VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Žiniatinklinių tinklalapių ir DBVS analizavimas bei optimizavimas

Mokslinis tiriamasis darbas

Atliko: Erikas Matijošius

Darbo vadovas: Dr. Margarita Kazakevičiūtė

Recenzentas: Asist. Linas Būtėnas

Vilnius
2008

Turinys

Sutartinių terminų sąrašas

- DBVS – Duomenų Bazių Valdymo Sistema;
- ASP.NET - (*Active Server Page*) aktyvaus serverio puslapis, NET(Network) – tinklas, programavimo kalba, skirta svetainių kūrimui;
- PHP – (*Hypertext Preprocessor*) dinaminė programavimo kalba skirta svetainių kūrimui;
- SQL - (Structured Query Language) struktūrizuota užklausų kalba, skirta aprašyti duomenis ir manipuliuoti jais reliacinių duomenų bazių valdymo sistemose
- HTML - (*Hypertext Tranfer Markup Language*) - tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete;
- XML (*eXtensible Markup Language*) - bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba;
- WAP (*Wireless Application Protocol*) – belaidžių aplikacijų protokolas. Yra atviras tarptautinis standartas sistemoms, kurios naudoja belaidį ryšį;
- WAN – (W*ide Area Network*) plataus diapazono tinklai;
- GUI – (*Graphical User Interface*) grafinė vartotojo sąsaja;
- CDM - (*Custom Development Method*) programuotojų pagalbininkas kūrimo stadijos metu.
- Žiniatinklis – (*World Wide Web, WWW*) pasaulinis informacijos tinklas.
- MSDN – (*Microsoft Developer Network*) Microsoft kompanijos pagalbininkas(tarpininkas) vartotojams.
- ODBC – (*Open Database Connectivity*) standartizuota taikomosios programinės įrangos programavimo sąsaja prisijungimui prie duomenų bazių.
- Caching(memory) – atminties podėliavimas.
- Batch queries – serijinės užklausos.
- Connection pooling – duomenų bazių prisijungimų podėliavimas(caching) duomenų bazės lygmenyje.

Anotacija

Tiriamajam moksliniam darbui buvo keliami reikalavimai sukurti tinkamiausią žiniatinklio kūrimo modelį vidutiniam vartotojui. Šiam modeliui buvo išanalizuoti ir pritaikyti labiausiai tinkantys greičio optimizavimo būdai. Pasiektas rezultatas tai – optimaliausias žiniatinklio kūrimo modelis kartu su tinkamiausiais optimizavimo metodais. Internetinis puslapis www.supermodels.lt buvo kuriamas, remiantis išanalizuotomis žiniatinklinių puslapių kūrimo ir DBVS technologijomis. Šiuo metu www.supermodels.lt yra patalpintas interneto serveryje. Mokslinio darbo eigoje sukurtas produktas yra sėkmingai ir sparčiai veikiantis puslapis tenkinantis visus vartotojų greičio poreikius.

Santrauka

Aprašytos labiausiai paplitusios žiniatinklių kūrimo technologijos : ASP.NET, PHP, Python, Ruby. Išanalizuotos ir aprašytos DBVS, kurios yra dažniausiai taikomos žiniatinkliniuose tinklalapiuose. Parinkta optimaliausia žiniatinklinių tinklalapių kūrimo technologijos ir DBVS sąveika PHP+MySql. Remiantis šia sąveika sukurtas puslapis www.supermodels.lt. Nustačius žiniatinklinio puslapio struktūrą, pradėtas greičio optimizavimo procesas. Išanalizuoti ir aprašyti: „podėliavimas“, „connection pooling“ , „serijinės užklausos“, grafikos bei sukompiliuotų bitų kodų optimizavimo metodai. Naudojant stebėjimo įrankį JBlizPro 5.1 buvo taikomi visi greičio optimizavimo metodai ir analizuojant rezultatus buvo daromos optimizavimo išvados. Galiausiai nustatyti optimaliausi optimizavimo metodai vidutiniam žiniatinklio tinklalapio poreikiui.

Summary

In this paper there were described the most popular WEB development technologies: ASP.NET, PHP, Python, Ruby with different DBMS (database management systems). The WEB page www.supermodels.lt was created using the most optional interaction - PHP+MySgl. Then the page optimization process started. The page optimization process started describing and sifting speed optimization methods: “Catching”, “Connection pooling”, “Batch queries”, graphic and bite code optimizers. All these methods were implemented in www.supermodels.lt page. The speed optimization conclusions were made using JBlizPro 5.1 watchouts results. Finally the most optimal optimization methods were found to make moderate WEB page.

Įvadas

Hiperteksto idėja pasauliui pateikta 1945 m., kai Vannevaras Bushas pristatė konceptualų prietaiso, skirto efektyviam informacijos kaupimui ir valdymui, modelį. 1965 m. Tedas Nelsonas sugalvojo terminą „hipertekstas“, kurį apibūdino kaip netiesinį lankstų informacijos pateikimo būdą, kai informacijos vienetai (kompiuterio ekrano puslapiai) dėstomi nenuosekliai, bet tam tikra autoriaus pasirinkta sistema. 1987 m. Tim Berners-Lee ir Robert Cailliau sukūrė HTML kalbą (ang. *Hypertext Transfer Markup Language*) – hiperteksto aprašymo kalbą. O kai 1987 m. internetas tapo atviras visiems vartotojams - atsirado populiariausia interneto paslauga — žiniatinklis. O pats žiniatinklio tinklalapis (ang. *Web Page, Page*) — tai žiniatinklio dokumentas parašytas hipertekstu.

Šiame darbe bus analizuojamos žiniatinklio tinklalapių kūrimo technologijos. Kadangi šiuolaikiniai tinklalapiai tampa vis labiau dinamiški ir neatsiejami nuo duomenų kaupimo technologijų, tai kartu bus analizuojamos ir skirtingos duomenų bazių valdymo sistemomis. Kitas etapas – optimizavimas. Išanalizavus žiniatinklio tinklalapio bei duomenų bazių valdymo sistemas, bus taikomi optimizavimo metodai ir daromi praktiniai tyrimai. Iš gautų tyrimų rezultatų bus pateikiamos optimizavimo išvados.

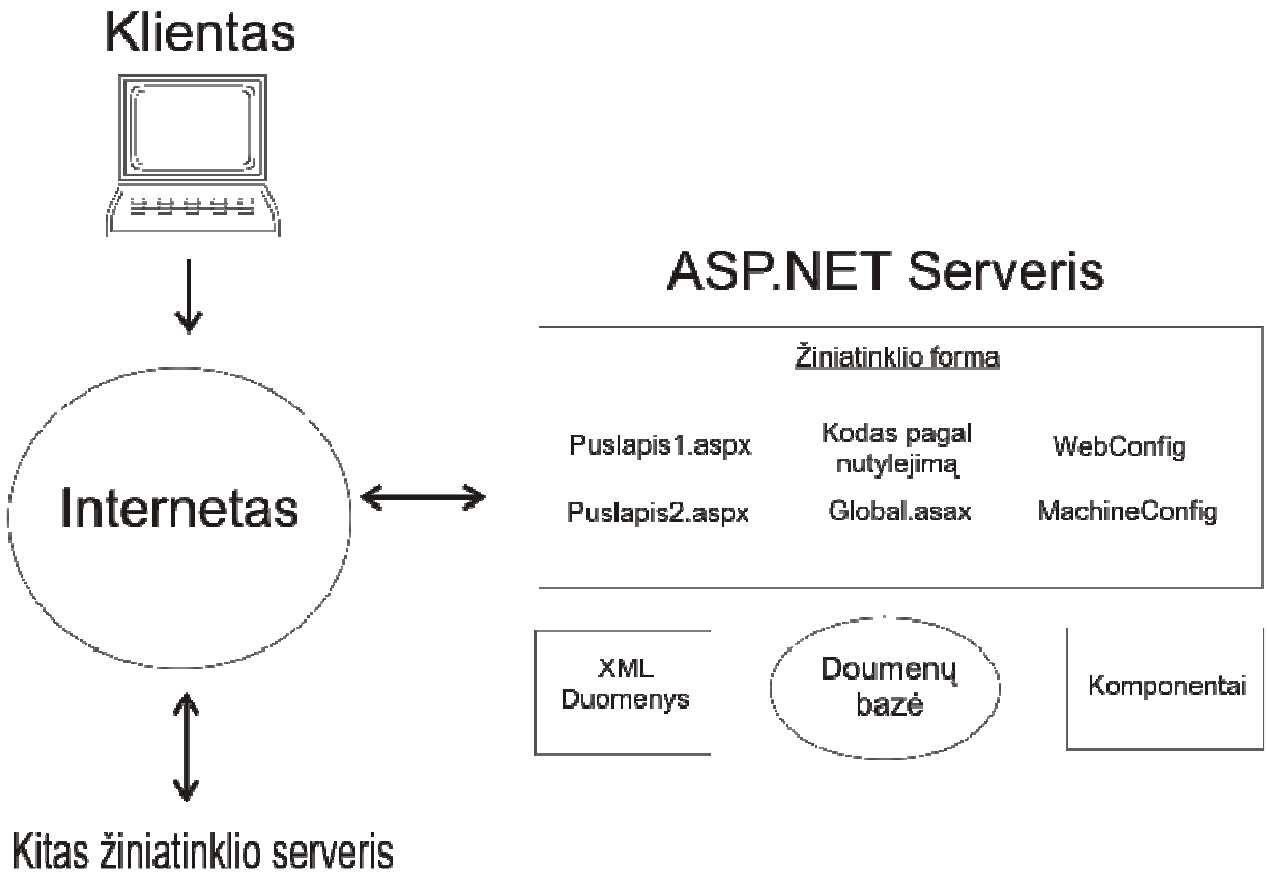
Žiniatinklių kūrimo technologijos

I. ASP.NET technologija

ASP(Active Server Page) – aktyvaus serverio puslapis, NET(Network) – tinklas. ASP.NET yra programavimo kalbų griaučiai, sukurti žiniatinklinių programų(WEB application) kūrimui. Jos pagrindas yra .NET platforma. ASP.NET yra pagrįsta objektiniu programavimu, todėl turi tokius bruožus kaip paveldėjimą, inkapsuliaciją ir polimorfizmą. Pagrindinis ASP.NET technologijos komponentas yra žiniatinklio(Web) forma. Žiniatinklio forma tai - internetinis puslapis kurį mato žiniatinklio vartotojai. Juolab, kad ASP.NET žiniatinklio programa turi daugiau nei vieną žiniatinklinę formą, kuri savo ruožtu yra dinaminis puslapis, kuris turi priėjimą prie serverių resursų. ASP.NET taip pat turi technologiją, kuri yra reikalinga kurti XML žiniatinklius. Programų kūrėjai daug metu naudojo ASP technologiją kurti dinaminius puslapius. Todėl panašiai kaip ir ASP,

ASP.NET yra patobulinimas ankstesnių programavimo griaučių, kurie turi savyje turtingesnį pagrindą, dinamiškesnius bei labiau suasmenintus puslapius[1].

ASP.NET žiniatinklio programa[1] susideda iš skirtingų dalių ir komponentų. Šie žiniatinklio komponentai yra pavaizduoti schemoje:



1. ASP.NET veikimo diagrama

Žiniatinklio forma (puslapis su *.aspx plėtiniu) - suteikia vartotojo sąsają su žiniatinklių programomis)

Kodas pagal nutylėjimą - programos kodas susietas su žiniatinklio forma, kurį atlieka serveris

Nustatymo bylos – XML bylos, kurios apibrėžia žiniatinklio programos bei žiniatinklio serverio nustatymus pagal nutylėjimą. Kiekviena žiniatinklio forma turi vieną **Web.config** nustatymų bylą. O žiniatinklio serveris savo ruožtu vieną **Machine.config** bylą.

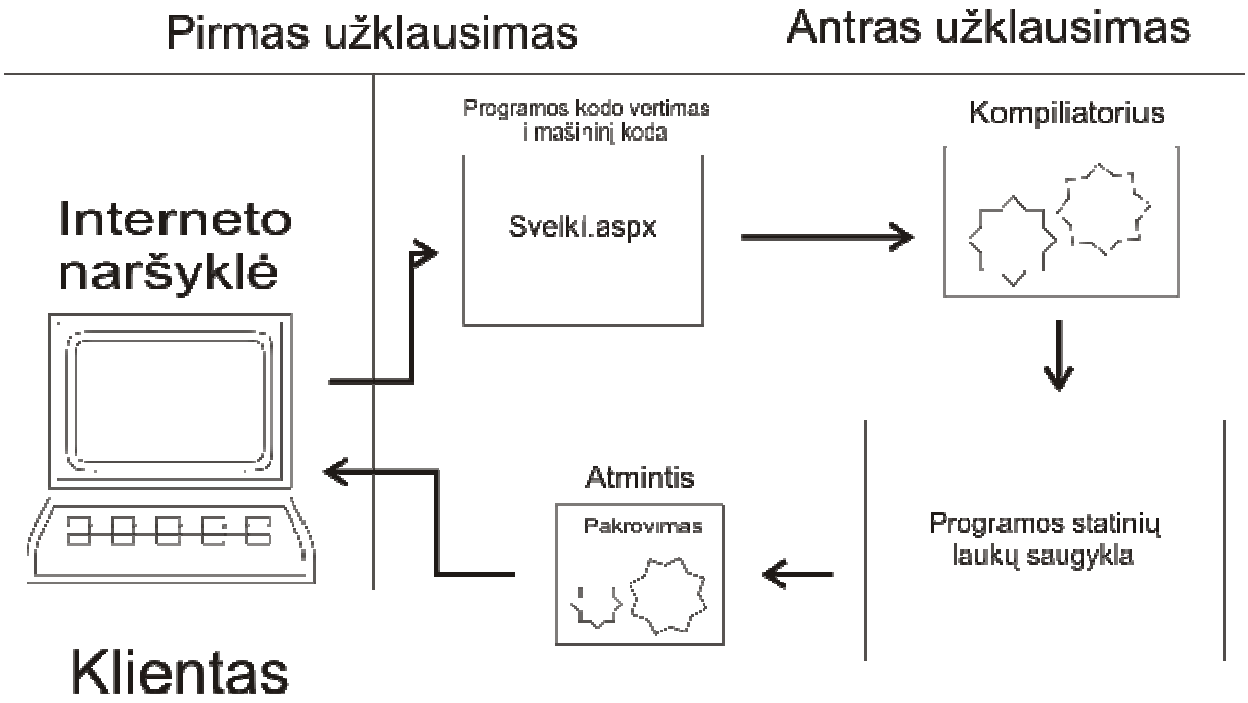
Global.asax byla – kontrolinė byla, susidedanti iš kodo, kuris priklauso nuo skirtingų programos lygio užklausų.

XML žiniatinklio serviso sąsaja – suteikia galimybę žiniatinklio programoms siusti ir gauti duomenis iš XML žiniatinklio serviso.

Duomenų bazės prisijungimo galimybė – suteikia žiniatinklio programoms keistis duomenimis su duomenų šaltiniais.

Talpykliniai mainai – suteikia galimybę žiniatinklio programoms grąžinti žiniatinklio formas ir duomenis daug greičiau po pirmojo užklausimo.

Pats ASP.NET žiniatinklio vykdymo modelis [1] atrodo taip:



2. ASP.NET žiniatinklio vykdymo modelis

Pirmas užklausimas. Klientas užklausia žiniatinklio puslapį pirmą kartą:

1. Kliento naršyklė pareiškia serveriui “GET HTTP” užklausą
2. ASP.NET analizatorius verčia pradini kodą į mašininį(Parser).
3. Jei kodas nebuvo prieš tai sukompiliuotas į dinaminę nuorodų biblioteką, ASP.NET pasitelkia kompiliatorių.
4. Nustatytas laiko tarpas (RUNTIME) pakrauną ir įvykdo Microsoft tarpines kalbos (MSIL) kodą.

Antras užklausimas. Klientas užklausia to pačio žiniatinklio puslapo antrą kartą:

1. Kliento naršyklė pareiškia serveriui “GET HTTP” užklausą
2. Nustatytas laiko tarpas (RUNTIME) pakrauną ir iškarto įvykdo MSIL kodą, kuris buvo sugeneruotas pirmo paleidimo metu.

1.1 ASP.NET technologijos privalumai

- Kadangi šios technologijos platformos yra .NET griaučiai, programų kūrimas galimas visomis programavimo kalbomis kurias palaiko .NET : VISUAL BASIC, VISUAL C#, VISUAL J#, VISUAL C++.
- Tradiciniai internetiniai puslapiai atliekantys pagrindines užduotis naudoja “skriptus” ir atlieka viską nuosekliai. O internetiniai puslapiai, sukurti naudojant ASP.NET technologiją, priešingai nei tradiciniai, dirba lygiagrečiai - vienu metu leidžia atlikti serverio užklausą, prisijungus prie duomenų bazės ir tuo pačiu generuoja žiniatinklį.
- Kadangi ASP.NET žiniatinklio formą nepriklauso nuo kliento naudojamo “skripto”, tuo pačiu ji taipogi nepriklauso nuo kliento naudojamos naršyklės ir operacinės sistemos. Šis nepriklausomumas leidžia kurti žiniatinklio formas, kurios yra matomos bet kuriame įrenginyje turinčiame priėjimą prie interneto.
- Žiniatinklių kūrimas naudojant ASP.NET technologiją yra toks pat kaip ir Windows programų kūrimas.
- Naudojant Microsoft® Visual Studio® programų paketą galimas ASP.NET žiniatinklio programos lūžio taškų stebėjimas bei kiekvienos kodo dalies analizavimas paleidimo metu.
- Naudojant Microsoft® Visual Studio® programų paketą galima dinamiška pagalba generuojant kodą, siūlant leistinų klasių, metodu, funkcijų, įvykių bei kintamųjų pavadinimus.
- ASP.NET turi didžiulę MSDN pagalbos ir dokumentacijos biblioteką[1].

1.2 ASP.NET technologijos trūkumai

- Kadangi ASP.NET yra Microsoft® kompanijos produktas, svetainių kūrimas galimas tik naudojant Microsoft® žiniatinklio serverį.
- Duomenų perdavimas ir apsikeitimas su (būtent) Oracle duomenų serveriu yra lėtesnis lyginant su PHP kalba.
- ASP.NET nėra atviro kodo programa, taigi visas programų paketas yra ganėtinai brangus[1].

II. PHP technologija

PHP (hypertext preprocessor) – plačiai paplitusi dinaminė interpretuojama programavimo kalba, sukurta 1997 m. ir specialiai pritaikyta interneto svetainių kūrimui. PHP sintaksė panaši į daugelį struktūrinių kalbų , ypatingai į C bei Perl. PHP kalba yra atviro kodo ir tai yra viena priežasčių, dėl ko kalba yra nors ir nesudėtinga, bet gana lanksti – veikia daugumoje operacinių sistemų, palaiko nemažai reliacinių duomenų bazių bei veikia su dauguma interneto serverių –CGI, FastCGI, ISAPI ir kitais protokolais. Nors ir PHP yra dažniausiai naudojama interneto puslapių kūrimui, PHP yra labai galingas įrankis atlikti kitas funkcijas komandinėje eilutėje. PHP skriptai yra interpretuojami ir įvykdomi serverio pusėje. Pagrindinis PHP privalumas prieš CGI buvo tai, kad PHP skriptas galėjo būti lengvai įterpiamas į HTML kodą, kaip CGI reikėjo visą HTML išvedinėti iš CGI skripto. Kuo toliau tuo rečiau HTML'as yra maišomas su HTML'u didesniuose projektuose, tam naudojami šablonai. PHP skirtumas nuo JavaScript yra tai jog PHP skriptai yra atliekami serverio pusėje o vartotojui yra gražinamas rezultatas (HTML'u ar XML'u). Rezultatą vartotojas mato savo naršyklėje. Tuo tarpu kai JavaScript yra pilnai perduodamas kompiuteriui ir atliekamas klientinėje dalyje. PHP sintaksės pagrindai yra paimti iš:C, Perl, Java, todėl programavusiems šiomis kalbomis yra ypatingai lengva pradėti programuoti PHP[2].

2.1 PHP technologijos privalumai

- php yra visiškai nemokamas
- php yra visiškai „cross“ platforminis (veikia įvairiose operacinėse sistemose: Win, *nix, MacOS, Solaris, HP-UX, AIX ir t.t.)
- php yra atviro kodo projektas todėl jį rengia ir tobulina didelė grupė žmonių, todėl iškilusios klaidos yra greitai ištaisomi
- PHP sparčiai plečiasi
- kaip ir „cross“ platforminis jis veiki ir ant daugelio WEB serverių: Apache, IIS, PWS, OmniHTTP, BadBlue ir t.t.
- išmokti PHP programavimo pagrindų yra labai lengva
- pasižymi dideliu greičiu serverio pusėje, bei dirbant su duomenų bazėmis
- nedideliuose projektuose PHP jūs paprastai galite įterpti į savo HTML'ą

- PHP tobulintojai ir iniciatoriai prirašė praplėtimų kiekvienam gyvenimo atvejui
- kadangi PHP programuotojų yra be galo daug, daugumą jau parašytų skriptų galima rasti nemokamuose internetiniuose puslapiuose.

2.2 PHP technologijos trūkumai

- PHP yra interpretatorius,
- perskaitomi PHP „source“-ai (kodai), nėra jokio konfidencialumo,
- php-gtk „source“-ai vis dar negali būti atkoduojami.

III. Python technologija

Python yra interpretuojama, interaktyvi programavimo kalba sukurta Guido van Rossumo 1990 metais. Pirmiausiai ji buvo scenarijų kalba AmoebaOS operacinei sistemai. Python dažniausiai lyginama su Perl, Java ir Ruby. Python kuriama kaip atviro kodo projektas. Python yra daugialypė programavimo kalba – ji leidžia naudoti keletą programavimo stilių: objektinį, struktūrinį, funkcinį, aspektinį. Python naudoja dinaminį tipų tikrinimą.Python kūrėjų tikslai buvo sukurti kalbą, kuri yra lengvai skaitoma, išraiškinga, išreikštinė, paprasta. Nors pradžioje ji buvo kuriama kaip scenarijų kalba, dabar ji naudojama ir dideliems programiniams projektams. Tai dinaminė aukšto lygio programavimo kalba. Python yra interpretuojama kalba, panašiai kaip Java (programa pirma kompiliuojama į baitinį kodą, o paskui virtuali mašina tą kodą interpretuoja). Skirtingai nuo Java, Pytohn kompiliavimas atliekamas automatiškai, paleidus interpretatorių[17]. Kaikurios Python programavimo kalbos savybės:

- Viskas yra objektai; galima sukurti klases, praplečiančias standartinius duomenų tipus.
- Klases ir funkcijas lengva dokumentuoti naudojant dokumentacijos eilutes („docstrings“).
- Dokumentacija pasiekiama ir programos darbo metu.
- Klasės ir funkcijos taip pat yra pirmos klasės objektai.
- Galimas paveldėjimas iš keleto klasių.
- Galima kurti anonimines funkcijas (nors tai planuojama panaikinti vėlesnėse versijose).
- Leksinis apgaubimas.

- Labai svarbus kodo išdėstymas (indentation).
- Modulių sistema.

3.1 Python technologijos privalumai

- Dėl indentacijos, galimas praktiškai vienintelis būdas (skiriasi tik tarpų/tabuliacijos ženklų vartojimas) parašyti kodą, todėl lengva dirbti grupėse
- Kodas gali būti kompiliuojamas į vidinę formą, kas leidžia greičiau įkrauti daug kartų naudojamus modulius ir pan.
- Greitas programuotojų darbo ciklas

3.2 Python technologijos trūkumai

- Python yra interpretatorius,
- perskaitomi Python sourc'ai, nera jokio konfidencialumo.

IV. Ruby technologija

Ruby - interpretuojama programavimo kalba greitam ir lengvam objektiškai orientuotam programavimui. Ruby ypač gerai tinka tekstiniams failams apdoroti. Tai paprasta, mobili ir lengvai plečiama kalba. Dažnai maišoma su Ruby On Rails programavimo terpe, kuri iš tikro pati naudoja Ruby. Ši programavimo kalba yra gerai pritaikyta Windows platformoje. Netgi daugiau: Ruby puikiai veikia visose pagrindinėse operacinėse sistemose: Windows, Linux (Unix), MacOs, BeOs, OS/2 ir t.t. Ruby yra interpretuojama programavimo kalba, o tai reiškia, kad kaip ir daugelis kitų kalbų (Python, Perl, Php, *.bat bylos) Ruby kodas nėra kompiliuojamas ir jį paleidžia interpretatorius. Ruby yra objektiškai-orientuota programavimo kalba: jei tokiose kalbose kaip Python’as arba Java yra objektų “pakaitalų”, tai Ruby faktiškai yra pilnai objektiškai- orientuota (objektiškumas ir buvo viena iš šios kalbos parašymo priežasčių). Ruby yra nauja grynai objektinė

programavimo kalba. Nors nedaugelis girdėjo apie ją Vakaruose, ji tampa labai populiari Japonijoje - populiarumu ji jau aplenkia Python. Ruby persipina patys geriausi pačių geriausių programavimo kalbų elementai ir sudaro vieningą visumą[15].

4.1 Ruby technologijos privalumai

- Ruby susilieja klasikinės objektinio programavimo kalbos *Smalltalk* objektinės galimybės ir interpretuojamą kalbą kaip *Perl* išraiškingumas ir patogumas.
- Ruby programos yra kompaktiškos, tačiau tuo pačiu metu lengvai perskaitomos ir suprantamos, keliomis eilutėmis galima daug padaryti, netgi neaukojant programos įskaitomumo.
- Kalbos sintaksė ir semantika yra labai paprastos. Kalboje nėra jokių "išimčių", kurias reikėtų prisiminti. Pavyzdžiui, skaičiai, klasės, *nil* - viskas yra objektai kaip ir bet kas kita.
- Kai išmokstami Ruby pagrindai, lengva teisingai atspėti kaip daryti naujus dalykus.
- Ruby išlaisvina nuo nuobodaus kompiliatoriaus sąlygų tenkinimo. Ruby turbūt vienintelė kalba, kuri leidžia susikoncentruoti prie algoritmo kūrimo, nevargindama smulkiomis sintaksės detalėmis.
- Ruby yra atviro kodo ir ją laisvai galima gauti tiek programavimui, tiek programų vykdymui.
- Ruby, kitaip nei kitos naujos kalbos, nepririša jūsų prie kokios nors platformos ar gamintojo.
- Ruby galite pasileisti ant Linux, Unix, Microsoft Windows arba specializuotų operacinių sistemų kaip BeOS ir kitų.

4.2 Ruby technologijos trūkumai

- Ruby technologijos pažinimas yra neįprastas pereinant nuo kitos programavimo kalbos
- Ruby yra gana nauja technologija, todėl atviro kodo programos dar nėra labai jai paplitusios.

V. Skirtingų DBSV analizavimas

Yra daugybė DBVS. Jas kuria įvairios firmos, kurios šias sistemas platina tarp vartotojų kaip atskirus, savarankiškus programinės įrangos vienetus. Tokie vienetai vadinami paketais. DB valdymo paketai skiriasi vieni nuo kitų kokybinėmis ir kiekybinėmis (techninėmis) charakteristikomis. Kokybinės DBVS charakteristikos - tai apimtis (sudėtingumo laipsnis), taikymo sritis, funkcionavimo bazė, darbo patogumas. Kiekybinės charakteristikos - tai, pvz., leistina apdorojamos DB apimtis, DB lentelių skaičius, lentelės apimtis ir pan. Vienas iš pagrindinių sistemų klasifikavimo kriterijų yra sistemų apimtis. Pagal savo apimtį DBVS galima suskirstyti į dideles (labai sudėtingas) sistemas, vidutines (mažiau sudėtingas) sistemas ir mažas sistemas. Didelės DBVS yra šios: Oracle, MySQL, MsSQL, PostgreSQL, Sybase, DB2, Ingres. Vidutinių sistemų yra daugiau. Pagrindinės yra šios: Access, Paradox, Clipper, Clarion, dBase, FoxPro ir kt. Dar daugiau yra mažų DBVS - jų šiuo metu suskaičiuojama daugiau kaip 50.

Daugumos šiuolaikinių DBVS funkcionavimo bazė yra personaliniai kompiuteriai. Visos didelės sistemos, be to, dar gali funkcionuoti ir minikompiuteriuose bei super-kompiuteriuose. Paprastai visi DB valdymo paketai yra orientuojami į darbą Windows ir Unix tipo operacinėse sistemose, būtent, Windows NT, Windows 95 ir pan. Dideli paketai gali veikti ir operacinėse sistemose, skirtose superkompiuteriams, pvz., VAX VMS, OS/2.

Kalbant apie darbo patogumą pažymėtina, jog vienose sistemose siekiama suteikti kuo lankstesnes, vaizdesnes grafinės sąsajos galimybes, o kitose sistemose pagrindinis dėmesys skiriamas tam, kad būtų kuo daugiau manipuliavimo duomenimis priemonių. Pirmosios sistemos daugiau orientuojamos į vadinamuosius galutinius vartotojus (ne į programuotojus). Antrosios sistemos orientuojamos į kvalifikuotus vartotojus (programuotojus). Pateikiami trumpi skirtingų duomenų bazių palyginamieji kriterijai:

DBVS	Lentelių kiekis	Lentelės dydis	Lentelės plotis	Laukų kiekis	Lauko plotis
Oracle 9i	---	64 TB	255000 baitai	1000	4 GB
PostgreSQL	---	32 TB	priklauso nuo laukų pločių	250-1600	1 GB
MySQL 5.0	---	64 TB	priklauso nuo laukų pločių	250	2 GB
SQL Server 2005	$2 \cdot 10^9$	32 TB	8036 baitai	1024	255 baitai
DB2 v9.0	priklauso nuo OA dydžio	64 GB	32677 baitai	1012	4005 baitai
Informix	$477 \cdot 10^6$	64 TB	32767 baitai	2767	32767 baitai
Sybase	$2 \cdot 10^9$	priklauso nuo OA dydžio	priklauso nuo OA dydžio	250	1962 baitai
Access	---	1 GB	---	255	255 baitai

3 lentelė. Duomenų bazių valdymo sistemų kiekybinės charakteristikos(2006m)

Didelės apimties, kompleksinės DBVS paprastai reikalingos stambioms organizacijoms, kompanijoms, bankams. Vidutinės sistemos tinka smulkesnėse įmonėse, įstaigose, firmose. Jos gali būti naudojamos ir atskiruose stambių organizacijų padaliniuose, filialuose. Kitas labai svarbus aspektas yra duomenų bazių ir jų palaikymo kainą. Trumpai apžvelgsime visų paminėtų duomenų bazių kainų skirtumus tarpusavyje:

Pavadinimas	Kaina
Oracle 9i database standard edition	$15.000,00
PostgreSQL	Atviro kodo programa
MySQL 5.0	Atviro kodo programa
MsSQL Server 2005	$7.500,00
DB2 v9.0	$4.999,99
Informix	----------
Sybase	$299,99
Microsoft Access 2003 Win32	$176,35 - $229,99

4. DBVS kainų palyginimas

Analizuojant visų DBSV privalumus ir skirtumus, reikėtų giliau apžvelgti kiekvienos duomenų bazės ypatybes:

Oracle – yra labai išplėsta duomenų bazės sistema su savo praplėtimais, įrankiais bei savybėmis. Skirtingai nei MySQL, ji palaiko visus standartinius SQL kalbos konstrukcijas. Dar daugiau, Oracle praplečia SQL kalbą, pridedama savo nestandartines SQL konstrukcijas. Tačiau visi šie privalumai, praplėtimai ir įrankiai padaro savo – Oracle kainos atžvilgiu yra brangiausias produktas.Oracle palaiko šias kompiuterines platformas:

- Linux x86
- Linux (x64)
- Linux zSeries
- Linux Itanium
- Linux POWER
- Microsoft Windows (x86)
- Microsoft Windows (x64)
- Microsoft Windows (64-bit Itanium)
- Solaris (x86)
- Solaris SPARC (64-bit)
- AIX
- HP-UX PA-RISC
- HP-UX Itanium
- HP Tru64 UNIX
- HP OpenVMS Alpha
- IBM z/OS
- Mac OS X Server

Kitaip tariant Oracle palaiko visas operacinių sistemų platformas, todėl Oracle – viena didžiausių ir dažniausiai naudojamų reliacinių duomenų bazių valdymo sistemų. Oracle yra labai plačiai naudojama bankinėse, finansinėse ir mokslinėse sistemose duomenų saugojimui, apdorojimui ir analizei. Oracle programinę įrangą kuria Oracle kompanija[13].

