyři testovací scénáře, které simulují činnost uživatelů ve webové aplikaci tak, jak ji budou opravdu používat. Vytvořené scénáře obsahují několik zátěžových testů s různou velikostí zátěže. Seznámil jsem se tak s problematikou zátěžových testů jak v obecné tak i praktické formě. Nastudoval jsem druhy zátěžových testů a způsoby jejich provádění. Při vytváření ukázkových dat, nad kterými bylo testování prováděno, jsem uplatnil již dříve získané zkušenosti s vytvářením skriptovacích programů. Díky těmto znalostem jsem výrazně zrychlil přípravu ukázkových dat. Dále jsem si rozšířil praktickou zkušenost s konfigurací webového a databázového serveru z hlediska výkonu. Provedl jsem srovnání nákupu a pronájmu webového serveru, kde se potvrdilo, že pronájem je nejen pohodlnější, ale také levnější variantou. Pro realizaci zátěžového testování jsem si vybral nástroj JMeter, který je doporučován pro svou variabilitu a jednoduchost použití. Za pomoci JMeteru jsem vytvořil testovací scénáře, které můžeme spouštět v libovolném prostředí a testovací scénáře lze tak snadno přenášet. V práci jsou uvedeny všechny spuštěné testy a u vybraných testů je provedeno zhodnocení jejich výsledků. V souhrnném závěru všech testů je uvedeno, zda daný hardware pro požadovaný provoz aplikace je dostačující a je navrženo doporučení pro provoz aplikace, tedy zda je potřeba upravit hardwarovou konfiguraci, nebo se podívat na možnou optimalizaci sa-

motné aplikace. Zjištěná maximální zátěž na testovaný systém, která splňuje požadavky použití aplikace, nejlépe splňuje test č. 3 u testovacího scénáře č. 4, tedy práce 3 lékařů, 30 pacientů v aplikaci a odesílání naměřených hodnot z 30 glukometrů. Za dobu zhruba 12 minut testovaný systém dokázal obsloužit 786 příchozích požadavků zátěžového testu, což je zhruba 3200 obsloužených požadavků za hodinu. Tato hodnota téměř pětkrát překračuje minimální nároky na testovaný systém. Možné zlepšení této práce vidím v rozložení objemu generovaných virtuálních uživatelů mezi více generátorů této zátěže. Toto rozložení by mělo být řízeno z kontroléru, který pouze dohlíží na průběh testování.

Použité zdroje

[1] s.r.o., W.: Serverhosting::Web4U. http://www.web4u.cz/cs/serverhosting/managed-servery, cited April 2015.

[2] Killelea, P.: *Web Performance Tuning: speeding up the web*. "O'Reilly Media, Inc.", 2002.

[3] Foundation, T. A. S.: Apache Performance tuning. http://httpd.apache.org/docs/2.2/misc/perf-tuning.html, cited April 2015.

[4] Foundation, T. A. S.: Apache MPM Common Directives. http://httpd.apache.org/docs/2.2/mod/mpm_common.html, cited April 2015.

[5] and/or its affiliates, O. C.: MySQL 5.5 Reference Manual. https://dev.mysql.com/doc/refman/5.5/en/index.html, cited April 2015.

[6] Palomäki, J.; aj.: Web Application Performance Testing. 2010.

[7] Halili, E. H.: *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing Ltd, 2008.

[8] Foundation, T. A. S.: Apache JMeter - User's Manual. http://jakarta.apache.org/jmeter/usermanual/, cited April 2015.

[9] Sýkora Jiří, S. T.: Analýza pro výkonnostní a zátěžové testy. http://www.systemonline.cz/sprava-it/vykonnostni-testy-podnikovych-aplikaci-2-dil.htm, cited April 2015.

[10] Subraya, B.: *Integrated Approach to Web Performance Testing: A Practitioner's Guide: A Practitioner's Guide*. IGI Global, 2006.

[11] Blazemeter.com: Loadosophia.org. http://loadosophia.org/, cited April 2015.

Příloha **A**

Seznam použitých zkratek

BASH Bourne again shell

CSS Cascading style sheets

FTP File transfer protokol

GIF Graphics interchange format

GUI Graphical user interface

HTTP Hypertext transfer protokol

IDE Integrated development evironment

JDBC Java database connectivity

JMS Java messaging services

JPG Joint Photographic Group

JS Javascript

LDAP Lightweight directory access protocol

NAS Network attached storage

RAM Random access memory

SLA Service-level agreement

SQL Structured query language

TCP Transmission control protocol

URL Uniform resource locator

XML Extensible markup language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `tests` ..... adresář obsahující všechny zátěžové testy
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF