

Masarykova univerzita v Brně
Fakulta informatiky

**Metody distribuce masívní databázové zátěže na
produkčním informačním systému**

Diplomová práce

Duben 2003

Bc. Miroslav Křipač

Zadání:

Nastudujte a zhodnoťte známé metody distribuce zátěže mezi aplikační a databázové servery produkčních informačních systémů. Eventuálně připravte sadu testů pro ověření a zhodnocení možných metod. Na základě zkušeností získaných z provozu informačního systému školy navrhnete další efektivní metodu nebo metody. Tyto zhodnoťte a implementujte do pokusného provozu.

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Rád bych poděkoval svému vedoucímu za cenné rady a podporu věnovanou při tvorbě této práce a celému Vývojovému týmu Informačního systému Masarykovy univerzity v Brně za podporu a poskytnutí zázemí.

Shrnutí

Tato práce se zabývá problematikou masívní zátěže, ke které může při provozu rozsáhlých informačních systémů docházet. Úvodní dvě kapitoly (2 a 3) popisují a hodnotí známé metody distribuce zátěže a vliv architektury systému na jeho zatížení. Kapitola 4 se věnuje autorem navržené metodě distribuce zátěže pomocí databázových klastrů realizovaných nad síťovým sdíleným diskovým subsystémem a detailně popisuje způsob jejího zavedení. Kapitola 5 vysvětluje jednotlivé metody testování zátěže, které byly použity v pokusném provozu. Ten je popsán v 6. kapitole. Závěrečná kapitola obsahuje zhodnocení použité metody a jejího přínosu pro informační systém školy. Práce je doplněna o přílohu, ve které jsou uvedeny grafy ilustrující dosažené výsledky.

Klíčová slova

informační systém, architektura systému, distribuce zátěže, testování zátěže, Linux Network Block Device, Oracle Real Application Clusters

Obsah

1 Úvod	6
2 Architektura webových informačních systémů	8
2.1 Vliv jednotlivých částí systému na distribuci zátěže	9
2.1.1 Webový klient	9
2.1.2 Aplikační server	10
2.1.3 Databázový server	12
2.2 Současná podoba Informačního systému Masarykovy univerzity . .	14
2.2.1 Aplikační část	15
2.2.2 Databázová část	15
3 Obecné metody distribuce zátěže	17
3.1 Nepřímé metody	17
3.1.1 Vliv návrhu aplikací na distribuci zátěže	17
3.1.2 Uměle vynucená zátěž	18
3.2 Vyrovnávací paměti	19
3.2.1 TTL Cache	20
3.2.2 Test Cache	20
3.2.3 Push Cache	20
3.3 Řízená klasifikace požadavků	21
3.3.1 Řízené předbíhání	21
3.3.2 Řízené zpomalení	21
3.4 Distribuce databázové zátěže	22
4 Řešení sdíleného diskového subsystému	24
4.1 Server síťového blokového zařízení	26
4.1.1 Kompilace programu	27
4.1.2 Konfigurace a běh serveru	28
4.2 Klient síťového blokového zařízení	30
4.2.1 Konfigurace síťového rozhraní uzlu	30
4.2.2 Kompilace, instalace a běh klienta	31
4.2.3 Konfigurace v prostředí databázového klastru	32
5 Metody testování zátěže	36

5.1	Klasické metody testování	36
5.2	Testování zátěže simulací reálného provozu	37
5.3	Techniky měření zátěže systému	39
5.3.1	Operační systém Linux	39
5.3.2	Operační systém SunOS/Solaris	40
6	Implementace prototypového řešení	42
6.1	Architektura prototypového řešení	42
6.2	Výsledky pokusného provozu	43
7	Závěr	45
8	Literatura a zdroje	47
9	Příloha: Výsledky testů	48

1 Úvod

V současné době dochází vlivem rozvoje informačních a komunikačních technologií k velkému rozmachu systémů poskytujících informace širokému spektru uživatelů. Zejména díky celosvětové síti Internet a webovým službám se tak informace stávají nejen snadněji zpracovatelné, ale také dostupnější v čase i prostoru. Zatímco donedávna mohl být hlavní cíl informačních systému zefektivnit proces zpracování informací, nyní je jejich úkolem navíc zpřístupnit informace doslova komukoliv, kdykoliv a kamkoliv. Přístup k informacím poskytovaným těmito systémy tedy již není omezen na určitý počet přístupových míst, které jsou obsluhovány pouze odbornými pracovníky v omezenou dobu, což dále zvyšuje efektivitu a snižuje nutnost podávat informace zprostředkovaně.

Takovéto požadavky však kladou na informační systémy řadu nových nároků. Zatímco dříve stačilo udržovat v systému pouze omezený počet klientských aplikací, které tvořily jedinou možnost komunikace mezi koncovým uživatelem a systémem, nyní je potřeba poskytnout (a zejména udržovat) klientskou aplikaci každému uživateli, který se systémem pracuje. Omezení počtu klientských stanic však mělo i další efekt na celkovou koncepci systému, neboť v případě zvýšeného počtu požadavků na systém, byl celkový počet aktuálně zpracovávaných úloh v systému vždy omezen součtem maximálních kapacit klientských aplikací, které byly schopny dohromady vytvořit pouze omezenou zátěž. Zbytek systému tak mohl být snadno nastaven na tento počet s tím, že případné zvýšení požadavků na systém by bylo fyzicky omezeno přístupem ke klientským stanicím. V případě otevření systému všem potencionálním uživatelům však nelze snadno stanovit míru jeho zvládnutelné zátěže (počet najednou obslužených uživatelů), neboť při přílišném zvýšení kapacity systému (které je obvykle možné za použití rychlejších strojů) by se systém mohl stát velmi neefektivním. Zbytečně drahé vybavení by sice umožnilo zvládat i náročné zátěžové špičky, většinu času by však systém pravděpodobně zůstal téměř úplně nevyužitý. Jako poslední důležitý důsledek potřeby zvýšení dostupnosti informací je rovněž nutnost poskytovat je okamžitě. Údaje zadané jedním uživatelem se ostatním musí projevit hned. Nelze tedy zpracování údajů odložit na dobu, kdy systém nebude pro uživatele dostupný a teprve poté hromadné zpracování provést. S tím souvisí také v poslední době stále více žádaná nepřetržitá dostupnost systému.

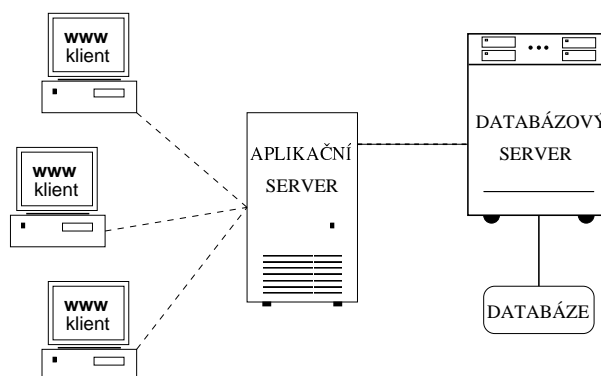
Právě výše zmíněné problémy s ochranou proti masívní zátěži na produkčním informačním systému byly námětem pro tuto diplomovou práci. Cílem je vytvořit systém, který by umožňoval distribuci masívní zátěže v provozních špičkách za současného zachování optimálního využití všech prostředí v běžném provozu. Jako hlavní motivace sloužily pro tuto práci zkušenosti s provozem a vývojem In-

formačního systému Masarykovy univerzity v Brně (IS MU), který je v současné době provozován jako systém osobní administrativy umožňující všem členům akademické obce elektronicky zpracovávat téměř celou agendu spojenou se studiem (od elektronického přihlášení ke studiu až po tisk závěrečného diplomu). Přesto mohou být zjištěné závěry užitečné i pro návrháře libovolného webového systému, který ve větší či menší míře umožňuje komplexní zpracování informací.

Důležitou roli při hledání optimálních metod rozvržení zátěže hraje zvolená architektura systému. Samotný vhodný návrh může práci celého systému podstatně zefektivnit. Navíc může vytvořit podmínky pro další softwarové metody rozvržení zátěže. Popis architektury systému, která se v tomto směru osvědčila jako efektivní je zmiňován ve druhé kapitole práce. Možné metody rozvržení zátěže jsou poté diskutovány v kapitole třetí. Celá čtvrtá kapitola se věnuje novému způsobu řešení sdíleného diskového subsystému jako metodě umožňující efektivní distribuci databázové zátěže pomocí síťového spojení. Toto nové řešení jsem ve své práci implementoval do pokusného provozu. Výsledky implementace včetně měření zátěže vlastními metodami popsány v kapitole pět jsem shrnul v šesté kapitole. Závěrečná kapitola se věnuje zhodnocení navržených metod a jejich přínos pro další rozvoj informačního systému školy. Celá práce je v příloze doplněna o grafy prezentující jednotlivé testované veličiny při zatížení a srovnání různých navržených konfigurací se současným stavem systému.

2 Architektura webových informačních systémů

Vhodný návrh architektury je vedle dalších možných způsobů jedním z hlavních faktorů ovlivňujících zátěž systému. Jak již bylo naznačeno v úvodní kapitole, omezíme se v této práci pouze na tzv. webové informační systémy, tedy systémy jejichž hlavní komunikační rozhraní je realizováno pomocí celosvětové sítě Internet na bázi jejich služeb označovaných jako World Wide Web. Výhody těchto systému jsou právě v jejich široké dostupnosti, která je snadno realizovatelná.



Obrázek 1: Schéma architektury obecného webového informačního systému

Architektura webových informačních systému je obvykle složena ze tří základních částí. Na uživatelské úrovni slouží pro komunikaci koncových uživatelů se systémem klientská aplikace (webový klient), která je v tomto případě představována obvykle internetovým prohlížečem. Ten tvoří rozhraní mezi uživatelskými dotazy a jejich výsledky a druhou částí – aplikačním serverem. Aplikační server (v našem pojetí zahrnuje i často rozšířené rozdělení na aplikační a webový server) je platforma pro samotné provádění aplikací, které na základě prohlížečem poskytnutých požadavků provedou potřebné operace nad daty uložené v systému a prohlížeči pak vrátí zpět jejich výsledek. Ten je poté klientem převeden do podoby snadno čitelné pro koncového uživatele. Výhodou tohoto řešení je jednak udržování vlastního kódu aplikací v aplikačním serveru (odpadá tedy distribuce aplikace až k uživateli) a zároveň provedení prezentační části až při konečné interpretaci uživatelem. Aplikační server se tak nemusí výslednou prezentací zatěžovat. Její popis pouze předá ve vhodně zvoleném jazyce (nejčastěji HTML). Poslední částí systému je databázový server, který slouží aplikacím jako jednotné rozhraní pro přístup k datům. Umožňuje navíc jejich snadnou a efektivní správu.

Samotné rozvržení zátěže ve všech částech systému může vyřešit celou řadu výkonostních problémů. Účelově navržený systém navíc poskytuje potřebné pro-

středí pro zapojení dalších metod ovlivňujících zatížení. Hlavním cílem tohoto snažení však je navrhnout systém na základě předem přesně stanoveného úzkého místa nárazové zátěže. Častou příčinou problémů s nevhodným rozložením zátěže na produkčním systému totiž je nevědomělé zatížení jinak dostatečně výkonných prvků. Pokud naopak celý návrh provádíme se znalostí úzkého místa, můžeme případné přetížení jiných částí snadněji detekovat jako problém a rychleji vyřešit, což může samo o sobě poskytnout dočasně dostatečné řešení. Typickým příkladem špatného navržení jednotlivých subsystémů může být například neúměrné zatížení jednoho z datových disků na úkor ostatních, které zůstávají nevyužity. Rozložením takové zátěže pak obvykle dojde k přenesení úzkého místa na jinou komponentu systému navíc za současného zvýšení propustnosti bez jakékoliv další investice do rychlejších disků. Snahou návrháře systému tedy je mít při dosažení maximální propustnosti systému (špičkové zátěži) nejlépe jedno nejméně snadno zrychlitelné místo, které bude v takovém případě nejvíce přetížené. To ovšem neznamená, že ostatní části budou v tuto dobu nevyužité. Pouze jejich případné posílení (které lze provést snadněji) by již na zrychlení systému nemělo vliv. Při dodržení tohoto principu pak nejsnadněji dosáhneme nejefektivnějšího využití všech zdrojů.

Protože diskuse jednotlivých přístupů k řešení konkrétních problémů implementace by si jistě zasloužila podrobnější popis nad rámec rozsahu této práce, zaměříme se v dalším popisu pouze na ty aspekty hlavních subsystémů, které mají přímý vliv na distribuci zátěže. Detailně se implementací systémů zabývá [1].

2.1 Vliv jednotlivých částí systému na distribuci zátěže

2.1.1 Webový klient

Webový klient slouží mimo jiné jako interpret hypertextového značkovacího jazyka HTML. Jeho úkolem je za pomoci daného protokolu získat od webového serveru požadovaný hypertextový dokument a jeho text za pomoci vložených značek převést do uživatelsky přívětivé podoby. Komunikace mezi webovým serverem a klientem se tedy omezuje pouze na zaslání dokumentů, které jsou (v podstatě jediným) produktem webových serverů. Tyto servery v případě informačních systémů dané dokumenty vytváří dynamicky jako aktuální výsledek operací nad požadavky zadanými uživatelem. To má poměrně značný vliv na architekturu aplikací, neboť komunikace pomocí těchto dokumentů je obecně bezstavová – systém generuje jednotlivé dokumenty bez jakékoliv návaznosti. Možným řešením tohoto problému může být interpretace části kódu na straně webového klienta. Tuto funkci již řada klientů podporuje. Současný vývoj však ukazuje omezení, která toto řešení přináší – zejména se jedná opět o nutnost distribuce kódu k uživateli nebo problém kompatibility jednotlivých kódů (ať už zdrojových nebo binárních) s různými klienty. Kompatibilita se stává stále důležitějším měřítkem kvality systému a to zejména s rozvojem mobilních zařízení sloužících jako webový klient.

Z toho důvodu se budeme zabývat pouze běžnými klienty, jejichž jediným úkolem je prezentace přijatých dokumentů uživateli. Na v současné době dostupném hardwarovém vybavení však tato činnost nevytváří velké zatížení, což dokazují i řádově méně výkonnostně vybavená mobilní zařízení. Rychle zpracovaná úloha poté netvoří zátěž ani na straně serveru, neboť jím dodaný výsledek dokáže přijmout v co nejkratším čase a tím uvolnit jeho prostředky pro zpracování dalších požadavků co nejdříve. Rovněž kapacita síťového spojení v oblasti současných informačních systémů nepůsobí problém mezi klientem a webovým serverem. Drtivá většina doposud poskytovaných informací je v textové podobě a ani přívětivější grafické uživatelské rozhraní nemusí linky či systém příliš zatížit.

Zdánlivě se proto může webový klient jevit jako bezproblémová složka systému. Běžný požadavek zadaný klientem do systému je rychle předán a neprodleně zpracován, čímž nepůsobí žádné výrazné zatížení. Ze základní vlastnosti webových technologií – dostupnosti, však vyplývá i hlavní vliv klienta na celkové zatížení systému. Současným problémem zátěže totiž není rychlost zpracování daného požadavku, ale míra propustnosti takových požadavků, tedy počet současně provedených operací na všech úrovních zpracování. Přestože samotné zpracování jednoho požadavku webovým klientem nemá na výkon systému téměř žádný vliv, současný přístup stovek až tisíců uživatelů jej ovlivní velmi zásadně. V uvažování nad vlivem architektury dalších částí systému na výkon systému se tak můžeme omezit pouze na zátěž způsobenou současným zpracováním velkého množství úloh.

2.1.2 Aplikační server

Pod pojmem aplikační server se v současném světě rozmanitých technologií webových systému označuje celá řada různých řešení. Základní úlohou této složky, ať už je řešená jakýmkoliv způsobem, je vytvoření prostoru pro provádění aplikací uchovávajících celou logiku výpočtu systému. Obecně je tedy úkolem aplikačního serveru převzít požadavek předaný webovým klientem, zpracovat jeho jednotlivé hodnoty jako vstupní data, nad kterými se za pomoci již uložených dat v systému provedou potřebné operace. Výsledkem je, jak už bylo naznačeno hypertextový dokument předaný zpět prohlížeči k interpretaci. Logicky je tedy činnost aplikačního serveru rozdělena na dvě části – zpracování vstupních dat za součinnosti uložených údajů v systému a jejich následná prezentace do podoby dokumentu ve značkovacím jazyce. Z tohoto důvodu se zavádí v mnoha zejména moderních přístupech k tvorbě webových aplikací jasné rozdělení mezi výpočetní a presentační částí. Toto rozdělení má veliký vliv na návrh a efektivní správu jednotlivých aplikací, neboť výsledný systém pak je složen z více specificky zaměřených programů. Program provádějící samotný výpočet se tak nemusí starat o správnou syntaxi výsledného dokumentu, naopak presentační složka má již předpřipravená všechna data a nemusí se tedy zabírat jednotlivými výpočty nebo komunikací s databázovým serverem. V poslední době se k tomuto účelu kromě tradičních objektových přístupů komunikace mezi jednotlivými částmi stále častěji uplatňují technologie

na bázi transformace dat uložených v různých typech značkovacích dokumentů XML.

Toto logické rozložení provádění výpočtu vede ke snadné implementaci jeho fyzického rozdělení, kdy jsou obě části prováděny na různých počítačích. To může být vhodným prostředkem pro distribuci takzvané vertikální zátěže neboli zátěže způsobené jedním požadavkem při jeho provádění. Presentace do podoby hypertextového dokumentu se totiž může začít na jiném stroji provádět již ve chvíli kdy není dokončen jeho samotný výpočet, což může celý proces zpracování jednoho požadavku urychlit. Na takzvanou horizontální zátěž, tedy zátěž způsobenou na jedné úrovni výpočtu více prováděnými úlohami najednou, však toto rozdělení nemá vliv. Vzhledem k tomu, že naším hlavním cílem je právě tato horizontální zátěž, můžeme si dovolit různé techniky aplikačního zpracování požadavků a jejich distribuci mezi webový (prezentační) a aplikační (obsahující veškerou logiku výpočtu) server zanedbat a počítat s aplikačním serverem jako s jednou autonomní součástí komunikující s webovým klientem na jedné straně a s databázovým serverem na straně druhé. Větší význam tohoto rozdělení je v koncepci návrhu aplikací, nikoliv v návrhu architektury zátěžově odolného systému.

Narozdíl od webových klientů, jejichž výpočet je prováděn až na jednotlivých stanicích je výpočet aplikačního serveru soustředěn výhradně do jednoho centra systému. To má velký význam zejména na správu aplikací, která je omezena pouze na toto centrum a případné změny v aplikacích není nutné distribuovat uživatelům. Zároveň však velký počet požadavků, které zvládne velký počet uživatelských stanic najednou vyprodukovat, musí tento aplikační server zpracovat. Z hlediska distribuce horizontální zátěže je tedy aplikační server oproti webovému klientovi velmi důležitý. Velkou výhodou aplikačního zpracování požadavků však je jejich vzájemná nezávislost. Dva uživatelské dotazy jsou tak snadno zpracovatelné současně a to na několika úrovních souběžného zpracování. První úroveň tvoří možnost současného zpracování těchto požadavků v multiprocesovém systému jednoprocessorového počítače. Velká rezie na komunikaci mezi aplikačním serverem a ostatními částmi systému dovoluje i na jednoprocessorovém stroji provádět souběžně výpočet více úloh, neboť jednotlivé části výpočtu jsou prokládány operacemi nezatěžující procesor, který je tak uvolněn pro ostatní výpočty. Druhou úrovní souběžného zpracování aplikací je přirozené rozšíření této možnosti o multiprocesorový systém, čímž se dále zvyšuje počet možných současně prováděných operací na aplikačním serveru. Výhodu vzájemné nezávislosti aplikačních výpočtů však lze využít ještě efektivněji v jejich rozdělení na více jednoprocessorových strojů, neboť nevyžadují žádnou vzájemnou komunikaci, která by jinak měla vliv na výkonnost současně běžících procesů. Tuto technologii tzv. klastrování lze proto pro aplikační servery s výhodou snadno využít, což má jeden ze základních vlivů na distribuci zátěže. Zároveň pomáhá zvyšovat další důležitou vlastnost moderních systému – vysokou dostupnost. Výpadek jednoho uzlu klastru (jednoho ze současně zpracovávajících strojů či jeho části) totiž nezpůsobí výpadek celého systému, jak by tomu mohlo být u jednoho (byť víceprocesorového) stroje.

Tvrzení o vzájemné nezávislosti jednotlivých výpočtů aplikačního serveru však není úplně přesné. V obecném systému totiž snadno může dojít k současnému vyslání dvou požadavků různých uživatelů, které mění stejné údaje již uložené v systému. V takovém případě by tedy muselo z důvodu zachování integrity dat v systému docházet ke vzájemné komunikaci mezi jednotlivými aplikacemi. Závěr, který jsme o distribuci horizontální zátěže na aplikační části systému vyslovili, však lze dodržet při použití moderních metod ve třetí hlavní části systému – databázovém serveru. Jedním z jeho hlavních úkolů je totiž zajistit nejen jednoduché rozhraní pro přístup k uloženým a ukládaným datům, ale i integritu těchto dat při současném přístupu více uživatelů. Tím, že jednotlivé procesy aplikačního serveru (nebo lépe jednotlivých uzlů klastru) nepracují přímo s daty, ale s databázovým serverem umožňujícím takovouto souběžnou práci, je dosažena skutečná nezávislost jednotlivých částí. Tím je podmínka pro dosažení distribuce horizontální zátěže aplikačního serveru pomocí techniky klastrování splněna a problém současného přístupu klientů je tedy přesunut ve směru zpracování požadavků na další část systému – databázový server.

2.1.3 Databázový server

Úkolem databázového serveru, jak sám název napovídá, je komplexní správa dat. Pod tímto obecným označením se skrývá zejména jejich efektivní uložení za současné možnosti rychlého vyhledání jejich relevantní části. V pozadí nezůstává ani možnost ochrany dat před jejich porušením při případném výpadku systému, jejich zálohování a obnova. Z pohledu distribuce horizontální zátěže systému způsobené velkým množstvím současně kladených požadavků je však nejdůležitější schopnost vytvořit přístupujícím aplikacím prostředí umožňující jejich navzájem nezávislou práci. Toho se dosahuje pomocí různých technik databázového zpracování. Jednoduše řečeno v případě, kdy dojde ke konfliktu v přístupu k datům, databázový server pozdrží všechny konfliktní aplikace na úkor jedné, jejíž operace provede přednostně. Návrh takového databázového systému je jistě samotným námětem pro další možný vývoj. Pro naše účely se můžeme spokojit se současnými výsledky tohoto vývoje v podobě komerčně dostupných řešení, která toto souběžné zpracování efektivně zajišťují. Vzhledem k úkolům, které musí databázový server zajišťovat, je důležité, aby spolu jeho části, přestože zpracování požadavků probíhá souběžně, navzájem komunikovaly. Rovněž data, se kterými pracují a která sdílí, je nutné udržovat co nejbližně jednotlivým zpracovávajícím procesům, tedy nejlépe ve společné sdílené paměti obsluhované souběžně běžícími procesy multiprocesorového systému. Při tradičním přístupu se proto (oproti aplikačnímu serveru) pro realizaci databázové části systému využívají multiprocesorové systémy.

Po zkušenostech s technikou klastrování v aplikační části se však výrobci databázových programů a návrháři informačních systémů snaží zapojit tuto techniku i na úroveň databázového serveru. Vznikají tak systémy umožňující distribuované zpracování dat na oddělených strojích. Vzhledem k tomu, že výkon jednotlivých multiprocesorových serverů lze zvýšit u většiny reálných nasazení poměrně snadno

(byť nákladně) navýšením počtu procesorů nebo zvýšením jejich rychlosti, která stále narůstá, primárním důvodem pro tvorbu databázových klastrů je dosažení vyšší dostupnosti systému, kterého nelze jednoduše dosáhnout jinak než právě technikou klastrování. Tím je jasně dán i trend současného návrhu architektury databázových klastrů (popsný v [2]), které od principu distribuce výpočetního výkonu včetně části dat na jednotlivé uzly klastru, přechází k modelu sdílení všech dat všemi výpočetními uzly. Takováto architektura umožňuje zachovat činnost systému i při výpadku jednoho z uzlů, neboť nedostupný uzel nemá uschována žádná data, která by se stala nedostupná. Rozvoj tohoto řešení je navíc podpořen rozvojem diskových subsystému, které umožňují rychlý přístup všem uzlům klastru ke všem datům.

Technika databázového klastrování se však nabízí i jako možná metoda distribuce masivní databázové zátěže, neboť umožní její rozložení na několik více nezávislých částí. Tak jako se nasazení klastru ukazuje jako vhodný způsob distribuce zátěže na aplikační úrovni, mohlo by zavedení databázového klastru buďto samo vyřešit nebo napomoci k řešení problému databázové zátěže. Sestavení databázového klastru je však závislé na pořízení rychlého sdíleného subsystému a často lepšího meziuzlového síťového spojení než je dostačující pro spojení aplikačního a databázového serveru, neboť jednotlivé uzly sdílí konkrétní data a tím v podstatě nahrazují sdílenou paměť klasického multiprocesorového počítače. V neposlední řadě důležitou roli pro zavedení databázového klastru je cena klastrového databázového software, která může být až několikanásobně vyšší. Tyto investice tak nepřímou vedou k nasazení databázových klastru výhradně za účelem zvýšení dostupnosti u systémů kriticky závislých na vysoké dostupnosti u nichž se vyplatí. Pro zvýšení výkonnosti databázového serveru bývá proto často jednodušší investice do rychlejšího hardwarového vybavení klasického řešení.

Zkušenosti z provozu administrativních informačních systémů však ukazují, že drtivá většina uživatelských úloh, které jsou systémem zpracovány, netvoří buďto žádné, nebo pouze minimální operace zápisu. Skutečnost, že většina dat se nemění, umožňuje databázovému serveru i v architektuře klastru jejich uchování v lokálních pamětech jednotlivých uzlů. Vzhledem k tomu, že ani ostatní hromadné operace, které zpracovávají a mění velké množství dat, nepracují s daty velkého objemu (nejčastěji jsou to pouze číselné nebo textové údaje), nedochází ani při velkém počtu současně vznesených požadavků k nadměrnému zatížení diskového subsystému. Důležitým problémem aplikací v těchto systémech je spíše složitý výpočet na vyžádání než zpracování velkého objemu dat v nočních dávkových procesech. Distribuce výpočetní zátěže se tak stává prioritou zatímco nároky na výkon vstupně-výstupních operací nejsou zdaleka tak vysoké. Tato skutečnost vede k hledání nových možností konstrukce databázových klastrů, které by sice nezajistily takový výkon jako v případě opravdových sdílených diskových subsystémů, pro celou řadu řešení by však umožňovaly levnější a zároveň dostatečnou alternativu. Metoda, kterou jsem ve své práci popsal, úspěšně zveřejnil a kterou si v následujících kapitolách představíme, se snaží tento problém vyřešit za použití síťového sdíleného diskového subsystému.

Z výše uvedeného základního popisu architektury jednotlivých částí webového informačního systému a jejich vlivu na distribuci zátěže vyplývá také určení požadovaného úzkého místa nárazové zátěže. Díky technice klastrování je problém rozložení horizontální zátěže přenesen až na databázový server. Zkušenosti ukazují, že při vhodném rozložení všech jeho komponent dochází k největšímu zatížení pouze u procesorů provádějících výpočet databázového stroje. Vzhledem k tomu, že se jedná obvykle o drahé multiprocesorové počítače, jejich případné posílení za účelem zvýšení výpočetního výkonu je většinou spojené s mnohem většími náklady než je tomu u aplikačního serveru, kde technikou klastrování snadno dosáhneme požadovaného výkonu za pomoci většího množství levných jednoprocessorových počítačů. V neposlední řadě je důvodem pro stanovení procesorové zátěže databázového serveru za úzké místo systému cena databázového programového vybavení, která se obvykle odvíjí právě od rychlosti počítače, na kterém je provozováno. Snahou vhodného návrhu systému je rozložit všechny jeho prvky tak, aby při špičkové zátěži došlo k maximálnímu zatížení procesorů databázového subsystému, neboť všechny ostatní části lze posílit snadněji. Cena za zvýšení propustnosti ve špičkové zátěži pak jasně závisí na investicích vložených do systému koupením nových (dalších) procesorů případně licencí databázového programu. Konfigurace ostatních částí by pak měla toto nastavení kopírovat jejich vhodným zatížením ve výkonnostních špičkách, aby nedocházelo ke zbytečnému plýtvání prostředky.

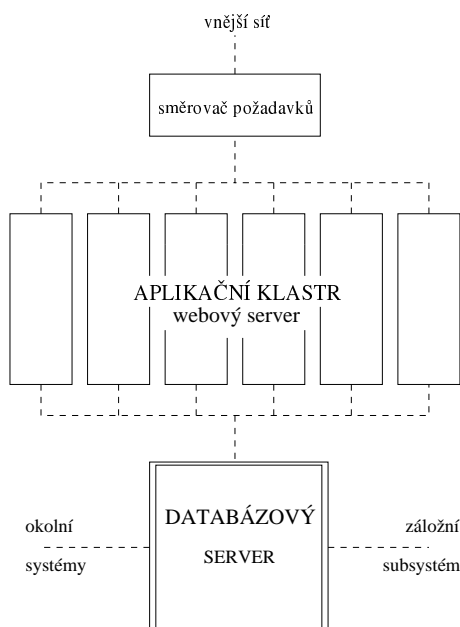
2.2 Současná podoba Informačního systému Masarykovy univerzity

Jako příklad architektury webového informačního systému může posloužit Informační systém Masarykovy univerzity v Brně (IS MU). Jedná se o administrativní systém, který v sobě zahrnuje v podstatě veškerou agendu související se studiem na vysoké škole. Zejména v otázkách evidence studia, ale i obecně slouží tento systém jako univerzální prostředek komunikace mezi studentem, učitelem a vedením školy. Usnadňuje tak běžnou činnost všech akademických pracovníků, čímž pomáhá soustředit jejich pozornost na jejich hlavní záměr – vzdělávání a výzkum.

Informační systém MU v současné době využívá asi 30 000 aktivních uživatelů (zejména tedy studentů, učitelů a dalších administrativních pracovníků), z nichž jich denně k systému přistoupí až 10 000. V provozních špičkách dosahuje systém propustnosti asi 50 000 operací za hodinu, přičemž běžně se systémem pracují stovky uživatelů najednou. Samozřejmostí se stala nepřetržitá dostupnost a možnost přístupu v podstatě z libovolného internetového prohlížeče včetně mobilních zařízení. Informační systém MU byl uveden do provozu v březnu roku 1999 a později úspěšně zaveden na několika dalších vysokých školách v České republice.

Ačkoliv úspěšné nasazení systému a jeho běžné používání v podstatě všemi aktivními uživateli má již nyní pro univerzitu velký přínos, průběžně dochází k dalšímu vývoji nových funkcí a vylepšování stávajících možností systému. Jedním z aktuálních problémů je řešení masivní zátěže způsobené nárazovým přístupem

velkého množství uživatelů. Tato práce se proto snaží přispět k vývoji tohoto systému a stává se tak jeho nedílnou součástí.



Obrázek 2: Schéma architektury IS MU

2.2.1 Aplikační část

Aplikační prostředí IS MU je tvořeno skripty psanými v jazyce Perl a interpretovanými webovým serverem Apache. Jako operační platforma byl během vývoje systému pro aplikační část zvolen operační systém Linux, který je vhodným a oblíbeným nástrojem pro implementaci aplikačních klastrů. Konkrétně je aplikační klastř realizován pomocí technologie Linux Virtual Server, kdy jeden server přijímá všechny požadavky klientů a podle aktuální zátěže je na úrovni síťového spojení předá jednomu z připravených aplikačních serverů. Seznam těchto dostupných serverů je navíc periodicky aktualizován, takže v případě výpadku jednoho z nich dojde automaticky k jeho vyřazení ze seznamu dostupných strojů. V současné době stačí pokrýt běžný provoz šest aplikačních strojů, každý s jedním procesorem typu Athlon XP.

2.2.2 Databázová část

Jako databázový software byl zvolen produkt Oracle Database, který je v současné době absolutní špičkou v oblasti technologie zpracování dat. Ačkoliv i on má své problémy a nedostatky, které se nepříznivě projevují při vývoji a provozu systému, obsahuje v sobě řadu jedinečných součástí, jako je například i výše zmíněná

podpora databázových klastrů. V současné době je databázový systém provozován na čtyřprocesorovém serveru Sun Fire V880 společnosti Sun Microsystems, který zajišťuje plně chod celé databáze. Tato konfigurace je pro běžný provoz systému plně dostačující. Systém se navíc podařilo úspěšně nastavit tak, že při nárazově velkém počtu přistupujících uživatelů dochází k rovnoměrné distribuci zátěže mezi jednotlivé části systému tak, že k omezení provozu dochází právě vlivem přetížení procesorů databázového serveru. Naproti tomu při běžné denní zátěži dochází k zatížení přibližně z 30% – 50% jejich celkové kapacity čemuž také odpovídá dostatečně nízká doba odezvy u většiny aplikací. Občasné přetížení systému způsobené různými důvody však může dobu odezvy podstatně zvýšit. Tato práce se pak snaží přispět k efektivnímu řešení tohoto problému pomocí různých metod popsaných v následující kapitole.

3 Obecné metody distribuce zátěže

Důvodem masívní zátěže produkčního informačního systému, kterou se budeme v této práci zabývat, je, jak již bylo popsáno při zkoumání architektury systému v předchozí kapitole, současné zadání velkého množství požadavků různými uživateli. Nebezpečí takového přetížení hrozí v podstatě každému modernímu systému, který se snaží na jedné straně poskytovat své služby libovolnému svému uživateli, na druhé straně je však limitován svými prostředky, jejichž možnosti jsou omezené a dané obvyklým počtem klientů při běžném zatížení. Ochrana před nárazovou zátěží pomocí zrychlení úzkého místa je tedy pouze dočasným kompromisem mezi možnostmi dalších investic do rychlejších strojů a jejich skutečným využitím. Pro zajištění trvalejší ochrany proti tomuto problému je nutné použít jiných technik distribuce zátěže.

3.1 Nepřímé metody

Prvním typem metod distribuce zátěže jsou takzvané nepřímé metody, jejichž realizace obvykle nevyžaduje technologické změny v architektuře samotného systému či v použití jeho jednotlivých komponent. O to horší však může být jejich skutečné nasazení v reálných podmínkách. Dříve než se jakýkoliv provozovatel systému začne zabývat možnostmi zlepšení jeho průchodnosti pomocí ostatních metod, měl by zvážit možnosti změn jakoby mimo samotný systém, neboť jejich zavedením může dosáhnout požadovaného zrychlení bez jakýchkoliv zásahů do architektury systému.

3.1.1 Vliv návrhu aplikací na distribuci zátěže

Jedno z obecných pravidel návrhu systému říká, že nejrychlejší operace v systému je ta, která se vůbec neprovede. Při návrhu aplikací s ohledem na zátěž systému dost často dochází k tvorbě programů, které zbytečně zatěžují systém. Příkladem může být aplikace, která ve snaze komplexně řešit zadaný problém nejprve provede velké množství různých dotazů na rozmanitá data, aby nakonec použila pouze malou část z nich. Druhým typem jsou aplikace, které sice všechna data a složité operace, které nad nimi provádí, využijí jako návrat uživateli, ale výsledek je natolik komplexní, že jej nevyužije v celé šíři uživatel. Při návrhu aplikací je tedy vhodné zamýšlet se nad množstvím dotazů a operací nad nimi, které budou muset být provedeny. Ukazuje se, že sada jednodušších aplikací je vhodnější než několik velkých. S úspěchem se také běh velkých aplikací může výrazně lišit v množství vykonaných operací v závislosti na poskytnutých vstupních datech, jejich počtu i hodnotách.

Při návrhu aplikací je rovněž nezbytné sledovat použité dotazy na data kladená databázovému serveru. Přestože se ptáme na údaje, které budou jistě pro další výpočet potřeba, je nezbytné ověřit, zda námi zvolený dotaz není databázovým serverem interpretován neoptimálně. Ve spolupráci s databázovým administrátorem tak může vývojář aplikací dosáhnout takových změn v použitých dotazech, které samy o sobě budou schopny snížit nutnou zátěž a tím i doby odezvy velmi výrazně bez dalších zásahů do jiných částí systému.

3.1.2 Uměle vynucená zátěž

Vzhledem k tomu, že zátěžová špička způsobená zvýšeným počtem současně přistupujících uživatelů k systému, je vůči normálnímu stavu jistým druhem anomálie, je nezbytné ji pečlivě zanalyzovat a vyhodnotit její příčiny. Administrativní systémy, které často kopírují původní metody administrace, se mohou snadno stát prostředkem pro realizaci různých typů soutěží. Je přirozené, že některé omezené zdroje jsou v běžném světě k dispozici pouze omezenému počtu lidí, kteří jsou vybráni na základě času jejich zájmu o tento zdroj. Například čerstvé pečivo je obvykle v obchodě dostupné pouze zájemcům v první části dne a zákazník, který přijde před zavírací dobou již počítá s jeho možnou nedostupností. Pokud však tento princip převedeme do elektronické podoby, stane se tak systém velmi nepříjemně prostorem pro nový druh soutěže, neboť narozdíl od klasického přístupu, který v případě administrativních systémů vynucuje fyzickou přítomnost v místě poskytnutí zdroje, umožňuje webový systém přístup ke zdroji všem bez ohledu na jejich aktuální polohu. Snadno tak dojde k přetížení systému, neboť všichni zájemci mohou řádově snadněji o získání tohoto zdroje usilovat. Příkladem z praxe univerzitního systému je například omezená kapacita některého semináře, která z různých důvodů vytváří několikanásobně větší množství zájemců. Pokud pro řešení tohoto problému použijeme klasickou administrativu, musí každý zájemce nejprve vyhledat způsob, jakým se do tohoto semináře zapíše a pak čekat až tento zápis bude možný. Naproti tomu při elektronickém zápise je přesně stanovený způsob a často i čas, kdy se zájemci k semináři mohou elektronicky přihlásit. Navíc toto přihlášení mohou pohodlně provést odkudkoliv a je proto pro ně řádově snadnější než klasický přístup.

Nárazová zátěž však nemusí nutně být způsobena takovouto formou distribuce omezených zdrojů. Stejně jako dopravní situace v centru velkých měst je závislá na období, ve kterém dochází obecně k většímu přesunu obyvatel, může být nárazová zátěž vyvolána podobnými časovými důvody, které nutně nemusí být dány nárazovým poskytnutím omezených zdrojů. Také tyto příčiny však je nezbytné pečlivě sledovat a vyhodnocovat.

Důležitým pravidlem pro odstraňování takto uměle vyvolané zátěže je vždy snaha odstranit její příčinu. Je systémovou chybou, když jsou dlouhodobě nabízeny omezené zdroje formou časového pořadí evidovaného v elektronickém informačním systému. Pokud by se systém posílil na úroveň, která by umožňovala poskyt-

nutí zdroje omezenému počtu nejrychlejších uživatelů bez viditelného zatížení, byli by takto rychlým systémem vybráni „nejrychlejší“ uživatelé v podstatě náhodně. Takového rozdělení však lze dosáhnout běžnými prostředky bez nákladného posilování systému a původní záměr vhodného přidělení zdroje se nesplní. Jedinou možností proto zůstává změnit podmínky pro poskytování zdrojů tam kde je to alespoň trochu možné.

Přesto existují situace, ve kterých je nezbytné zachovat přístup k systému velkému množství uživatelů, kteří zároveň přistupují byť jen k jeho omezené části. Vyvolá-li takováto potřeba zvýšenou zátěž, je vhodné vytvořit pro ostatní uživatele podmínky pro zachování požadované dostupnosti. Jinými slovy pokud musí k přetížení dojít a není možné mu zabránit, pokusíme se zpomalit pouze ty uživatele, kteří vynucenou zátěž způsobují. Tuto metodu, která bude diskutována v další části, budeme nazývat řízená klasifikace požadavků.

3.2 Vyrovňovací paměti

I přes optimalizace dosažené nepřímými metodami zamezení zátěže existují případy, kdy se provádění složitých operací nevyhneme. Souběžné zpracování těchto náročných úloh navíc může zvyšovat zátěž i při běžném provozu bez ohledu na provozní špičky. S rostoucími nároky na aktuální dostupnost všech informací navíc dochází k zatěžování systému výpočty, které se doposud daly provádět v noci. Také zobrazení aktuálních dat v různých podobách nyní klade větší nároky na operace, které je nutné před každým zobrazením provést. Některé z těchto operací jsou však často prováděny opakovaně a většina z těchto provedení také opakovaně vrací tentýž výsledek. Vezmeme-li v úvahu předpoklad, že více operací prováděných v administrativním informačním systému pouze čte již vložená data, zjistíme, že opakováním téhož výpočtu nad daty, která se nezměnila, zvyšuje zátěž zbytečně.

Obvyklou ochranou proti tomuto jevu je použití vyrovňovacích pamětí, které uchovávají výsledky již provedených operací pro případné opakované použití. V případě webových informačních systému lze tuto techniku dobře využít v podpůrném rozhraní mezi aplikacemi a databázovým serverem. Aplikace tak namísto přímého dotazu do databáze vyvolají podpůrný mechanismus tohoto rozhraní, které nejprve ověří, zda výsledek tohoto dotazu již není ve vyrovňovací paměti aplikačního serveru. Další možné využití je například v uchování ve vyrovňovací paměti celý hypertextový dokument vytvořený prezentační částí aplikačního serveru. Ten pak může být odeslán bez použití nejen databázového serveru, ale i samotného aplikačního výpočtu.

Problém tohoto řešení spočívá v procesu invalidace vyrovňovací paměti, neboť pro konzistenci podávaných informací je nutné zajistit uchování ve vyrovňovací paměti pouze aktuálních dat, která jsou uložena v samotné databázi. Jednotlivé zmíněné typy vyrovňovacích pamětí se liší právě v řešení invalidace.

3.2.1 TTL Cache

Implementačně nejjednodušším typem vyrovnávací paměti je takzvaná TTL Cache (z anglického Time To Live). U dotazů, které pracují nad téměř úplně statickými daty, lze totiž předpokládat, že se v případě opakovaného použití během krátké doby vůbec nezmění. V takovém případě je možné výsledkům těchto operací přiřadit při počátečním vložení do vyrovnávací paměti jednoznačné časové razítko, které působí jako invalidátor. Data z vyrovnávací paměti se použijí pouze v případě, že ještě nebyl překročen časový limit daný touto značkou. Po jeho překročení se vyrovnávací paměť naplní opět aktuálním obsahem. U často prováděných dotazů můžeme tímto způsobem dosáhnout velké úspěšnosti i v případě poměrně krátké „doby života“ ukládaného záznamu. V praxi se tento druh vyrovnávací paměti vzhledem ke snadné implementaci často nasazuje také jako pojistka při implementaci složitějších způsobů invalidace. Pokud by nastala v invalidaci jinou metodou z nějakého důvodu chyba, takže by vyrovnávací paměť obsahovala neaktuální data, byla by tato alespoň po nějaké době nahrazena správnými. Toho lze využít samozřejmě pouze u dat, jejichž časová konzistence není kritická.

3.2.2 Test Cache

Principem metody Test Cache je nutnost před každým použitím dat z vyrovnávací paměti ověřit v primárním zdroji, že se jedná o aktuální verzi. Tohoto principu lze ovšem obvykle využít pouze při uchování výsledků velmi složitých operací, neboť primárním úkolem vyrovnávací paměti je použít data bez nutnosti jakéhokoliv přístupu k primárnímu zdroji. Obvykle se tato technika využívá ve zmíněné prezentační části aplikačního serveru, kdy ve vyrovnávací paměti jsou uchovávány například přeložené zdrojové texty aplikací nebo prezentačních šablon, u kterých zjištění, zda došlo ke změně je řádově rychlejší než jejich znovuvytvoření.

3.2.3 Push Cache

Metodou využívající přístup invalidace při změně je takzvaná Push Cache. Na proces změny všech dat vstupujících do výsledků je nutné nejprve připojit mechanismus invalidace všech vyrovnávacích pamětí obsahujících každý z těchto dotazů. To však může být při použití složitějších struktur ukládaných do vyrovnávací paměti netriviální problém. V některých případech je rovněž možné označit některá data jako neměnitelná a při eventuální nutnosti jejich změny vyvolat zneplatnění celé vyrovnávací paměti.

Různých technik realizujících vyrovnávací paměti použitelné v jednotlivých částech výpočtu informačního systému je celá řada. Jednotlivé způsoby lze navíc obvykle vhodně kombinovat, čímž lze dosáhnout poměrně velkých úspěchů při snižování doby odezvy a distribuci zátěže. Pro konkrétní použití je však nutné

přesně znát specifikum daného systému, aby zvolené řešení co nejlépe odpovídalo konkrétním potřebám.

3.3 Řízená klasifikace požadavků

Až doposud jsme se ve hledání způsobů ochrany před zvýšenou zátěží informačních systémů zabývali pouze metodami, které se snaží různými způsoby přetížením zabránit. Existují však případy, a zmíněný problém uměle vyvolané zátěže je toho důkazem, kdy použitelná možnost ochrany neexistuje. Při provozu reálných informačních systémů tak čas od času nastanou situace, u kterých přetížení nelze zabránit. U systémů poskytujících komplexní informace širokému spektru uživatelů však takovéto situace zbytečně způsobují nedostupnost celého systému i v případě, kdy k přetížení došlo pouze u jedné části. Techniku, která se snaží vyřešit i tento aspekt zátěže, budeme nazývat řízená klasifikace požadavků na základě jejich priority. Jejím podstatou je zajištění přístupu k systému některým uživatelům i ve chvíli, kdy je ostatním přístup odepřen z důvodu přetížení (tzv. řízené předbíhání), nebo naopak zabránit přístupu těm uživatelům, kteří jeho přetížení mohou díky uměle vyvolané zátěži způsobit (řízené zpomalení). Zátěž systému se za pomoci této metody sice nepodaří odstranit úplně, může však dojít k podstatnému snížení jejího nepříjemného dopadu na celkovou dostupnost.

3.3.1 Řízené předbíhání

Mezi různorodé služby, které informační systém poskytuje, jsou obvykle zařazeny také ty, které jsou svým posláním významné pro nepřetržitý chod systému. Zatímco u většiny služeb není vyžadována bezprostřední odezva, u těchto kritických aplikací může být jejich provedení jednou z hlavních priorit systému. Takovýmito aplikacím a následně pak operacím, které provádí, je vhodné přiřadit zvláštní prioritu, která by umožnila jejich bezproblémové fungování i v době zvýšené zátěže systému. Podobně může být užitečné přidělení zvýšené priority nikoliv konkrétní aplikaci, ale uživateli, který s ní pracuje. Tato aplikace pak může být obsloužena systémem různým způsobem v závislosti na uživateli, který ji spustil. Tento jev řízené klasifikace požadavků, kdy je předem určena priorita některých subjektů označujeme jako řízené předbíhání.

3.3.2 Řízené zpomalení

Z důvodu snížení vlivu masivní zátěže je však potřeba provést klasifikaci požadavků spíše na základě aktuálního stavu systému, než pomocí apriorního zvolení priority všech subjektů (aplikací nebo uživatelů). V takovém případě je vhodné naopak provádění aplikace, která způsobuje dočasnou zátěž, pozdržet a poskytnout

tak prostředky ostatním aplikacím být na stejné prioritní úrovni. Automatická detekce příčiny přetížení není obecně jednoduchá. Přesto existují zejména v případě uměle vynucené zátěže systému heuristiky, které umožňují požadavky způsobující dočasné přetížení detekovat a následně jejich provádění automaticky omezit. Tento princip vychází z předpokladu, že požadavky vyvolávající zátěž budou systémem vlivem této zátěže vždy obslouženy později. Pokud je tedy na úkor ostatních požadavků systém pozdrží – vyvolá řízené zpomalení – jejich provedení se sice ještě více zpomalí, ale ostatní aplikace zátěž nepocítí. Pokud tedy není možné například pomocí nepřímých metod zátěži zabránit (a lze s ní tedy počítat), omezí se alespoň její vliv na zbývající provoz systému.

Realizace metod řízené klasifikace požadavků závisí na způsobu rozlišení jednotlivých subjektů klasifikace (ať už určených na základě jejich významu explicitně, nebo pomocí detekce jejich vlivu na aktuální zatížení systému). To lze a je vhodné provést při procesu provádění požadavků co nejdříve, tedy nejlépe v počáteční fázi zpracování požadavku na aplikačním (webovém) serveru. Z daného požadavku je ještě před začátkem jeho provádění jasné, jaké aplikace se týká a kdo požadavek zadává. Moderní architektura webových informačních systémů popsána v předchozí kapitole však dokáže zátěž na aplikační úrovni poměrně snadno přenést na databázovou část. Aplikační server se tak při klasifikaci požadavků snaží zabránit zátěži, která je vyvolaná až při provádění operací na databázovém serveru. V případě, že databázové operace provádí jeden databázový server (nikoliv klastr jak je tomu u aplikačního serveru), mechanismus klasifikace požadavků musí poměrně složitým způsobem jeho zátěž monitorovat, aby předešel jeho zatížení. Na základě tohoto monitoringu musí v případě zvýšené zátěže některé aplikace pozdržet bez toho, aniž by se zahájilo jejich provádění. Možné řešení tohoto jevu však nabízí distribuce databázové zátěže pomocí klastrování, kdy jednotlivými požadavky lze zatížit různé uzly databázového klastru tak, že k přetížení dojde pouze u těch uzlů, které obsluhují řízeně zpomalené nebo neprioritní požadavky. Zároveň však nedochází k pozdržení jednotlivých aplikací uměle na aplikačním serveru, ale až na základě konkrétního zatížení procesorů databázových uzlů vyhraněných pro zátěž.

Hlavní podmínkou pro realizaci řešení řízené klasifikace je tedy změna architektury databázového subsystému. Tato metoda distribuce zátěže je proto hlavním tématem dalších částí práce. Jednotlivé aspekty řízené klasifikace mohou být pak na základě jejich výsledků více rozšereny v eventuálním dalším vývoji v této problematice.

3.4 Distribuce databázové zátěže

Poslední z obecných metod distribuce masívní zátěže produkčního informačního systému kterou zmíníme, se týká distribuce zatížení databázového serveru. Jak již vyplývá ze samotné architektury systému, vhodnou technikou pro její dosažení je rozložení výpočetního výkonu databázového serveru na několik nezávislých strojů, které společně sdílí stejná data uložená ve sdíleném diskovém subsystému. Samotná

technologie databázových klastrů představuje celou řadu možných přístupů a řešení, které jsou obvykle výsledkem dlouhého vývoje velkých softwarových společností. Pro úlohu nasazení těchto technologií do produkčního informačního systému proto využijeme výsledky jejich vývoje dostupné v podobě komerčních produktů a našim úkolem bude vytvořit vhodné podmínky pro jejich začlenění do produkčního informačního systému.

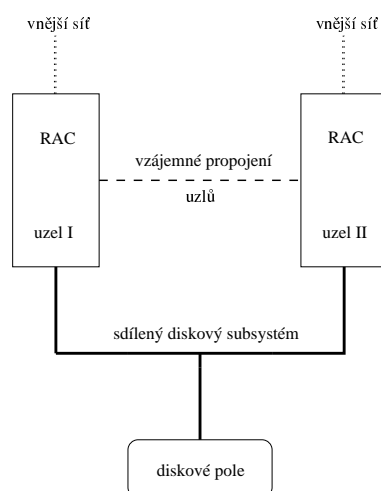
Důvodů, proč lze považovat řešení distribuce databázové zátěže formou databázového klastru v rámci metod zamezení zátěže celého informačního systému za nejdůležitější, je hned několik. Předně z analýzy architektury systému vyplývá že výpočetní část databázového serveru je vhodným místem pro určení úzkého místa zátěže celého systému a tedy jeho optimalizace je v průběhu procesu zmírnění důsledků zátěže zcela zásadní. Rozdělení zátěže na více paralelně běžících uzlů klastru může zvýšit celkovou výpočetní sílu systému a tím řadě zátěžových špiček zabránit. Toto rozložení však navíc vytvoří vhodné podmínky pro metodu řízené klasifikace, která pak může rozdělit zátěž přicházející s požadavky na jednotlivé databázové stroje a tím umožnit vysokou propustnost alespoň u některých prováděných služeb.

Budeme-li se zabývat konkrétní implementací databázového klastru nad databázovým systémem Oracle, musíme oproti standardní konfiguraci zajistit databázovému programu tzv. sdílený diskový subsystém, což je souhrn hardwarových a softwarových prvků podporovaných operačním systémem, které způsobí, že každá instance databázového programu bude mít přístupná všechna sdílená data. Komerčních řešení pro různé operační systémy a hardwarové platformy existuje celá řada. Nejrozšířenější jsou zřejmě externí sdílená disková pole, která za pomoci rychlé externí sběrnice (typu SCSI nebo Fibre Channel) umožní propojení všech uzlů klastru se samotnými disky. Investice do takového zařízení může být oproti dostupným dostatečně rychlým uzlům klastru postavených na platformě osobního počítače velmi vysoká. Záměrem této práce proto je nalézt řešení snižující náklady na pořízení sdíleného diskového subsystému zavedením softwarové emulace diskového spojení pomocí běžné počítačové sítě.

4 Řešení sdíleného diskového subsystému

V této kapitole se zaměříme na praktickou část implementace databázového klastru, který je, jak vyplývá z předchozích kapitol, důležitým předpokladem pro dosažení optimálního rozdělení zátěže informačního systému. Jako databázový program byl zvolen produkt Oracle Database společnosti Oracle, který ve verzi 9i představuje špičku mezi současnými komerčně dostupnými relačními databázovými programy. Jeho volitelnou součástí je rovněž propracovaná podpora databázových klastrů s názvem Real Application Clusters (popsaná v [3]), která umožňuje vybudování plnohodnotného databázového klastru. Jednou z hlavních výhod tohoto řešení je navíc podpora velkého množství různých operačních platforem, která umožňuje velkou pestrost při návrhu databázového subsystému. Bližší seznámení s jeho principy a způsoby implementace lze získat z článků [4] až [8].

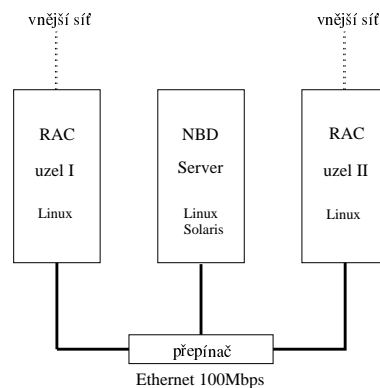
Základem řešení Oracle Real Application Clusters je spojení jednotlivých uzlů pomocí meziuzlové komunikace a současně jejich sdílení společných dat pomocí sdíleného diskového subsystému. Díky velké portabilitě se tak nabízí celá řada možností, jak jednotlivé části naimplementovat. Meziuzlová komunikace se v základní verzi může omezit na běžnou síť typu ethernet a standardní protokoly TCP/IP. Pro náročnější operace (zejména pro efektivnější synchronizaci sdílené paměti) však je možné použít i rychlejší metody (například různé formy spojení typu memory channel). Při stále se zvyšující rychlosti sítí typu ethernet a dalším technologickém rozvoji jak optických tak metalických vodičů, se ukazuje toto řešení jako dostačující při současném zachování velmi nízké pořizovací ceny. Pro meziuzlové spojení tedy budeme dále uvažovat pouze síť typu ethernet.



Obrázek 3: Schéma obecného zapojení dvouuzlového databázového klastru

Podobné možnosti se nabízí také při volbě technologie sdíleného diskového subsystému. Od původních sdílených diskových polí připojených na sběrnici SCSI se do popředí dostávají stále dostupnější diskové subsystémy typu Fibre Channel, umožňující opět větší rychlosti a nižší přístupovou dobu při práci s disky. Přesto však je pořizovací cena těchto diskových polí vzhledem k pořizovací ceně rychlých počítačů pro samotné výpočetní uzly velmi vysoká. Řada architektů informačních systémů tak hledá nové dostatečně výkonné metody propojení všech uzlů klastru se sdílenými disky. Existují například možnosti externího spojení disků pomocí sběrnice Fire Wire původně vyvinuté pro propojení osobních počítačů s multimediálními periferiemi zajišťující rychlý přenos multimediálních dat.

Na základě zkušeností s rychlými a zároveň dostupnými sítěmi typu ethernet se však nabízí možnost využití tohoto přenosového média také pro propojení diskové části systému. Kromě jiných možností, které však vedou zpět ke drahými diskovými polím poskytujícím diskovou kapacitu pro rozsáhlé sítě, se nabízí také námi zvolená technika síťového sdíleného subsystému. Hlavní myšlenka tohoto řešení spočívá v obecném mechanismu ukládání dat po síti realizovaném pod názvem Network Block Device v operačním systému Linux. Záměrem tohoto projektu bylo zakomponovat podporu pro virtuální disky na nejnižší úrovni operačního systému tak, že síťový disk se tváří jak pro aplikace tak pro služby operačního systému jako lokální disk.



Obrázek 4: Příklad zapojení klastru pomocí síťového diskového subsystému

Oproti běžnému sdílení dat jaké známe například z všeobecně podporovaných síťových souborových systémů NFS či AFS nebo podobných řešení na jiných platformách je tento přístup zcela transparentní na zvoleném souborovém systému. K síťovému disku tak může operační systém přihlížet jako k lokálnímu disku na úrovni blokového vstupně výstupního zařízení, což například umožňuje vytvářet softwarové zrcadlené disky s vysokou ochranou proti výpadku, neboť jednotlivé části zrcadla mohou být po síti uloženy fyzicky na jiném místě. Podstatnou vlastností tohoto řešení, která vedla k myšlence jejího zapojení do konfigurace databázového klastru, je možnost napojení tohoto blokového zařízení uvnitř jádra operačního systému Linux na zařízení umožňující přímý (raw) přístup k periferiím, což je nutný požadavek pro uložení dat v prostředí Real Application Clusters. Zatímco u dra-

hých diskových polí je toto zařízení obsluhováno přímo ovladačem pro toto pole, technika síťového blokového zařízení vytváří jakoby virtuální zařízení, které je ve skutečnosti umístěno jako služba na jiném počítači v síti.

Pro vybudování databázového klastru budeme proto potřebovat operační systém Linux a jeho podporu pro síťové blokové zařízení. Velkou výhodou tohoto řešení však je možnost umístit data pro síťové sdílení na počítač i s jiným operačním systémem. Takzvaný server síťového blokového zařízení, tedy program poskytující po síti ovladačům blokových zařízení na straně klienta, je běžný program, který může být spuštěn pod libovolným uživatelem a tedy jako takový lze přeložit i pro systémy, jejichž jádro není pro návrháře aplikací přímo k dispozici. Server síťového blokového zařízení je ve skutečnosti provozován pomocí několika procesů, ze kterých každý obsluhuje jeden fyzický soubor na disku prostřednictvím jednoho portu síťového protokolu TCP. Takovýto soubor je pak pomocí ovladače síťového diskového zařízení v jádře operačního systému Linux zpřístupněn jako znakové (raw) zařízení příslušným vstupně-výstupním procesům databázového klastru Oracle Real Application Clusters, které sami zaručí synchronizaci přístupu k jednotlivým datům ze všech zapojených uzlů. Pro úplnost dodejme, že v současné době je ve fázi testování možnost zpřístupnit tato zařízení nikoliv přímo, ale pomocí souborového systému Oracle Cluster Filesystem, který umožní (opět pod operačním systémem Linux) připojení sdíleného disku ke kořenovému souborovému systému uzlu, což dále usnadňuje správu jednotlivých dat. V dalším výkladu se však zaměříme pouze na tradiční pojetí formou přímého přístupu k datům bez použití souborového systému.

Ačkoliv je metoda zapojení sdíleného diskového subsystému na první pohled přímočará, její implementace není úplně jednoduchá, což dokazuje také řada kladných ohlasů od architektů databázových systémů, které jsem po jejím zveřejnění obdržel. Následující části proto obsahují přesný popis implementace této metody.

4.1 Server síťového blokového zařízení

Ze všech dostupných platforem, na kterých by mohl být server síťového blokového zařízení jako součást síťového diskového subsystému provozován, jsem zvolil operační systém SunOS/Solaris na platformě Sun Ultra SPARC. Ačkoliv by se mohlo zdát, že tato platforma nesplňuje předpoklady motivace našeho řešení, tedy nižší náklady na pořízení diskového subsystému, mohou se zkušenosti získané z jejího provozu dále uplatnit. Vzhledem k tomu, že implementace pro operační systém Linux jako vývojovou platformu sdíleného blokového zařízení nepřináší takové problémy, nebude následující popis zahrnující většinu kroků společných pro obě platformy zbytečný.

Pro naše účely budeme dále používat programový balík síťového blokového zařízení Network Block Device verze 2.0, jehož domovská stránka je uvedena v sekci Literatura a zdroje [11]. Tato stabilní verze, která je rovněž standardní součástí

distribuce linuxového jádra, má v současné době následovníka v podobě tzv. rozšířeného síťového blokového zařízení (Enhanced Network Block Device). Přestože tento projekt přináší některá další vlastnosti umožňující například snadnější správu síťových disků, v době tvorby našeho řešení docházelo k jejímu prudkému vývoji a případné použití v prostředí databázového klastru nebylo možné. Do budoucna však nelze jeho použití pro tyto účely vyloučit.

4.1.1 Kompilace programu

Prvním úkolem, který je nutné na platformě systému Solaris vyřešit je kompilace programu. Zatímco pro běžné distribuce operačního systému Linux je server síťového blokového zařízení distribuován v binární formě a stačí jej tedy nainstalovat, pro ostatní systémy se kompilaci nevyhneme.

Zdrojový kód balíku obsahujícího jak server tak klient je distribuován ve formě standardního archivu TAR komprimovaného programem GZIP. V novějších verzích operačního systému Solaris je již tento program standardní součástí instalačních disků, případně jej lze pro platformu Solaris doinstalovat nebo také přeložit z dostupných zdrojových textů.

Protože primární platformou této distribuce je systém Linux, překlad zdrojového kódu počítá s jeho standardním prostředím zejména ve formě hlavičkových souborů jazyka C. Pro operační systém Solaris proto musíme z distribuce Linuxu přenést soubor `nbd.h` do nově vytvořeného podadresáře `nbd/linux`, kde `nbd` je adresář obsažený v právě rozbaleném archivu.

Samotnou kompilaci je nutné provést na platformě Solaris s použitím opět volně přístupného překladače GCC verze 2. V našem případě šlo o verzi 2.95.2. Oproti verzi pro systém Linux je však nutné kompilaci provést za použití knihoven `socket` a `ns1`, které zajišťují síťovou komunikaci.

Výsledkem této kompilace, proběhne-li úspěšně, by měl být spustitelný soubor `nbd-server`. Po jeho spuštění bez jakýchkoliv parametrů obdržíme základní informace o programu s nápovědou k použití.

Tento soubor je z celé distribuce jediná potřebná součást, která je pro spuštění serveru síťového blokového zařízení potřeba. Lze ji tedy zkopírovat na vhodné místo, odkud bude následně pomocí startovacích skriptů spouštěn. Příkladem takového adresáře, kam lze soubor umístit je například adresář `/usr/local/sbin/`.

Volitelně lze tento program doplnit o konfigurační soubor `nbd_server.allow`, který obsahuje seznam IP adres strojů oprávněných k serveru přistupovat. Tím by měla být základní kompilace serveru hotova.

4.1.2 Konfigurace a běh serveru

Server jako standardní služba může realizovat svoji síťovou komunikaci pomocí neprivilegovaného portu protokolu TCP, což umožní jeho běh i pod identitou běžného uživatele. Je tedy vhodné veškerou činnost serveru provádět pod nově založeným uživatelem nikoliv pod superuživatelé root.

Samotný běh serveru síťového diskového zařízení však pro účely databázového klastru nestačí. Pro jeho spuštění je potřeba nejprve vytvořit alespoň základní datové soubory, které budou jednotlivým uzlům nabízeny ke sdílení. Před instalací databáze stačí vytvořit prázdné soubory, které ovšem budou zabírat maximální možnou velikost daného databázového svazku (tablespace). Vytvoření lze provést například programem `dd`, nebo v případě systému Solaris pomocí standardního příkazu `mkfile`.

Právě velikost jednotlivých souborů na serveru je velmi důležitá neboť pro správný běh a vytvoření databáze je nutné vytvořit soubory vždy o něco větší než je minimální požadovaná velikost v dokumentaci k databázovému programu. Zvláštní pozornost je pak vhodné soustředit na svazku `SYSTEM`, jehož minimální velikost 300MB není dostatečná v případě použití dalších komponent databáze. V takovém případě je vhodné jeho velikost navýšit alespoň na dvojnásobek. Následující tabulka představuje seznam minimálních datových souborů, jejich jmen a doporučených minimálních velikostí:

tablespace/file	size	datafile name
SYSTEM	300M	system_raw
USERS	30M	users_raw
TEMP	110M	temp_raw
UNDOTBS (per instance)	210M	undo_<n>_raw
INDX	30M	indx_raw
TOOLS	20M	tools_raw
controlfile1	120M	controlfile_1_raw
controlfile2	120M	controlfile_2_raw
redo logs (2 per inst)	130M	redo<n>_<x>_raw
spfile	5M	spfile_raw
srvmconfig	110M	srvctl_raw
node monitor	10M	nm_raw

Zkratka `<n>` představuje číslo databázové instance (uzlu klastru), kdy každá instance vyžaduje svůj zvláštní datový soubor, který je ovšem viditelný všemi uzly klastru. Číslo `<x>` pak označuje pořadí souboru v případě, že je nutné založit více souborů téhož typu.

Soubory uvedené v této tabulce vytvoříme pomocí série příkazů:

```
mkfile 300m /orac/system_raw
```

```

mkfile 30m /orac/users_raw
mkfile 110m /orac/temp_raw
mkfile 210m /orac/undo_1_raw
mkfile 210m /orac/undo_2_raw
mkfile 30m /orac/indx_raw
mkfile 30m /orac/tools_raw
mkfile 120m /orac/controlfile_1_raw
mkfile 120m /orac/controlfile_2_raw
mkfile 130m /orac/redo1_1_raw
mkfile 130m /orac/redo1_2_raw
mkfile 130m /orac/redo2_1_raw
mkfile 130m /orac/redo2_2_raw
mkfile 5m /orac/spfile_raw
mkfile 110m /orac/srvctl_raw
mkfile 10m /orac/nm_raw
mkfile 20m /orac/drsys_1_raw

```

kteře je vhodné uchovat ve zvláštním konfiguračním souboru pro případ násobného spuštění instalace klastru. Podobně je vhodné vytvořit další soubory pro uživatelská data.

Adresář /orac je možné a doporučené vytvořit z důvodu odolnosti vůči chybám nad lokálním svazkem serveru, který je vytvořen redundantním spojením několika fyzických disků. V případě operačního systému Solaris je možné softwarovou redundanci zajistit pomocí programu Solaris Solstice DiskSuite.

Máme-li vytvořeny všechny potřebné soubory pro zajištění běhu klastru, můžeme nad nimi spustit jednotlivé procesy serveru síťového blokového zařízení, který jsme po kompilaci uložili do adresáře /usr/local/sbin. Pro každý soubor je nutné nejprve zvolit číslo neprivilégovaného portu protokolu TCP, pomocí něhož bude komunikace probíhat. Prakticky toto číslo závisí pouze na ostatních službách v systému. Je proto vhodné zvolit nejlépe jednu řadu čísel portů, na kterých poběží pouze tyto procesy.

Program nbd-server se spouští s parametry číslo portu a jméno souboru, který bude obsluhován. V praxi tak je vhodné vytvořit startovací skript, který nastartuje všechny potřebné procesy nad již vytvořenými soubory. Základní skript pak bude obsahovat řádky:

```

/usr/sbin/nbd-server 4101 /orac/system_raw &
/usr/sbin/nbd-server 4102 /orac/users_raw &
/usr/sbin/nbd-server 4103 /orac/temp_raw &
/usr/sbin/nbd-server 4104 /orac/undo_1_raw &
/usr/sbin/nbd-server 4105 /orac/undo_2_raw &
/usr/sbin/nbd-server 4106 /orac/indx_raw &
/usr/sbin/nbd-server 4107 /orac/tools_raw &

```

```
/usr/sbin/nbd-server 4108 /orac/controlfile_1_raw &  
/usr/sbin/nbd-server 4109 /orac/controlfile_2_raw &  
/usr/sbin/nbd-server 4110 /orac/redo1_1_raw &  
/usr/sbin/nbd-server 4111 /orac/redo1_2_raw &  
/usr/sbin/nbd-server 4112 /orac/redo2_1_raw &  
/usr/sbin/nbd-server 4113 /orac/redo2_2_raw &  
/usr/sbin/nbd-server 4114 /orac/spfile_raw &  
/usr/sbin/nbd-server 4115 /orac/srvctl_raw &  
/usr/sbin/nbd-server 4116 /orac/nm_raw &  
/usr/sbin/nbd-server 4117 /orac/drsys_1_raw &
```

a lze jej interpretovat běžným příkazovým interpretem. Spuštěním všech procesů, které nyní běží na pozadí a čekají na jednotlivých portech na navázání komunikace s klienty, je část serveru síťového blokového zařízení dokončena.

4.2 Klient síťového blokového zařízení

Oproti předchozímu serveru je klient síťového zařízení závislý přímo na operačním systému Linux. Vzhledem k faktu, že klient síťového zařízení musí být provozován přímo na uzlu databázového klastru, je nutné zvolit verzi programu Oracle Database pro Linux. Tato platforma je však dostupná a v poslední době nejen v souvislosti s databázovými klastry stále více podporovaná.

4.2.1 Konfigurace síťového rozhraní uzlu

Uzel databázového klastru, který je vystavěn nad síťovým sdíleným diskovým subsystémem, je součástí celkem tří obecně nezávislých sítí. Prvním rozhraním je spojen s ostatními částmi systému, zejména pak s aplikačním serverem, který mu pomocí tohoto rozhraní uděluje své požadavky. Druhé síťové rozhraní zajišťuje komunikaci mezi jednotlivými uzly klastru, které mezi sebou zasílají zprávy o vzájemné dostupnosti služeb a data uložená v operační paměti jednotlivých uzlů. Poslední síťové rozhraní je určeno ke komunikaci klienta síťového blokového zařízení se serverem. V jednoduché formě lze tyto sítě všechny spojit do jedné, neboť jednotlivé služby jsou na sobě nezávislé a používají rozdílné porty protokolu TCP. Obecně zejména z důvodu rozložení síťové zátěže však je možné tyto tři logicky oddělené sítě rozdělit a zvýšit tak výkonnost každé z nich.

Následující příklad ukazuje rozdělení jednotlivých síťových rozhraní a jejich pojmenování v rámci souboru `/etc/hosts` na prvním uzlu. Případné úpravy síťových zařízení jádra pak lze snadno vytvořit na základě tohoto souboru.

```
# Adresa externího veřejného rozhraní (eth0)  
147.251.48.101 rac1
```

```
# Venkovní adresa druhého uzlu
147.251.48.198 rac2

# Adresa pro meziuzlovou komunikaci na rozhraní (eth1)
10.0.0.1 rac1-int
# Vnitřní adresa druhého uzlu
10.0.0.2 rac2-int

# Adresa rozhraní síťového diskového subsystému (eth2)
10.1.1.1 rac1-data
# Adresa serveru síťového diskového subsystému
10.1.1.100 rac-data
```

Z výše uvedeného souboru vyplývá, že komunikace mezi jednotlivými uzly bude probíhat nad rozhraním eth1, zatímco mezi serverem sdíleného diskového subsystému a jednotlivými klienty pomocí rozhraní eth2 klienta. Za tímto účelem je vhodné zvolit rychlost rozhraní serveru tak, aby dokázala jednotlivé klienty obsloužit bez zbytečných prodlev. K výkonnostním aspektům se ještě vrátím v kapitole zaměřené na zkušební provoz.

4.2.2 Kompilace, instalace a běh klienta

Samotný klient síťového blokového zařízení se skládá ze dvou částí. První je obvykle formou modulu obsažena přímo v jádře operačního systému Linux a zajišťuje funkci ovladače příslušného zařízení, který komunikuje namísto s obvyklou periferií s druhou částí, která běží jako samostatný proces pod superuživatелеm root. Kompilace obou částí však obvykle není potřeba neboť první část je většinou zahrnuta v předkompilovaných jádrech distribucí, zatímco druhá je zahrnuta v jejich instalačních souborech. Případná kompilace je však pečlivě popsána v samotné distribuci zdrojového balíku a vzhledem k tomu, že na použití v prostředí databázových klastrů nemá vliv, nebudeme se jí podrobněji věnovat.

Po kompilaci a instalaci všech obou potřebných částí se můžeme zaměřit na jejich spuštění. První část představovanou modulem nbd jádra, lze zavést do systému pomocí příkazu `modprobe`.

V případě, že příslušný modul je správně nainstalován, zapíše jádro do systémového logu hlášení o jeho zavedení ve tvaru:

```
nbd: registered device at major 43
```

Druhá část klientské aplikace se spouští podobně jako tomu bylo u serveru pomocí programu `nbd-client`. Syntaxe jeho spuštění:

```
/usr/sbin/nbd-client <data server> <port> /dev/nb<n>
```

však kromě síťového jména (případně adresy) serveru a portu na kterém daný proces běží vyžaduje také jméno blokového zařízení, pod kterým budou data poskytována na daném portu dostupná. Hlavní číslo zařízení v rámci Linuxu je 43 a obvykle jsou jednotlivá zařízení pojmenována v adresáři /dev písmeny nb a pořadovým číslem. Ke každému portu serveru síťového blokového zařízení je tedy nutné přiřadit jedno blokové zařízení a spustit samostatný klientský proces.

4.2.3 Konfigurace v prostředí databázového klastru

Při konfiguraci klienta síťového blokového zařízení v prostředí databázového klastru je nutné nejprve ověřit, zda je v adresáři /dev vytvořeno dostatečné množství potřebných zařízení s hlavním číslem 43 (/dev/nb<x>). Poté pro ně lze po zavedení modulu jádra nbd spustit jednotlivé procesy, nejlépe opět pomocí spouštěcího skriptu obsahujícího následující řádky:

```
/usr/sbin/nbd-client rac-data 4101 /dev/nb1
/usr/sbin/nbd-client rac-data 4102 /dev/nb2
/usr/sbin/nbd-client rac-data 4103 /dev/nb3
/usr/sbin/nbd-client rac-data 4104 /dev/nb4
/usr/sbin/nbd-client rac-data 4105 /dev/nb5
/usr/sbin/nbd-client rac-data 4106 /dev/nb6
/usr/sbin/nbd-client rac-data 4107 /dev/nb7
/usr/sbin/nbd-client rac-data 4108 /dev/nb8
/usr/sbin/nbd-client rac-data 4109 /dev/nb9
/usr/sbin/nbd-client rac-data 4110 /dev/nb10
/usr/sbin/nbd-client rac-data 4111 /dev/nb11
/usr/sbin/nbd-client rac-data 4112 /dev/nb12
/usr/sbin/nbd-client rac-data 4113 /dev/nb13
/usr/sbin/nbd-client rac-data 4114 /dev/nb14
/usr/sbin/nbd-client rac-data 4115 /dev/nb15
/usr/sbin/nbd-client rac-data 4116 /dev/nb16
/usr/sbin/nbd-client rac-data 4117 /dev/nb17
```

Pomocí jednotlivých blokových zařízení 1 až 17 by tak měly být k dispozici všechny soubory poskytované servery síťového blokového zařízení na serveru rac-data a portech 4101 – 4117.

Databázový server Oracle však pro účely uložení dat potřebuje přímý (raw) přístup k datům, tedy nikoliv blokový, který je poskytován jádrem u blokového zařízení. Pomocí příkazu raw lze však z libovolné blokové zařízení namapovat na zařízení znakové, které pak umožňuje k jednotlivým datům přistupovat bez nut-

nosti přístupu k celým blokům. Všechna bloková zařízení je proto nutné namapovat na zařízení znaková (raw device).

Znaková raw zařízení mohou být v adresáři /dev umístěna ve zvláštním podadresáři. Jejich namapování vyžadovat spuštění následujících příkazů:

```

/usr/bin/raw /dev/raw/raw1 /dev/nb1
/usr/bin/raw /dev/raw/raw2 /dev/nb2
/usr/bin/raw /dev/raw/raw3 /dev/nb3
/usr/bin/raw /dev/raw/raw4 /dev/nb4
/usr/bin/raw /dev/raw/raw5 /dev/nb5
/usr/bin/raw /dev/raw/raw6 /dev/nb6
/usr/bin/raw /dev/raw/raw7 /dev/nb7
/usr/bin/raw /dev/raw/raw8 /dev/nb8
/usr/bin/raw /dev/raw/raw9 /dev/nb9
/usr/bin/raw /dev/raw/raw10 /dev/nb10
/usr/bin/raw /dev/raw/raw11 /dev/nb11
/usr/bin/raw /dev/raw/raw12 /dev/nb12
/usr/bin/raw /dev/raw/raw13 /dev/nb13
/usr/bin/raw /dev/raw/raw14 /dev/nb14
/usr/bin/raw /dev/raw/raw15 /dev/nb15
/usr/bin/raw /dev/raw/raw16 /dev/nb16
/usr/bin/raw /dev/raw/raw17 /dev/nb17

```

Takto namapovaná zařízení budou přístupná jednotlivým procesům databázového serveru Oracle a je tedy nutné přidělit jim potřebná přístupová práva v rámci souborového systému. Vlastníkem bude uživatel, pod jehož identitou je spuštěn databázový server. Ostatní uživatelé by měli mít jakýkoliv přístup zamezen.

Pro běžnou práci je však vhodné na první pohled nicneříkající očíslovaná jména jednotlivých zařízení převést zpět na jména souborů jednotlivých databázových svazků. Toho jednoduše dosáhneme pomocí souborových linků, tedy standardním unixovým příkazem ln:

```

ln -s /dev/raw/raw1 /orac/system_raw
ln -s /dev/raw/raw2 /orac/users_raw
ln -s /dev/raw/raw3 /orac/temp_raw
ln -s /dev/raw/raw4 /orac/undo_1_raw
ln -s /dev/raw/raw5 /orac/undo_2_raw
ln -s /dev/raw/raw6 /orac/indx_raw
ln -s /dev/raw/raw7 /orac/tools_raw
ln -s /dev/raw/raw8 /orac/controlfile_1_raw
ln -s /dev/raw/raw9 /orac/controlfile_2_raw
ln -s /dev/raw/raw10 /orac/redol_1_raw
ln -s /dev/raw/raw11 /orac/redol_2_raw

```

```
ln -s /dev/raw/raw12 /orac/redo2_1_raw
ln -s /dev/raw/raw13 /orac/redo2_2_raw
ln -s /dev/raw/raw14 /orac/spfile_raw
ln -s /dev/raw/raw15 /orac/srvctl_raw
ln -s /dev/raw/raw16 /orac/nm_raw
```

Toto přiřazení je poté vhodné uložit také do zvláštního konfiguračního souboru, který bude načten při vytváření databáze podpůrným konfiguračním programem Database Creation Assistant. Jeho jednoduchá syntaxe má tvar

```
<tablespace or datafile>=<path to raw device>
```

a vytvořit jej lze pomocí příkazu:

```
cat > /orac/datafiles.conf <<EOF
system=/orac/system_raw
users=/orac/users_raw
temp=/orac/temp_raw
undotbs1=/orac/undo_1_raw
undotbs2=/orac/undo_2_raw
indx=/orac/indx_raw
tools=/orac/tools_raw
control1=/orac/controlfile_1_raw
control2=/orac/controlfile_2_raw
redo1_1=/orac/redo1_1_raw
redo1_2=/orac/redo1_2_raw
redo2_1=/orac/redo2_1_raw
redo2_2=/orac/redo2_2_raw
spfile=/orac/spfile_raw
srvconfig_loc=/orac/srvctl_raw
EOF
```

Jméno vytvořeného konfiguračního souboru se pak instalačnímu programu předá pomocí proměnné prostředí `DBCA_RAW_CONFIG`.

Takto vytvořené prostředí na všech uzlech lze použít pro spuštění instalace databázového programu i pro vytvoření samotné databáze.

Instalace a následných běhů databáze vyžaduje na straně systému rovněž podporu pro softwarový monitoring činnosti databázového serveru. Monitorovací software v případě, že není něco v pořádku, zajistí pomocí jádra operačního systému restart počítače, který způsobí jeho automatické vyřazení z klastru, aby nedocházelo ke zbytečnému odmítání klientů, kteří se jinak snaží k uzlu přistupovat.

Z hlediska operačního systému je tato součinnost prováděna pomocí modulu `softdog`. Pro testovací účely je vhodné tento modul zavést se zvláštním parametrem `soft_noboot=1`, který zamezí fyzickému restartu počítače v případě, že došlo vlivem počátečního ladění k nějaké chybě.

Nyní je diskový subsystém plně připraven pro instalaci databázového serveru, pro který se tváří jako běžné diskové pole, ačkoliv se jedná vlastně o jakési virtuální síťové propojení. Samotná instalace databázového serveru vyžaduje rovněž řadu technicky specifických kroků. Ty jsou však přesně popsány v dokumentaci dodávané spolu s databází případně v poskytované podpoře. V naší práci proto tyto další kroky zahrnovat nebudeme a omezíme se pouze na výše uvedený popis mého řešení.

5 Metody testování zátěže

Důležitou součástí implementace nových vlastností do informačního systému jsou testy, které usnadňují rozhodnutí, zda novinky do systému zaváděné lze použít v produkčním provozu. Jedním z hlavních požadavků, kromě testů správné funkčnosti celého systému po zavedených změnách, je testování zátěže. Důvodem tohoto testování je zjištění, zda nové změny nebudou mít neblahý vliv na výkonnost systému a jeho odezvu. V případě změn, které se přímo týkají chování systému při zvýšené zátěži, je pak metodika měření zátěže velice důležitá. Z toho důvodu byla jejímu zvolení věnována v této práci zvláštní pozornost. Její výsledky popíšeme v následující kapitole.

V praxi se problematika vhodných testů zátěže dostává do stále větší pozornosti tvůrců a provozovatelů informačních systémů, neboť jejich vhodné zvolení, které se ukazuje být čím dál tím náročnější, může při finální implementaci ušetřit velké množství prostředků. Problém správného odhadnutí kapacity systému nelze vyřešit jednoduchou sadou testů, ale je nutné uvažovat velké množství vlivů působících na systém z různých zdrojů. Samotné testování zátěže by si vyžádalo další hlubší zkoumání. My se omezíme pouze na metodiky testování, které se osvědčily při zkušebním provozu nové metody distribuce databázové zátěže – implementace databázových klastrů. Jak se ukazuje, je dále popsána metoda vhodnou alternativou pro běžně používané a komerčně dostupné způsoby testování.

5.1 Klasické metody testování

Dříve než přejdeme k popisu metody zvolené v této práci pro ověření funkčnosti a vlivu na zatížení systému popisované metody databázových klastrů, zmíníme dvě techniky, které se v současné době běžně používají pro testování zátěže. První z nich spočívá v dopředném určení sady operací, které jsou pak na prototypové řešení opakovaně aplikovány při současném měření zatížení všech relevantních částí systému. Výhodou tohoto přístupu je snadná implementace, neboť pouze stačí spustit unifikovaný test na základě něhož se lze zaměřit na zjištěné úzké místo. Nevýhody tohoto zjednodušení se však v komplexním systému mohou projevit velmi zásadně. Zejména není u většího počtu složitých služeb, které dnes mohou informační systému poskytovat, jednoduché určit, které operace se mají testu účastnit. Pokud se zvolí nevhodná část operací, může se příslušný návrhář systému pokusit o jejich optimalizaci mnohdy však na úkor jiných operací. Nežádá se pak stává, že systém je optimalizovaný na určitou sadu testů, v nichž dosahuje vynikající výsledky, ale v reálném provozu je neefektivní. Druhým důvodem proč není vhodné na výsledky takového testu spoléhat, je skutečnost, že zejména při použití

na heterogenních webových informačních systémech je výkon systému závislý na konkrétním chování uživatelů, kteří přistupují z různých klientských stanic připojených k systému různě rychlými sítěmi. Počet současně prováděných operací a jejich struktura se tak dynamicky mění, jak jsme popsali v úvodních kapitolách. Testy založené na apriorním určení prováděných operací tyto vlivy nejsou schopny zohlednit a jejich případné následky tak nemohou být odhaleny. Tento typ testů však vzhledem k jeho snadné implementaci může být vhodným vodítkem při konstrukci zcela nových systému jako první přiblížení možného nastavení jednotlivých parametrů.

Druhá ze zmiňovaných metod se snaží nevýhodu apriorního zvolení testovacích operací eliminovat tím, že novou množinu operací vytvoří na základě přístupu testovacího uživatele k systému ve chvíli jeho testování. Před samotným testem se tedy nechá uživatel přistupovat k systému a testovací program zachycuje jeho komunikaci se systémem a tvoří si tak seznam operací, které by mohly být spouštěny při reálném provozu. Testování zátěže je pak prováděno opakovaným spouštěním těchto operací automaticky. Tím se lépe odhadnou dopředu možnosti prototypového řešení a vliv nastavení jeho parametrů na zatížení systému. Tato metoda ovšem stále nezohledňuje vliv přístupu různých uživatelů a různorodost spektra současně spuštěných operací, neboť automatický test může současně provádět maximálně ty operace, které ve skutečnosti testovací uživatel prováděl postupně. Nasazení většího počtu testovacích uživatelů by sice tento vliv snížilo, ovšem za současného významného zvýšení nákladů. Testovací uživatel však nikdy nemůže postihnout reálné požadavky uživatelů při provozu systému. V neposlední řadě tato metoda testování způsobuje konkrétní technické problémy, neboť například zachytávání operací testovacího uživatele v případě webových systémů není tak jednoduché, jak tomu bylo u klientských aplikací. Pokud je navíc reálný provoz systému prováděn nad zašifrovaným protokolem, což je v současné době běžné, nelze je provádět vůbec.

5.2 Testování zátěže simulací reálného provozu

Princip techniky testování zátěže pomocí simulace reálného provozu, kterou jsem použil ve této práci, se snaží poskytnout návrháři systému co nejvěrnější obraz chování jednotlivých komponent systému při jeho skutečném provozu. Toho je dosaženo na základě přesné znalosti všech operací, které se v daném systému kdy prováděly. Hlavní slabinou takového řešení je proto jeho nepoužitelnost v případě budování zcela nových systémů. Zaměříme se tedy nyní pouze na testování zátěže již existujícího systémů, u kterého se snažíme přidáním nových schopností docílit nových možností.

V moderních webových informačních systémech se ukazuje jako nezbytné evidovat přesně každou operaci pro účely zpětného dohledání. Správce systému tak může snadno dohledat v případě problému všechny informace o tom, co se uživatel snaží provádět, což může automaticky vést rychleji k odhalení problémů na straně

uživatele. Rovněž podstatným úkolem této evidence je napomoci k odhalení případného neoprávněného provádění operací v autentizované části systému. Některé z takto evidovaných údajů je však možné použít také pro účely testování zátěže prototypového systému. Mezi základní údaje, které lze pro tento účel využít, jsou kromě internetové adresy aplikace, kterou každý z přístupů spouští, také všechny parametry předávané prohlížečem. Navíc je při snaze co nejvěrnějšího přiblížení reálnému provozu nutné evidovat všechny identifikátory přístupujících uživatelů tak, aby každá aplikace byla prováděna pod identitou a tedy s přístupovými právy skutečného uživatele.

Pro samotné testování je nejprve nutné na straně databázového serveru testované konfigurace informačního systému vytvořit podmínky pro spuštění takto získaných záznamů reálných operací. To lze nejlépe provést importováním v ideálním případě všech dat z produkční části systému. Před získáním vzorku testovaných operací tedy nejprve provedeme jeho export z produkční databáze a následně export všech operací nad těmito daty později prováděných. Pro účely testování obvykle není nutné ovlivnit chod systému, neboť úplně konzistentní stav vynucující zvláštní režim ukládání změn po dobu exportu není nezbytný. Po importu všech dat do databáze prototypového systému je navíc nutné nastavit všem uživatelům nové heslo, pod kterým se vůči systému autentizují, aby bylo zajištěno provádění autentizovaných operací testovacími skripty.

Takto upravená databáze v rámci pokusného systému je již připravena na spuštění testovacích operací. V reálném systému však jsou operace prováděné skutečnými uživateli spouštěny z webových prohlížečů různě výkonných počítačů připojených z různě dostupných sítí. Tuto vlastnost lze ale pouze velice nesnadno při umělém spuštění zohlednit. Pro účely testování si tedy dovolíme tento vliv zanedbat a použijeme pouze lokálně dostupné testovací stroje. Vzhledem k tomu, že při spuštění operací založených na reálných požadavcích kladených uživateli je vhodné zachovat jejich serializovatelnost (každá operace může navazovat na stav systému ovlivněný předchozí operací), omezíme dále spuštění testovacích požadavků na jediný multiprocessorový počítač, který bude schopen bez jakékoliv složité synchronizace požadavky spouštět sice paralelně, ale při zachování předem daného pořadí.

Při realizaci této metody testování zátěže simulací reálného provozu jsem použil programovací interpretovaný jazyk Perl, který v sobě skrývá značnou podporu mimo jiné také pro komunikaci s webovým serverem včetně možnosti šifrování a lze tedy velmi snadno vytvořit jednoúčelovou aplikaci typu webový klient bez nutnosti spuštění externích programů. Výsledkem je jediný poměrně jednoduchý skript, který po načtení potřebné konfigurace otevře soubor, ve kterém jsou uloženy všechny prováděné operace, a ten postupně v cyklu zpracovává až do konce. Po načtení každého požadavku zavolá tento proces svého potomka (realizovaného novým procesem), který již ve své režii požadavek předá serveru, převezme od něj výsledek v podobě HTML dokumentu a zpět předá chybový status vrácený serverem. Takto lze paralelně spustit libovolný počet klientů zároveň, přičemž rodičovský proces, který požadavky čte a nechává zpracovávat svými potomky, je

v případě dosažení maximálního možného počtu spuštěných potomků automaticky pozastaven až do chvíle, kdy některý z nich neskončí. Počet současně spuštěných klientů lze přitom pohodlně měnit podle dosaženého zatížení na jednotlivých částech testovacího informačního systému.

Takto postavený skript se ukázal jako vhodná metoda pro vytvoření testovacího prostředí. Konkrétní výsledky, které s jeho pomocí byly při testování zvoleného řešení distribuce databázové zátěže pomocí databázových klastrů dosaženy, budou popsány v následující kapitole.

5.3 Techniky měření zátěže systému

Při testování zátěže systému je kromě zvolené techniky jejího generování nutné také přesně stanovit veličiny, které budeme při takovémto testování měřit a také způsob jejich měření. Z bližšího popisu architektury systému víme, že hlavní zátěž je soustředěna na procesory databázové části systému. Při měření zátěže je tedy nutné stanovit metodiku měření zátěže procesorů jednotlivých částí systému. Při zavádění databázových klastrů nad síťovým sdíleným diskovým subsystémem které jsme provedli, je však rovněž nezbytné znát zatížení diskového subsystému a to jak u původního řešení tak u nově sestaveného databázového klastru. To obnáší měření zátěže jak disků, tak síťového rozhraní operačního systému. Techniky zjištění těchto údajů se v jednotlivých použitých operačních systémech liší a jejich zjištění nemusí být vždy jednoduché. V následujícím textu proto popíšeme způsob, jaký byl při testování pokusného provozu použit.

5.3.1 Operační systém Linux

Operační systém Linux zpřístupňuje všechny údaje rozšířením souborového pseudosystému `proc` připojeného obvykle ke stejnojmennému adresáři kořenového svazku. Toto jednotné rozhraní umožňuje snadný přístup k datovým strukturám jádra obsahujícím všechny důležité informace. Chceme-li tedy zjistit například aktuální hodnoty parametrů jádra, aktuální konfiguraci hardwarových periférií, či libovolný ukazatel stavu systému, stačí otevřít příslušný soubor umístěný v adresáři `/proc` pro čtení jakoby to byl běžný soubor.

Pro účely měření testované zátěže systému budeme zjišťovat nejprve míru zatížení procesoru jako průměrný počet procesů připravených k provádění na procesoru za posledních pět minut. Ačkoliv se jedná o průměrnou hodnotu, pro účely testování zátěže procesoru je to dostačující ukazatel, neboť není tolik ovlivněn dočasnou výchytkou v době zjištění hodnoty oproti delšímu normálu. Příslušný údaj je uložen jako první desetinné číslo v souboru `/proc/loadavg`. Pro případné procentuální vyjádření je však nutné tento údaj vydělit počtem dostupných procesorů, abychom získali hodnotu procesorové zátěže celého počítače. Kromě hodnoty aktuální procesorové zátěže je zejména pro účely monitorování stavu systému v době

optimalizace jednotlivých zátěží měřit dobu, kdy jsou procesory ve stavu nečinnosti (idle), tedy kdy část procesů čeká obvykle na nějakou vstupně-výstupní operaci. Snaha návrháře pak je udržovat v době vysoké zátěže tuto hodnotu co nejnižší. Její procentuální vyjádření je vypočteno z obecné statistiky operačního systému uložené v `/proc/stat` navíc na základě všech měřených hodnot zatížení, které představují v případě operačního systému Linux uživatelské procesy, jádro systému a procesy s vyšší prioritou.

V případě operačního systému nám dále stačí měřit pouze zatížení síťového rozhraní, neboť sdílený diskový subsystém je v případě uzlů databázového klastru realizován právě pomocí sítě. Vzhledem k tomu, že v případě přístupu k programovým souborům na lokálním disku nedochází ke zvýšené zátěži a to ani na aplikačním ani na databázovém serveru, které jsou oba realizovány nad operačním systémem Linux, není potřeba diskovou zátěž pro naše účely u tohoto operačního systému zjišťovat. Síťové zatížení je podobně jako u zatížení procesoru uloženo jako celkový počet bytů přenesených tímto rozhraní od jeho zavedení v souboru `/proc/net/dev`. Po dosažení maximální hodnoty uloženého celého čísla se začíná počítat od nuly. Počet přijatých i odeslaných bytů za sekundu lze tak jednoduše spočítat jako průměr rozdílu celkového množství přenesených dat za celý měřený interval.

Kromě výše zmíněných údajů musí skript zaznamenávající jednotlivé veličiny v čase ukládat také pro každé měření časovou známku, která následně slouží pro srovnání údajů na jednotlivých strojích. Vzhledem k tomu, že měření je prováděno pouze krátkou dobu, není podle mých zkušeností nutné tuto časovou známku mezi jednotlivými synchronizovat vzájemnou komunikací. Jako vhodný prostředek se osvědčilo po počáteční synchronizaci systémového času jednotlivých počítačů s atomovými hodinami zaznamenávat údaj vždy na začátku pětisekundového intervalu. Hodnoty naměřené na jednotlivých strojích se tak liší pouze o velmi krátký časový interval, který je pro tento účel průměrného měření dostačující a velmi snadno implementovatelný. Tato metoda je pak použita rovněž při měření zátěže na operačním systému Solaris.

5.3.2 Operační systém SunOS/Solaris

Měření veličin stavu operačního systému Solaris budeme používat ke dvěma účelům. Prvním měřením je nutné zjistit zatížení diskového subsystému současného databázového serveru, který je vybudován právě nad operačním systémem Solaris. Druhé měření poté umožní zjistit aktuální hodnoty požadovaných veličin na serveru síťového blokového zařízení sloužícího jako sdílený subsystém testovaného databázového klastru.

Standardní součástí operačního systému unixového typu je program `iostat`, který v současné verzi pro operační systém Solaris přímo poskytuje všechny potřebné údaje zatížení disků systému. Pro naše účely nezávislého testování současné dis-

kové zátěže informačního systému jsem proto využil přímo údaje získané pomocí tohoto nástroje.

Pokud potřebujeme podobně jako v předchozím případě zaznamenat při testování zátěže také hodnoty jiných veličin, musíme opět napsat skript zjišťující tyto informace z obecného zdroje, neboť standardní program `iostat` nezobrazuje například aktuální hodnotu přenesených dat pomocí síťového rozhraní. Na rozdíl od operačního systému Linux slouží pro přístup k vnitřním strukturám jádra systému Solaris systémová knihovna `kstat`. V programovacím jazyce Perl pak existuje rozhraní s názvem `Solaris::Kstat`, které funkce poskytované touto knihovnou zpřístupňuje i pro toto programové prostředí. Jednotlivé hodnoty pak snadno získáme voláním příslušných funkcí knihovny s požadovanými parametry. Kromě hodnot zatížení procesoru a kapacity síťového rozhraní lze tímto způsobem získat také hodnoty zatížení disků bez nutnosti spouštění externího programu `iostat`.

Tím jsou připraveny všechny prostředky pro vytvoření prototypového řešení včetně podpory testování jeho provozu při zvýšené zátěži. Následující předposlední kapitola se věnuje konkrétním zkušenostem získaným při testovacím provozu a popisuje nejdůležitější závěry testů. Celkový průběh nejdůležitějších testů je poté přesně popsán a v grafech vykreslen v příloze.

6 Implementace prototypového řešení

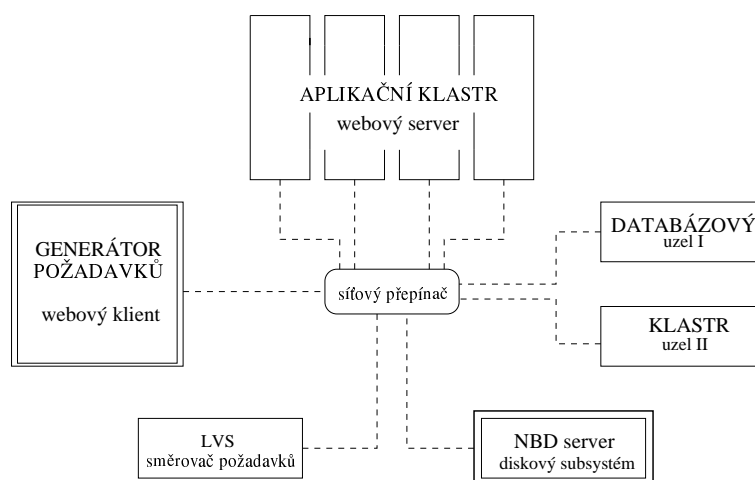
Jedním z hlavních cílů této práce bylo implementovat do pokusného provozu jednu z metod vedoucí ke snížení zátěže produkčního informačního systému školy, které jsme popsali v předchozích kapitolách. Hlavním cílem bylo jednak ověřit nově použité technologie a jejich vlastní zakomponování do stávajícího informačního systému školy a jednak zjistit jejich vliv na zátěž systému v provozních špičkách. Ačkoliv by se mohlo zdát, že úkolem prototypového řešení bylo vytvořit ve vývojových podmínkách novou verzi informačního systému zvládající lépe provozní špičky, naším cílem bylo pouze ověřit vliv jednotlivých změn na zbytek systému. Současný informační systém Masarykovy univerzity v Brně je provozován na strojích, které optimálně zvládají výkon při běžném zatížení. Takto výkonné stroje však pro vývojové prostředí nejsou k dispozici, proto za prototypové řešení byla zvolena méně výkonná architektura, která však na základě svých testů dokáže prověřit vliv případného dalšího rozšíření za účelem dosažení požadovaného výkonu na ostatní již existující části systému. Implementace prototypového řešení tedy spočívala v sestavení databázového klastru, který při zapojení k aplikačním serverům a testovacímu webovému klientovi umožnil práci nad reálnými daty již existujícího systému.

6.1 Architektura prototypového řešení

Původním záměrem bylo do pokusného provozu zapojit dva stroje databázového klastru, které by společně obsluhovaly databázi uloženou na discích pracovní stanice s operačním systémem Solaris a ke kterým by se připojil aplikační server podobný tomu, jaký je použit v aplikačním klastru produkčního informačního systému. K dispozici jsem měl sadu několika osobních počítačů, které byly původně využity právě jako stroje aplikačního serveru a v současné době jsou nahrazeny modernějšími. Přes drobné technické problémy, které instalaci provázely, se mi podařilo na základě postupu uvedeného v předchozích kapitolách databázový klastr nad síťovým sdíleným diskovým subsystémem spustit. Po přenesení všech dat z produkčního informačního systému a příslušné konfiguraci všech částí systému se však poměrně brzy ukázala tato architektura jako nedostačující. V takovéto konfiguraci totiž nedošlo k požadovanému rozložení zátěže tak, aby servery databázového klastru dosáhly maximálního zatížení před tím, než byly zatíženy jiné části systému, zejména aplikační server. Bylo nutné proto vybudovat i v tomto případě aplikační klastr a teprve při zapojení čtyř počítačů aplikačního klastru došlo k očekávanému zatížení dvou stejně rychlých počítačů v klastru databázovém. To dokazuje, že ani při zaměření se na distribuci zátěže není možné opomenout ostatní části systému. Problém se zatížením se však objevil i na straně webového klienta

spouštějícího jednotlivé klientské požadavky jako je popsáno v kapitole zabývající se testováním. Ten musel být pro zatížení celého systému schopen vygenerovat až čtyřicet požadavků najednou. Testovací skript jsem proto byl nucen spouštět na osmiprocesorovém studentském serveru a to pouze v pozdních večerních hodinách, kdy měl tento počítač volnou potřebnou výpočetní kapacitu.

Postupně jsem tedy pro prototypové řešení zvolil následující konfiguraci. Klient testovaných požadavků byl studentský server Silicon Graphics Origin 2100 s osmi procesory MIPS R12000 pracujícími na 350 MHz a operačním systémem IRIX. Jako brána pro aplikační klastr rozdělující požadavky kladené testovacím skriptem sloužila běžná pracovní stanice s operačním systémem Linux a procesorem AMD Athlon pracujícím na frekvenci 1500MHz. Zatížení tohoto stroje však není nijak vysoké a v praxi pak často může při takovéto konfiguraci sloužit i dalším službám. Celkový počet šesti osobních počítačů s procesorem AMD Athlon 900MHz, 256MB paměti a operačním systémem Linux byl rozdělen na čtyři stroje aplikačního a dva stroje databázového klastru. Databázový klastr pak využíval pro uložení dat starší pracovní stanici Sun Ultra I, která disponovala jedním procesorem Sun UltraSPARC taktovaným na frekvenci 168 MHz. Data byla uložena na discích připojených ke standardní SCSI sběrnici. Všechny počítače využívaly služeb jednoho síťového přepínače s kapacitou přenosu 100Mb/s.



Obrázek 5: Schéma zapojení prototypového řešení v samostatné privátní síti

6.2 Výsledky pokusného provozu

Prvním cílem pokusného provozu bylo ověřit možnost nasazení databázového klastru s technikou síťového sdíleného diskového subsystému v prostředí současného informačního systému školy. Všechny technické překážky, které implementace pokusného provozu přinesla, byly nakonec odstraněny a případné implementační problémy popsány v praktické kapitole zabývající se implementací této metody.

Tento výsledný stav však ukázal, že nabízené řešení je plně funkční a s možností jeho zavedení do produkčního prostředí je možné se dále zabývat. Překonané dílčí problémy, které byly postupně odstraněny, tedy není třeba dále v této kapitole specifikovat.

Hlavní úkol zavedení této metody však spočívá v odstranění problémů vzniklých zvýšenou zátěží systému. Cílem tedy bylo zjistit, jak se takovýto systém chová při zvýšené zátěži a zejména zda při distribuci zátěže mezi jednotlivé uzly databázového klastru nedojde k nově vzniklým problémům například při komunikaci mezi jednotlivými databázovými procesy a hlavně při uložení dat na zařízení dostupné přes pomalejší síťové propojení. Přestože z testů, které byly provedeny na současném databázovém serveru vyplývá, že maximální zatížení disků nepřekročí teoretickou kapacitu přenosového síťového média, existuje nebezpečí, že při použití technologie klastrů dojde ke zpomalení systému právě vlivem pomalejší komunikace. Jak však posléze testy ukázaly, síťová zátěž nepřekročila ani čtvrtinu maximální celkové zátěže sítě o rychlosti 100Mb/s, což by i při větším počtu uzlů databázového klastru mělo být pro server síťového blokového zařízení připojeným k přepínači dnes dostupnou rychlostí 1GB/s dostačující.

Při použití databázového klastru bylo také zajímavé v pokusném provozu ověřit, jaký vliv má přidání druhého uzlu na výkon celého systému. Z celkových výsledků vyplývá, že dva databázové servery jsou maximální možnou zátěž schopny obsloužit až 1,91-krát rychleji než je tomu v případě použití pouze jednoho uzlu. V praxi může být dalším použitím standardních databázových optimalizací dosaženo ještě lepšího výkonu, zatímco konkrétní spektrum kladených požadavků v reálném nasazení může tyto hodnoty rovněž změnit a to oběma směry.

Poslední z hlavních výsledků dosažených v pokusném provozu bylo ověření maximální možné propustnosti testované konfigurace. Při použití různých nastavení jsem nakonec dospěl k celkové propustnosti 31 452 uměle vyvolaných operací informačního systému za hodinu, zatímco v roce 2002 byla nejvyšší dosažená propustnost Informačního systému Masarykovy univerzity v Brně přibližně 50 000 požadavků za hodinu. Ačkoliv tato čísla jistě nelze srovnávat a propustnost produkčního systému se stále zvyšuje, ukazuje pokusný provoz poměrně značný potenciál, jaký databázové klastry navíc ve spojení s dostupným řešením diskového subsystému využívajícího síťové blokové zařízení dosahují.

Tyto tři základní výsledky a zejména pak skutečnost, že nově navržená metoda je použitelná i při zvýšené zátěži bez větších problémů na straně diskového subsystému jsou hlavním výsledkem této práce. Přesná čísla naměřených veličin jsou včetně grafického znázornění průběhu zátěže zmíněna v příloze Výsledky testů.

7 Závěr

Téma distribuce zátěže je dnes v popředí zájmu řady návrhářů a architektů produkčních informačních systémů, jejichž cílem je navrhnout efektivní a snadno dostupný systém. Úkolem této práce bylo seznámit se s různými možnostmi řešení tohoto problému a jejich následnou použitelností v provozu stávajícího Informačního systému Masarykovy univerzity v Brně. Důležitým předpokladem pro porozumění principům distribuce zátěže je pochopení architektury systému. Té se věnuje úvodní kapitola připravující vhodný základ pro popis principů jednotlivých studovaných metod. Kromě zmínění nepřímých metod poskytujících ochranu před vznikem zátěže se tato práce věnuje principům vyrovnávacích pamětí, takzvané řízení klasifikace a metodě distribuce databázové zátěže pomocí techniky databázových klastrů.

Právě poslední ze zmíněných metod se stala hlavním základem celé práce, neboť poskytuje dostupné řešení pro distribuci výpočetní zátěže databázového serveru, která je často úzkým místem při přetížení. Řešení, které je v této práci nabídnuto, navíc umožňuje sestavit databázový klastr pomocí dostupnějšího síťového diskového subsystému, což celkový proces distribuce zátěže značně usnadňuje. Toto řešení bylo při tvorbě práce zveřejněno a úspěšně přijato řadou odborníků z oboru databázových systémů (například v článku [12]). Lze tedy říci, že tato metoda je dobrou alternativou pro běžně používaná řešení.

Toto tvrzení je v práci podpořeno popisem konkrétního prototypového systému, který byl s úspěchem zaveden do pokusného provozu a umožnil tak provést sadu testů dokazujících možné nasazení také v praxi. Testy, které byly k tomuto účelu použity, vychází z dosavadní zkušenosti testování masívní zátěže produkčních informačních systémů a rovněž obsahují nové principy vedoucí k výsledkům více se přibližujícím reálnému zatížení systému.

Na základě těchto zkušeností lze metodu databázových klastrů s využitím techniky síťového sdíleného diskového subsystému použít jako alternativu k tradičním databázovým systémům. Vzhledem k nízkým pořizovacím nákladům se také může stát součástí dalšího rozvoje Informačního systému Masarykovy univerzity v Brně, neboť nabízí nejen možné zvýšení výkonu, ale také vhodnou platformu pro realizaci dalších softwarových metod distribuce dočasné špičkové zátěže. Přestože by při jejím případném zavedení vzrostly investiční náklady na pořízení vhodného databázového programového vybavení, snížení nákladů na hardwarové vybavení zajistí celkově nízkou pořizovací cenu systému. V neposlední řadě lze mezi výhody této metody zařadit také značné zvýšení dostupnosti, které databázové klastry přinášejí.

Přestože zmíněná metoda popsaná v této práci je vhodná pro praktické nasazení v produkčním informačním systému, během práce se objevily další možnosti zejména z oblasti softwarových metod distribuce zátěže, které by dalším případným vývojem mohly přinést ještě lepší výsledky. Výhodou diskutované techniky databázových klastrů je navíc skutečnost, že těmto metodám připravuje vhodné prostředí a stává se tak branou k dalšímu vývoji v této oblasti.

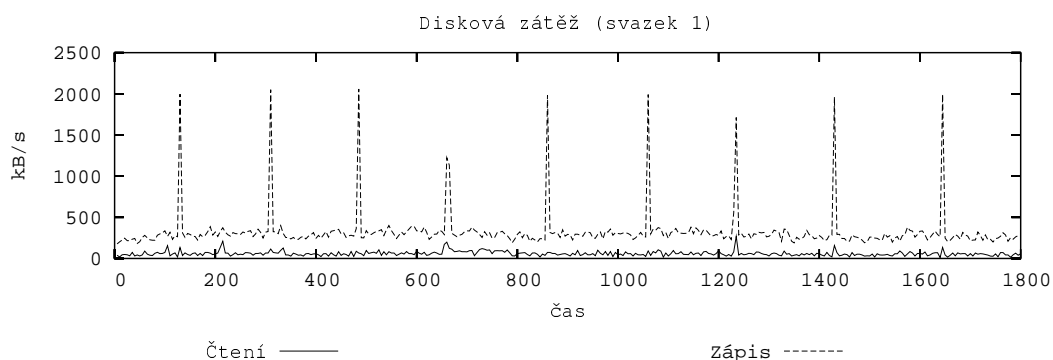
8 Literatura a zdroje

- [1] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: *Database system implementation*, Prentice Hall, ISBN: 0130402648, 1st edition, 1999
- [2] Tushar Mahapatra, Sanjay Mishra, Jonathan Gennick, Deborah Russell: *Oracle Parallel Processing*, O'Reilly & Associates, ISBN: 156592701X, 1st edition, 2000
- [3] Mark Bauer: *Oracle9i Real Application Clusters Concepts Release 2*, Oracle Corporation, Part No: A96597-01, 2002
- [4] Jack Cal, Simon Leung: *Building Highly Available Database Servers Using Oracle Real Application Clusters*, Oracle Corporation, May 2002
- [5] Lawrence To: *Maximum Availability Architecture*, Oracle Corporation, July 2002
- [6] Sohan DeMel, Merrill Holt, Mark Bauer: *Oracle9i Real Application Clusters, Cache Fusion Delivers Scalability*, Oracle Corporation, May 2001
- [7] Vijay Lunawat: *Oracle E-mail Server Deployment to Real Application Clusters Environment*, Oracle Corporation, January 2002
- [8] *Step-By-Step Installation of RAC on Linux*, Oracle Corporation, Doc ID: 184821.1, 2002
- [9] *Setting up Real Application Cluster (RAC) environment on Linux - Single node*, Oracle Corporation, Doc ID: 166830.1, 2002
- [10] *Oracle Technology Network*, <http://otn.oracle.com/>, informační zdroj společnosti Oracle Corporation
- [11] *Network Block Device Homepage*, <http://nbd.sourceforge.net/>, domovská stránka projektu síťového blokového zařízení.
- [12] Brian Hengen: *Developing a RAC System On a Limited Hardware Budget* In: IOUG Live! 2003, bude publikováno

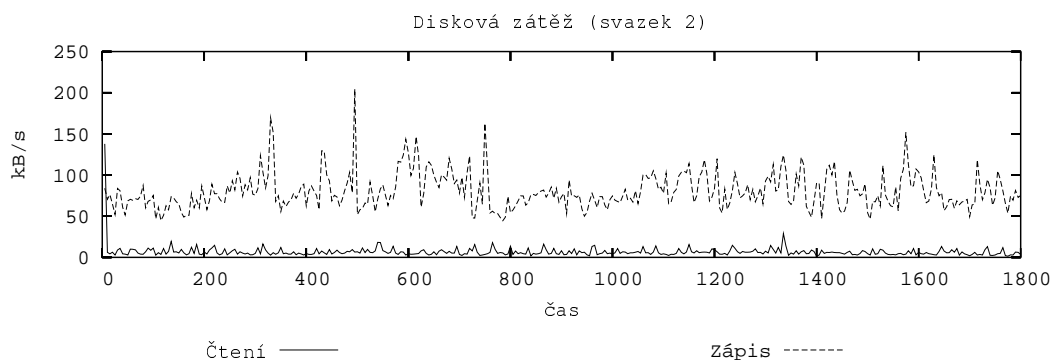
9 Příloha: Výsledky testů

I. Testy současného systému

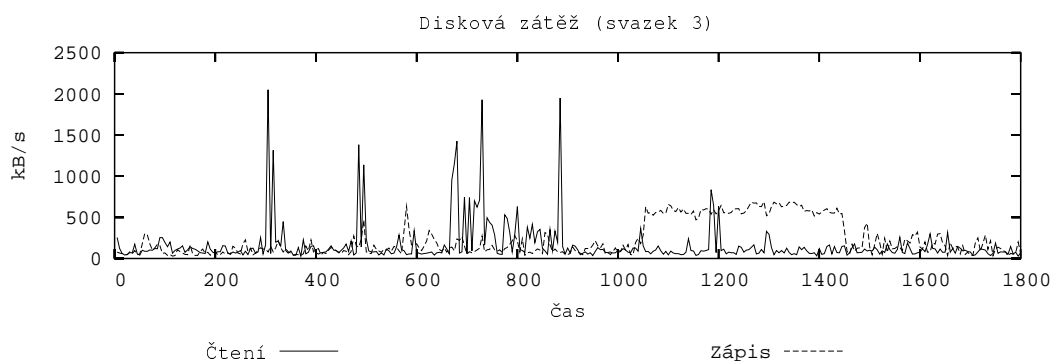
Úkolem testů současného systému bylo ukázat, jaké maximální zátěži je vystaven diskový subsystém stávajícího řešení a stanovit tak velikost potřebného výkonu nově zaváděného síťového sdíleného diskového subsystému. Zde uvedené příklady grafů představují běžnou denní zátěž na všech třech datových svazcích databázového serveru. Jednotlivé naměřené hodnoty udávají rychlost čtení a zápisu v kilobytech za sekundu.



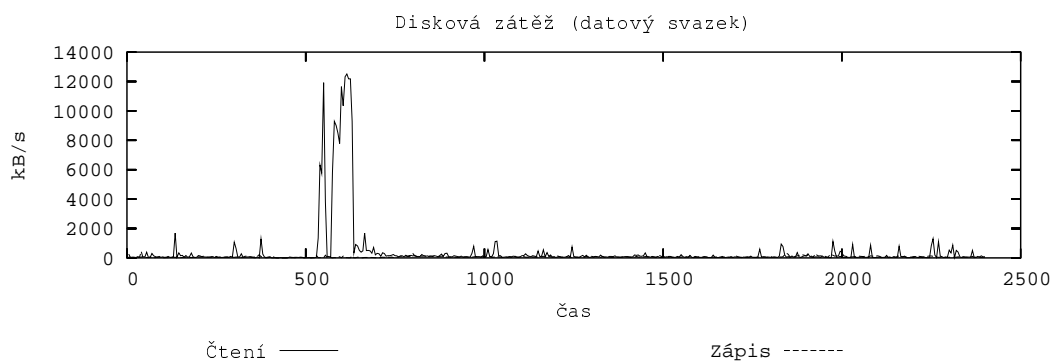
Periodické špičky, které se objevují na svazku 1 jsou způsobeny opakovaným zálohováním databázového logu na tento svazek. Další zátěž je způsobena méně důležitými datovými soubory.



Převažující zápis na diskový svazek 2 je způsoben datovými soubory pro uchování aplikačních logů systému, které obsahují informace o každé prováděné operaci a jsou zapisovány vždy po jejím provedení. Naopak čtení z těchto souborů se obvykle neprovádí.



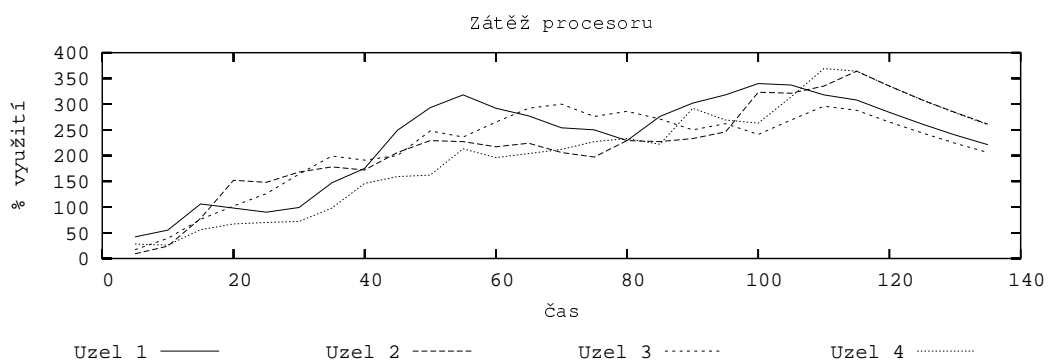
Svazek 3 představuje zátěž na hlavních datových souborech, kde dochází jak ke čtení tak k zápisu změn.



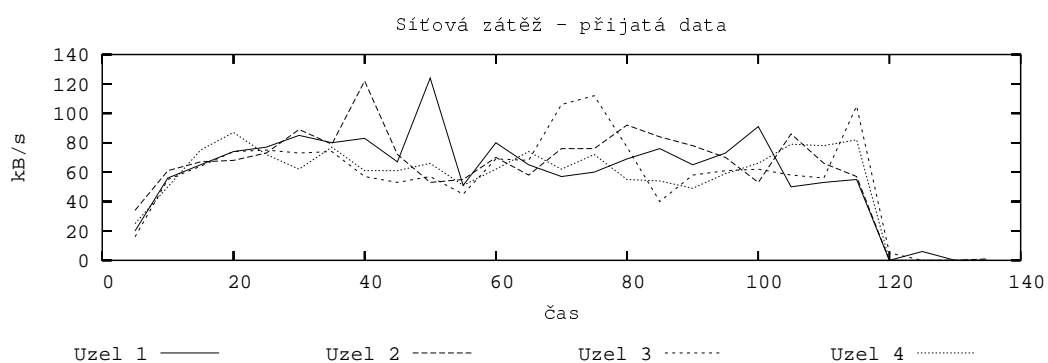
Poslední z uvedených grafů ukazuje špičkovou zátěž v době startu systému, kdy dochází ke kontrole (čtení) integrity dat, což je maximální dosažená rychlost čtení.

II. Testy prototypového řešení

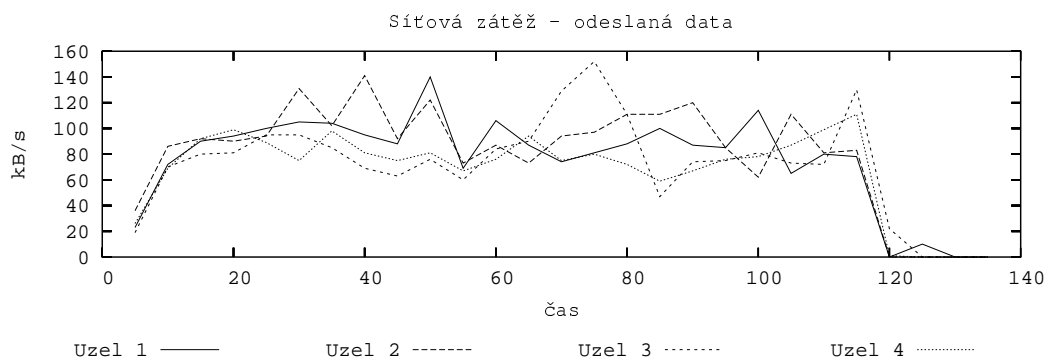
Testy prototypového řešení ukazují výkon jednotlivých částí nově navrženého systému při pokusném provozu. Ze všech prováděných testů se zde omezíme pouze na test, při kterém byl systém optimálně nastaven tak, že bylo dosaženo maximálního výkonu. Tento pokusný provoz byl prováděn nad menším objemem dat než předchozí test. Výsledky obou testů však naznačují, že dosažení výkonu stávající konfigurace je v novém modelu možné.



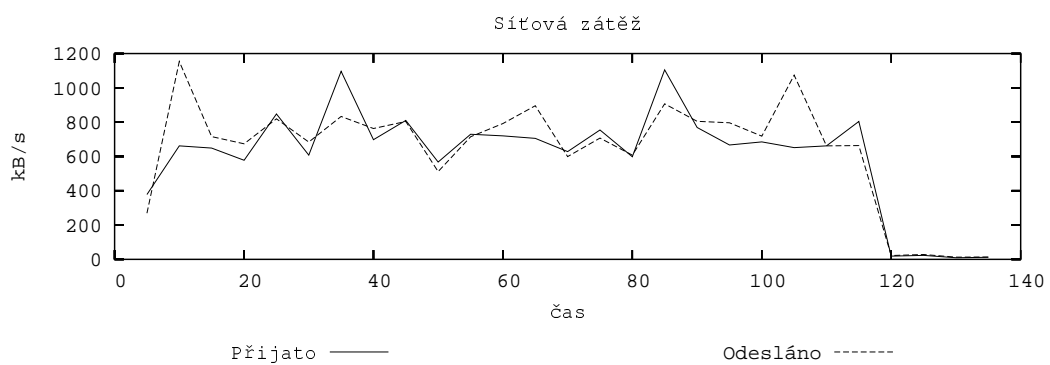
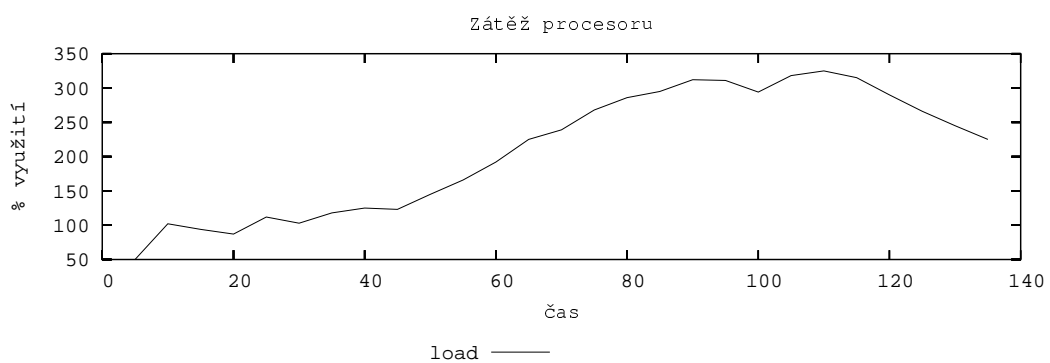
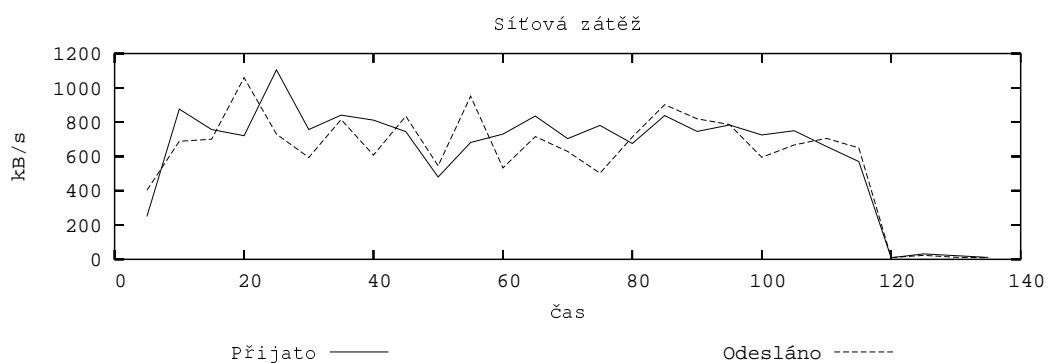
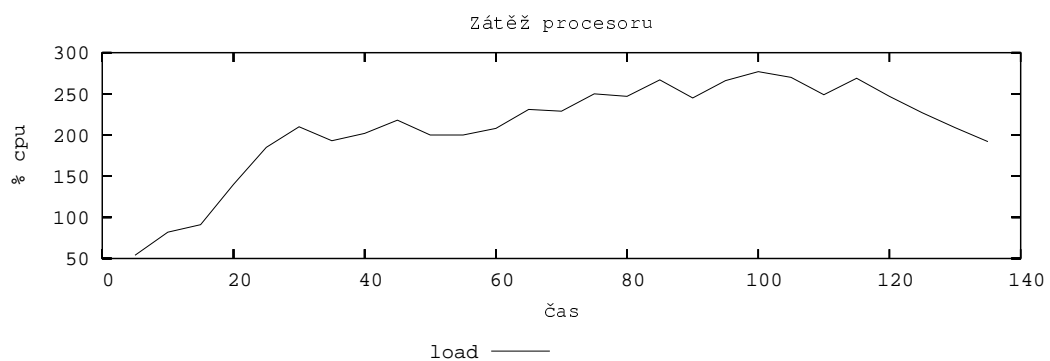
První z uvedených grafů představuje vývoj zatížení procesorů na jednotlivých uzlech databázového klastru během testu.



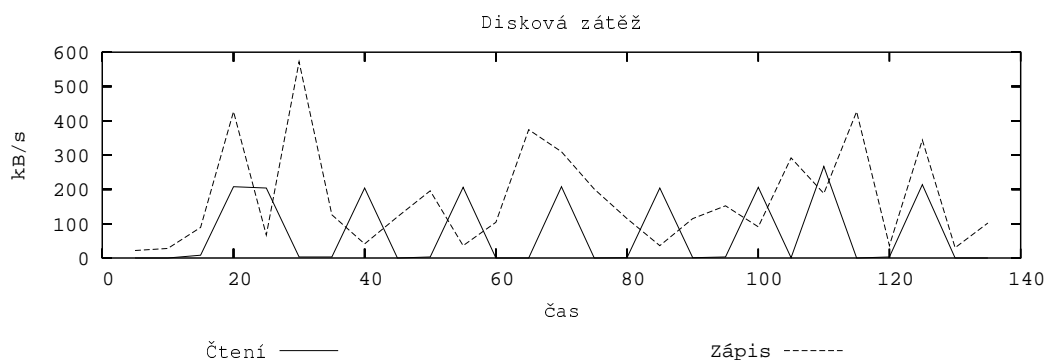
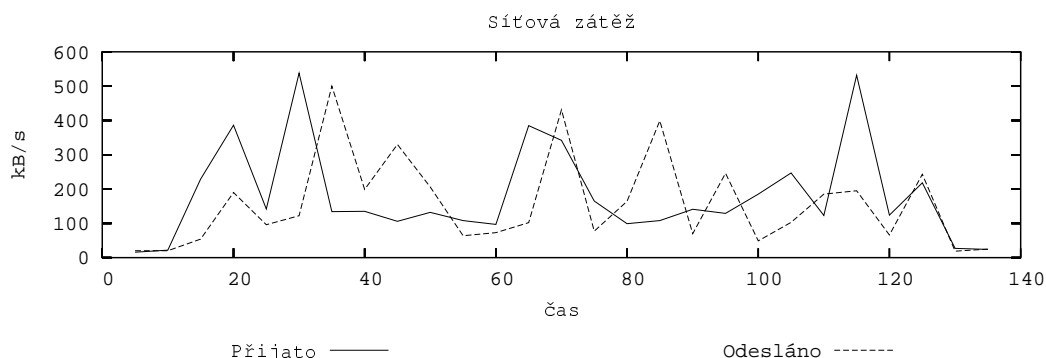
Síťová zátěž je v případě aplikačního klastru tvořena součtem zátěže způsobené komunikací mezi aplikačním klastrem a testovacím webovým klientem, stejně jako mezi uzly aplikačního a databázového klastru.



Následující čtyři grafy představují postupně procesorovou a síťovou zátěž na obou uzlech databázového klastru.



Poslední a zřejmě nejdůležitější test ukazuje zátěž na serveru síťového blokového zařízení, který poskytuje data oběma uzlům databázového klastru pomocí síťového rozhraní. Jak je vidět z průběhu měření, při velkém výpočetním zatížení databázového klastru nedochází k přetížení a to ani na síťové ani na samotné diskové části datového serveru.



Grafy jednoho konkrétního testu jsou na závěr doplněny o graf zobrazující průběh síťové zátěže datového serveru při různých testech a konfiguracích, které jsou od sebe navzájem odděleny kratšími přestávkami. Tyto grafy ukazují, že ani v jednom směru nedošlo při různých testech k takové zátěži, která by byla schopná síťové rozhraní s teoretickou rychlostí 100 Mbit/s přetížit.