PostgreSQL – atviro kodo, reliacinių duomenų bazių valdymo sistema. Ši sistema aktyviai kuriama daugiau nei 15 metų. Veikia daugelyje operacinių sistemų, įskaitant Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) ir Windows. Be standartinių SQL galimybių, palaiko skirtingų tipų indeksavimą, išsaugotas procedūras ir trigerius įvairiomis kalbomis, rafinuotą užklausų planavimą bei optimizavimą, didelių binarinių objektų laikymą bei daugybę kitų funkcijų. Turi vidinį programavimo interfeisą, pritaikytą dirbti su C, C++, Java, Perl, Python, Ruby, Tcl. Yra sukurtos ODBC(atviro duomenų bazių prisijungimo) tvarkyklės[11].

MySQL – yra labai greita, turinti didelį našumą bei gerai charakterizuota duomenų bazės sistema. MySQL – viena iš reliacinių duomenų bazių valdymo sistemų, palaikanti daugelį naudotojų, dirbanti SQL kalbos pagrindu. MySQL yra atviro kodo programinė įranga, vystoma ir palaikoma švedų kompanijos „MySQL AB“, kurios įkūrėjai – švedai David Axmark, Allan Larsson ir suomis Michael „Monty“ Widenius. Ši produktas gali be vargo palaikyti didelį duomenų kiekį ir dirbą su dauguma standartinių SQL kalbos konstrukcijų[14]. MySQL yra populiariausia atviro kodo reliacinių duomenų bazių valdymo sistemų, dirbanti SQL kalbos pagrindu. Ji dažnai pasirenkama programuojant internetines svetaines, populiarumas tiesiogiai susijęs su PHP kalbos populiarumu, nes dažniausiai naudojama šių dviejų priemonių kombinacija.

SQL Server – yra Microsoft kompanijos reliacinės duomenų bazes sistema. Ji yra visapusis, integruotas ištisinio duomenų perdavimo sprendimas, leidžiantis vartotojams naudoti saugią, patikimą ir produktyvią duomenų ir verslo tyrimo platformą. SQL Server informacinių technologijų profesionalams ir asmenims, dirbantiems su informacija, turi efektyvius žinomus įrankius, kuriuos naudojant paprasčiau kurti, diegti, valdyti ir naudoti platformoje esančius duomenis ir analitines programas, tokias kaip mobilieji įrenginiai bei duomenų sistemos. SQL Server visapusis priemonių rinkinys, suderinamumas su esamomis sistemomis ir įprastų duomenų automatizavimas, skirtas visų duomenų sprendimui bet kokio dydžio įmonėje. Didžiausias šios duomenų bazės minusas yra tai, jog ji yra Microsoft kompanijos produktas, ji yra pririšta prie Windows operacines terpės[10].

DB2 – IBM kompanijos produktas. Tai visapusišką duomenų bazių valdymo sistemą, kuri leidžia lengviau ir paprasčiau kurti bei vykdyti taikomąsias programas, sprendimus. DB2 v9.0 duomenų bazių valdymo sistema, papildyta didesniu papildomų paslaugų paketu, kuris specialiai sukurtą programinei įrangai vystyti, ją talpinti, perkelti ir įterpti į anksčiau ruoštas taikomąsias programas.

Nemokama duomenų bazių valdymo sistema DB2 veikia 32 ir 64 bitų procesorių architektūros "Windows" ir "Linux" operacinėse sistemose[12].

Access – galimybės yra labai apkarpytos, lyginant su kitomis duomenų bazėmis, kurios palaiko SQL kalbą. Juolab ši duomenų bazė sunkiai susitvarko su dideliu duomenų kiekiu. Kadangi Access yra Microsoft kompanijos produktas, jis palaiko (GUI) grafinį vartotojo sąsajos "VisualBasic" braižą, bet tuo pačiu ir pririša vartotoja prie Windows platformos. Access duomenų bazė sukurta taip, kad ji butu kuo patogesnė vartotojui su ja naudotis. Viskas supaprastinama iki aukšto lygio ir pateikiama plati naudotojo instrukcija. Taip šio produkto vartojimas tampa labai patogus ir lengvas[7].

Kitos Duomenų bazių valdymo sistemos. **Informix** 2001.04.28 nusipirko IBM kompanija už 1 milijardą JAV dolerių. Tai metais Informix galutinai išnyko iš DB varžytinių Bet ji liko tarp "penkių didžiausių duomenų programinės įrangos bazių kūrėjų pasaulyje, kurios produktais naudojasi daugiau kaip 100 tūkstantis klientų. Šiame darbe ji pateikiama tik kaip statistiška duomenų bazė. **Sybase** – taip pat jau beveik išnykusi iš rinkos duomenų bazės valdymo sistema. Tačiau 10 metų bėgyje Sybase konkuravo ir mažai kuo nusileido tokioms didelėms kompanijoms kaip Oracle ar DB2. Šiame darbe ji taip pat yra parinkta tik kaip palyginimo kriterijus. Šiuo metu Sybase kompanija orientuojasi daugiau į atskirų IT produktų gamybą.

Informix, Sybase, ir Access šiuo atžvilgiu negali konkuruoti prieš likusias duomenų bazes dėl savaime aiškių priežasčių(dėl galimybių ribotumo ir naujų vartotojų poreikių nepatenkinimo). O lyginant Oracle ir PostgreSQL peršasi išvada kad PostgreSQL yra apkarpytas Oracle variantas. Apibendrinant galima būtų pasakyti, jog pati praktiškiausia galimybių ir kainos atžvilgiu duomenų bazė yra MySQL Tad logiškiausia, mano atžvilgiu, visus laurus atiduoti MySQL duomenų bazei. MySQL duomenų bazė yra suprojektuota taip, kad būtų išnaudoti visi interneto teikiami privalumai sistemas kompleksiškai ir centralizuotai valdant, aptarnaujant ir atnaujinant iš vartotojo darbinės vietos. Kadangi MySQL yra atviro kodo programa, tai žymiai sumažėja programinės įrangos administravimo ir eksploatacijos kaštai. Manau pasirinkimą lemia kuriamų planų apimtys, sudėtingumai bei vartotojų galimybės.

Pagal šiame darbe išanalizuotą medžiagą galimą būtų pateikti pirmines šio darbo išvadas. Iš pirmo žvilgsnio susumuojant rezultatus, mano nuomonė, būtų, jog optimaliausios ir populiariausias žiniatinklio kūrimo produktas yra PHP technologija. O kas liečia duomenų bazes, tai analizuojant

išvadas galima būtų priskirti visus laurus MySQL duomenų bazei. Dėl šių dviejų technologijų populiarumo, ir yra labai naudinga bei tikslinga pritaikyti optimizavimo metodus būtent šioms technologijoms. O atliekant tyrimus ir pasinaudojant gautais rezultatais, galima bus padaryti galutines magistrinio darbo išvadas.

VI. Bendra žiniatinklių optimizavimo analizė

Žiniatinklinių aplikacijų optimizavimas – kas tai? Nuo pat internetinių puslapių atsiradimo datos, visuomet buvo keliama problema, kaip optimizuoti žiniatinklinius tinklalapius taip, kad jų veikimas būtų kuo kokybiškesnis, spartesnis ir patogesnis.

Tinklalapių patogumas (angl. Usability) yra kokybiškumą atspindinti savybė, kuri parodo vartotojo sąsajos paprastumą ir aiškumą. Terminas „usability“ taip pat apima visus metodus, kuriais siekiama pagerinti vartotojo naršymo patogumą. Šie metodai naudojami kuriant tinklalapio dizainą. Pagal padarytus komercinius tyrimus tinklalapių patogume yra siūloma išskirti šiuos žiniatinklių kokybės elementus[9] :

- **Mokymasis** (kaip lengvai žmogus gali atlikti norimus veiksmus susidūrę su tinklalapiu pirmą kartą);
- **Efektyvumas** (kaip greitai žmonės gali atlikti norimus veiksmus, jau būdami įsigilinę į tinklalapį);
- **Įsiminimas** (kaip greitai žmogus gali pakartoti savo veiksmus, sugrįžęs į tinklalapį po tam tikro laiko);
- **Klaidos** (kaip dažnai klysta lankytojai ir kaip greitai jie gali šias klaidas ištaisyti);
- **Malonumas** (kiek tinklalapio dizainas yra malonus lankytojui).

Yra daugiau kitų svarbių tinklalapio savybių, kurios parodo tinklalapio kokybiškumą. Viena iš jų yra naudingumas, parodanti tinklalapio dizaino funkcionalumą: ar tinklalapio dizainas daro tai, ko tikrai reikia vartotojui? Patogumas ir naudingumas yra vienodai svarbūs: tinklalapio elementas ar objektas turi mažai reikšmės, jei jį patogu naudoti, bet jis yra ne tai, ko reikia vartotojui. Taip pat nėra gerai, jei sistema gali atlikti tai, ką jūs norite, bet jūs negalite atlikti norimo veiksmo dėl to, kad vartotojo sąsaja yra per daug sudėtinga. Norint analizuoti naudingumą, galite taikyti tuos pačius metodus, kaip ir tinklalapių patogumui.

Internete tinklalapių patogumas yra būtina išlikimo sąlyga. Jei tinklalapis yra per daug sudėtingas naudojimui, vartotojai jį palieka. Jeigu tinklalapis aiškiai neparodo, ką jame ras vartotojai, jis paliekamas. Jei vartotojai pasiklysta tinklalapyje, jie palieka jį. Vartotojai palieka tinklalapį, jei informaciją sunku skaityti arba ji neatsako į pagrindinius vartotojų klausimus. Tinklalapyje nepateiksi jo skaitymo instrukcijų – yra milijonai kitų tinklalapių, iš kurių gali rinktis vartotojai, ir išėjimas iš svetainės yra pirmasis veiksmas, ką jie padaro. Pirmoji el. komercijos taisyklė teigia, kad jeigu žmogus negali rasti produkto, jis jo negali ir nusipirkti. Užsienio tinklalapių patirtis rodo, kad yra išleidžiama maždaug 10% tinklalapio dizaino biudžeto jo patogumo analizei – tai vidutiniškai pagerina tinklalapio lankomumo rodiklius du kartus[9].

Yra daug metodų analizuoti tinklalapių patogumą, bet pagrindinis iš jų ir pats naudingiausias yra vartotojų atliekami tinklalapio bandymai. Šis testavimas turi tris dalis:

- Parenkami atitinkantys tam tikrus reikalavimus vartotojai, pvz. El. komercijos vartotojai
- Jie prašomi atlikti tam tikras užduotis
- Stebima ką veikia vartotojai, kur jiems sekasi, kur jie susiduria su problemomis.

Tinklalapių patogumas užima nemenką reikšmę tinklalapių kiekviename dizaino kūrimo etape. Prieš kuriant naują dizainą, yra testuojamas senasis, išskiriant stipriąsias puses, kurias reikėtų palikti arba pabrėžti, ir silpnąsias puses, kurios sukelia nepatogumų[9].

1. Padaromi popieriniai variantai vieno ar kelių naujojo dizaino varianto.
2. Atrenkamos geriausios idėjos, pakartojant bandymus kelis kartus.
3. Patikrinamas dizainas pagal teorines nuostatas ir ankstesnius bandymus.
4. Atliekami tyrimai, įdiegiamas galutinis dizainas. Išbandomas dar kartą. Sunkiau aptinkamos problemos dažniausiai atrandamos diegimo metu.

Kitas begalo svarbus dalykas yra tai, kad lankytojai yra suinteresuoti, kad tinklalapis būtų netik patogus ir patrauklus vartojimui, bet ir greitai pasiekiamas. Šis rezultatas yra gaunamas tik redaguojant techninę žiniatinklio kodo dalį, optimizuojant jo „veikimo“ greitį.

VII. Greičio optimizavimas

Visi optimizavimo elementai yra begalo svarbus žiniatinklinių aplikacijų kūrime, bet mes panagrinėkime techninę optimizavimo dalį. Nors šiuo metu kai kurios naujausios operacinės sistemos automatiškai reguliuojasi, kad bet kuriuo metu galėtų priimti daugiau duomenų, ir nustato jūsų interneto ryšio greitį ir galimą pralaidumą. Taip atsiranda galimybė greičiau atsisiųsti bylas ir transliuoti informaciją naudodami didelio greičio interneto ryšį. Tai reiškia, kad galite daugiau laiko skirti darbui su turiniu ir trumpiau laukti, kol jis bus atsiųstas. Tačiau yra būdų, kaip mes patys galėtume optimizuoti žiniatinklinių aplikacijų greitį. Techninė optimizavimo dalis ir jos kriterijai susideda iš:

- Prisijungimo prie žiniatinklio sparta
- Persiunčiamų duomenų kiekio sparta
- Transakcijų įvykdymo sparta
- Žiniatinklinių aplikacijų su duomenų baze duomenų apsikeitimo sparta
- Užklausos atsakymo sparta

Visiems šiems kriterijams optimizuoti egzistuoja begalo daug būdų. Tačiau vieni iš efektyviausių ir labiausiai paplitę yra :

- Podėliavimo(angl. Caching) panaudojimas.
- „Connection pooling“ panaudojimas.
- Serijinių užklausų(angl. batch queries) panaudojimas.

Kad pateikti bendrą supratimą apie analizuojamus optimizavimo būdus, panagrinėkime PHP programavimo kalbos galimybes kartu su MySql duomenų baze ir Apache 2 palaikymo serveriu.

7.1 Podėliavimas

Dinaminių žiniatinklinių puslapių pakrovimas užimta ganėtinai daugiau laiko nei statinių. Tai priverčia vartotojus laukti – kas toli gražu nepriveda prie teigiamų rezultatų.

Podėliavimas(cahing) yra galinga priemonė sutrumpinti ar bent dalinai sumažinti puslapių pakrovimo..laiką.

Šiuo metu mes su nostalgija žiurime į tuos laikus, kai žiniatinklių kūrimas susidėdavo iš kelių statinių HTML puslapių, kuriuose būdavo keli teksto, bei piešinukų laukai. Žiniatinklio siuntimo procesas iš serverio į naršyklės langą buvo „neskausmingas“ ir paprastas. Serveris turėjo paimti keletą bylų iš sistemos ir persiusti jas į naršykle, kuri savo ruožtu žaibiškai gaudavo sutvarkytą puslapį. Kai tik puslapis buvo pakraunamas, naršyklė išsyk iššaukdavo šabloniška podėliavimą tam, kad išsaugoti pakrautą puslapį kompiuterio atmintyje iki sekančios naršyklės užklausos tam pačiam puslapiui. Po to, kai kitą syk būdavo gaunama naršyklės užklausą tam pačiam puslapiui, naršyklė naudodama žiniatinklio serverį įsitikindavo ar puslapis nebuvo atnaujintas. Tokiu atveju, buvo atvaizduojamas podėliuotas puslapis iš kompiuterio atminties.

Tačiau su laiku žiniatinkliniai puslapiai tapo vis labiau sudėtingesni, atsirado naujų technologijų nukreiptų dinaminio turinio panaudojimui. Dauguma puslapių migravo iš statinių į dinaminius. To sekoje senasis podėliavimo procesas tapo nebeveiksmingas.

Šiuo metu dauguma internetinių puslapio užklausų susideda iš kelių vidiniu procesų, kurie lemia duomenų keliavimo kelią nuo žiniatinklio aplikacijos iki vartotojo. Dažniausia šie tarpiniai veiksmai yra prisijungimas prie duomenų bazės, jos duomenų užklausimas ir t.t. Tai lemią rezultato vėlavimą tarp vartotojo užklausos ir galutinio atvaizdavimo naršyklės lange.

Yra begalo daug šios problemos sprendimo būdų. Vienas iš jų, tai serverio pusės podėliavimas pasitelkiant į pagalbą PHP mechanizmą. Tam , kad išsiaiškinti, kaip galima būtų sumažinti užlaikymą iš serverio pusės, reikėtu sugeneruoti PHP puslapį. Puslapis turėtų gauti duomenis iš duomenų bazės, apdoroti juos, persiusti atgal ir tuo pačiu generuoti visus likusius veiksmus reikalingus puslapiui „gyvuoti“. Toliau reikėtų išsaugoti išvedimo duomenys atskiroje byloje(cache file) prieš tai, kol jie bus galutinai persiusti į naršyklės langą. Kitą kartą naršyklei užklausus puslapį bus galimybė patikrinti ar išaugota byla atitinka naršyklės užklausą. Jei užklausa atitinką išsaugotą bylą(cache file), tuomet duomenys yra nuskaitomi iš bylos ir iškarto siunčiami tiesiai į naršyklę. Tai žymiai sutrumpina užklausos atvaizdavimo laiką. Vietoj to, kad gauti serverio išvedimo duomenys, mes pasinaudosime keliomis buferio valdymo funkcijomis, kurias mums siūlo PHP programavimo kalba. Buferio valdymo funkcijos siūlo didelį mechanizmą serverio duomenų išvedimo valdymui[22]. Pvz:

```
<?php
```

```
   // visas skriptas prasideda funkcija ob_start
ob_start();
   // tekstas, kuris bus išsaugotas buferyje
echo ‘This text is in the buffer!<br />’;
   // išsaugojamas buferio tekstas į kintamąjį bufferContent
$bufferContent = ob_get_contents();
   // sustabdomas ir išvalomas buferio skaitytuvas
ob_end_clean();
   //   išvesties tekstas
echo ‘This text is not in the buffer!<br />’;
   //   buferio išsaugotas tekstas
echo $bufferContent;
   //   echo the buffer content
?>
```

Skripto išvestis:

```
This text is not in the buffer!
This text is in the buffer!
```

Šis pavyzdys puikiai atvaizduoja, kaip yra išsaugojami duomenys iš buferio skaitytuvo. Taigi šis principas lengvai gali būti pritaikytas bet kokio dydžio išvesties duomenims. Kiekviena žiniatinklio užklausa, kiekviena žiniatinklio klaida ir bet kokie išvesties duomenys galėtų būti podėliuojami tokiu būdu. Tokiu principu, galima būtu atsisakyti daugelių tarpinių veiksmų tarp užklausos inicializavimo ir jos atvaizdavimo naršyklės lange.

Kitame pavyzdyje pasinagrinėkime, kaip išvesties buferis gali būti susiejamas su serverio pusės optimizavimu. Susikurkime paprasta pavyzduką, kaip buferio turinys yra išsaugojamas atskiroje byloje[22]:

```
<?php

//   podėliavimo bylos patikrinimas
if ( file_exists( ‘cachefile.txt’ ) ) {
   //   jai egzistuoja atitinkanti turinį podėliavimo byla tuomet
//  nuskaitome iš bylos
  readfile ( ‘cachefile.txt’ );
  exit();
}

//   jei atitinkama podėliavimo byla neegzistuoja, sukuriame naują
ob_start();

//   atvaizduojame HTML tekstą (jis bus saugojamas podėliavimo byloje )
?>

<html>
<head>
```

```
<title>Caching server output</title>
</head>
<body>
<h2>This page is a cached Page</h2>
</body>
</html>

<?php
$bufferContent = ob_get_contents();
   //   get buffer content
ob_end_flush();
   //   išvalome ir atvaizduojame buferio turinį naršyklės lange
$fp = fopen ( 'cachefile.txt' , 'w' ) or die ( 'Error opening cache file' );
   //  įrašome buferio turinį į bylą
fwrite ( $fp , $bufferContent );
fclose( $fp );
?>
```

Šio skripto išvestis yra naršyklės lange atsiradęs tekstas:

```
This page is a cached Page
```

Podėliavimo bylos turinys 'cachefile.txt' yra toks pat kaip HTML tekstas:

```
<html>
<head>
<title>Caching server output</title>
</head>
<body>
<h2>This page is a cached Page</h2>
</body>
</html>
```

Visų pirma skriptas patikrina ar yra supodėliuota jau esama puslapio versija. Tai yra įvykdoma sutikrinant ar egzistuoja padėliavimo byla, atitinkanti užklausos turinį. Jei tokia byla egzistuoja, ji yra nuskaitoma ir informacija siunčiama tiesiai į naršyklės langą. Kitu atveju, aktyvuojamas buferio kaupiklis, kuris pradeda kaupti visą generuojamą informaciją. Kai šis veiksmas užbaigiamas, visa sukaupta informacija buferio kaupiklyje yra išsaugojama kintamajame ($bufferContent). Toliau buferio kaupiklis yra išvalomas ir HTML tekstas yra atvaizduojamas vartotojuj. Galiausiai visa informaciją išsaugota kintamajame ($bufferContent) yra perrašoma į atskirą podėliavimo bylą. Akivaizdu, jog šis skriptas yra labai paprastas, tačiau turintys savyje begalo dideles potencines galimybes, kurias pateikia PHP programavimo kalba. Kitas loginis žingsnis būtų podėliuoti atskiras žiniatinklio teksto dalis. Pvz. <header>, <body>, <footer> ir t.t. Tos dalys, kurios dažniausiai pasikartoja ir rečiausiai yra atnaujinamos žiniatinklio skripte.

Tam, kad nustatyti ir integruoti į serverio lygmenį dažniausiai pasikartojančiu žiniatinklio dalių padėliavimą, mes turėtume nusistatyti kriterijus pagal kuriuos tikrinsime ar užklausos atitinką būtent ieškomus podėliuojamųjų bylų turinius. Svarbiausios padėliavimo politikos kriterijai yra šie[22] :

- Laiko podėliavimas (laiko atžymos galiojimo pabaiga).
- Turinio podėliavimas (turinio informacijos pasikeitimas).
- Rankinis podėliavimas (rankinių būdu aplikacijos informavimas apie galiojimo laiko pabaigą).

Visų pirma mes apsibrėžiame, kad puslapio antraštė(<header>) turi galiojimo laiką viena diena - 86400 sekundžių(tik kaip pavyzdys), <body> dalies galiojimo laikas yra 10sekundžių ir <footer> 86400 sekundžių. Pasižymėję kiekvienos dalies atskirai galiojimo laikus, pasirašome dvi intuityvias funkcijas. Pirmoji skirta padėliavimo bylų kūrimui, kita jų turinio informacijos tikrinimui[22]:

```
function createCache ( $content ,  $cacheFile ) {
  $fp = fopen( $cachefile , ‘w’ );
  fwrite( $fp , $content );
  fclose( $fp);
}

function getCache ( $cacheFile ,  $expireTime ) {
  if ( file_exists ( $cacheFile ) && filemtime ( $cacheFile ) >( time() -
$expireTime ) ) {
    return file_get_contents( $cacheFile );
  }
  return false;
}
```

CreateCache funkcija pasiima du parametrus: buferio išvesties turinį ir podėliuojamosios bylos pavadinimą, kur išvesties turinys bus išsaugotas. Kai tik buferio turinys yra gaunamas, jis iškarto yra išsaugojamas kaip atskira byla. Funkcija getCache taip pat gauna du argumentus : podėliuojamosios bylos pavadinimas ir galiojimo laiko vienetus(sekundėmis). Funkcija patikrina ar egzistuoja podėliavimo bylą ir ar jos galiojimo laikas dar nepasibaigė. Jei galiojimo laikas dar nėra pasibaigęs, tuomet yra gražinamas podėliuojamosios bylos turinys, kitu atvejų gražinama – false[22]:

```
?>

<?php
```

```
function createCache ( $content ,  $cacheFile ) {
  $fp = fopen( $cachefile , ‘w’ );
  fwrite( $fp , $content );
  fclose( $fp); }
function getCache ( $cacheFile ,  $expireTime ) {
  if ( file_exists ( $cacheFile ) && filemtime ( $cacheFile ) >( time() -
$expireTime ) ) {
    return file_get_contents( $cacheFile );
  }
  return false;
}
ob_start();
if ( !$header = getCache( ‘headerCache.txt’ , 86400 ) {
//  header dalis
?>
<html>
<head>
<title>Cached Page</title>
</head>
<body>
The header section is updated on a daily basis.
<?php
$header = ob_get_contents();
ob_clean();
createCache( $header , ‘headerCache.txt’ );
}
// body dalies patikrinimas
if ( !$body = getCache( ‘bodyCache.txt’ , 10 ) {
//  body dalis
?>
<h1>This section is updated every 10 seconds.</h1>

<?php
$body = ob_get_contents();
ob_clean();
createCache( $body , ‘bodyCache.txt’ );
}
// footer dalies patikrinimas
if ( !$footer = getCache( ‘footerCache.txt’ , 86400 ) {
//  footer dalis
?>
The footer section is updated on a daily basis.
</body>
</html>
<?php
$footer = ob_get_contents();
ob_clean();
createCache( $footer , ‘footerCache.txt’ );
}
ob_end_clean();
echo $header .  $body .  $footer;
?>
```

Skripto išvestis:

```
The header section is updated on a daily basis.
```

```
This section is updated every 10 seconds.

The footer section is updated on a daily basis.
```

Kiekviena puslapio dalis bus atnaujinta priklausomai nuo podėliavimo politikos. Bendra šio podėliavimo idėja ir slypi tame, kad bet koks šio skripto pagerinimas ar patobulinimas gali būti pasiektas tik pasitelkiant teisinga podėliavimo politiką atskiroms žiniatinklio dalims. Kita žiniatinklių greičio optimizavimo priemonė yra „connection pooling“.

7.2 “Connection pooling”

„Connection pool**“** - yra duomenų bazių užklausų podėliavimas(caching) duomenų bazės lygmenyje. Jis yra naudojamas prisijungimų pakartotiniam panaudojimui, kai duomenų bazė gauna busimas pasikartojančias duomenų užklausas[20].

„Connection pool“ yra naudojamas tam, kad padidinti ir pagerinti vykdomąsias užklausų komandas duomenų bazėje. Kaskart sukurti duomenų bazei prisijungimą ir palaikyti nepertraukiamą ryši kiekvienam vartotojui yra ganėtinai brangu ir gali privesti prie resursų išvaistymo. Panaudojant „Connection pooling“ yra sukuriamas prisijungimas ir jis yra patalpinamas į atminties vietą(pool).Vėliau, ateityje, tas pats prisijungimas yra kaskart iškviečiamas būtent iš tos atminties vietos, kur jis buvo patalpintas, o ne kiekvieną kartą inicializuojant jį iš naujo. Jei visi prisijungimai yra užimti, yra inicializuojamas naujas prisijungimas ir yra automatiškai patalpinamas vėl į atminties vietą(pool). „Connection pooling“ yra taip pat priklausomas nuo laiko, kuo ilgiau yra nesinaudojama prisijungimais tuo jų kiekis automatiškai mažėja atmintyje[20].

„Connection pooling“ yra dažniausiai naudojamas internetinių aplikacijų paremtose formose(web-based) ir yra valdomas jų serveryje. Dinaminiuose interneto puslapiuose inicializuoti bei uždaryti prisijungimus su duomenų baze yra savaime suprantamas reiškinys, bet pagal nutylėjimą, kai atsiranda būtinybė naujam prisijungimui inicializuoti, jis yra neinicializuojamas, o būtent yra iškviečiamas iš atminties vietos(pool), kuri yra aprūpinama iš aplikacijos serverio. Ir atvirkščiai, kai prisijungimas prie duomenų bazės yra sunaikinamas, tai pagal nutylėjimą jis yra tiesiog gražinamas atgal į atminties vietą(pool)[20].

Visi šie „connection pool“ procesai yra valdomi ir tokie parametrai kaip – minimalus prisijungimų skaičius, maksimalus prisijungimų skaičius ir tuščiaeigių prisijungimų skaičius, yra

lengvai reguliuojami tam, kad „coonection pool“ veiktu nepriekaištingai priklausomai nuo aplinkos, kuriai ji yra pritaikyta[21].

Panagrinėkime praktinį „connection pool“ panaudojimą Apache2 serverio aplinkoje palaikant MySql duomenų bazę. Apache2 serveris savyje jau turi kelias funkcijas, kurios programiškai reguliuoja dinaminius „connection pool“ išteklius. Belieka tik parūpinti konstruktorių ir destruktorių mūsų resursams. Taip pat pasirašyti funkcijas, kurios galėtų manipuliuoti resursais iš atminties(pool). Visą kitą optimizavimo darbą atlieka apr_reslist modulis. Tam, kad viską aiškiau atvaizduoti, pasirašykime keletą esminių funkcijų MySql įdiegimui(mod_mysql_pool). Visų pirmą konstruktorius ir destruktorius. Mes niekada jų tiesiogiai nekviečiame, apr_reslist modulis iškviečia juos kai yra būtinybė[21].

```
/* apr_reslist_constructor naudojamas prisijungimui prie MySQL
   Inicializuoja prisijungimą prie MySQL ir išsaugo jį *db
*/
static apr_status_t mysqlpool_construct(void** db, void* params, apr_pool_t*
pool) {
  svr_cfg* svr = (svr_cfg*) params ;
  MYSQL* sql = NULL ;
  sql = mysql_init(sql) ;
  if ( sql == NULL ) {
    ap_log_perror(APLOG_MARK, APLOG_CRIT, 0, pool, "mysql_init failed") ;
    return APR_EGENERAL ;
  }
  *db = mysql_real_connect(sql, svr->host, svr->user, svr->pass,
        svr->db, svr->port, svr->sock, 0) ;

  if ( ! *db ) {
    ap_log_perror(APLOG_MARK, APLOG_CRIT, 0, pool, "MySQL Error: %s",
        mysql_error(sql) ) ;
    return APR_EGENERAL ;
  }
  return APR_SUCCESS ;
}
static apr_status_t mysqlpool_destruct(void* sql, void* params, apr_pool_t*
pool)
{
  mysql_close((MYSQL*)sql) ;
  return APR_SUCCESS ;
}
```

Kitam etapui yra reikalinga funkcija, kuri registruotų dinaminius išteklius. Tai vadinasi post_config kilpa. Joje yra iškviečiamas metodas apr_reslist_create inicializuojantis atminties vietą(pool) prisijungimams prie MySql duomenų bazės[21]:

```
static int setup_db_pool(apr_pool_t* p, apr_pool_t* plog,
        apr_pool_t* ptemp, server_rec* s) {
  svr_cfg* svr = (svr_cfg*)
```

```
        ap_get_module_config(s->module_config, &mysql_pool_module) ;

  if ( apr_reslist_create(&svr->dbpool, svr->nmin, svr->nkeep,
        svr->nmax, svr->exptime, mysqlpool_construct, mysqlpool_destruct,
        svr, p) != APR_SUCCESS ) {
    ap_log_error(APLOG_MARK, APLOG_CRIT, 0, s, "MySQLPool: failed to initialise"
) ;
    return 500 ;
  }
  apr_pool_cleanup_register(p, svr->dbpool,
        (void*)apr_reslist_destroy,
        apr_pool_cleanup_null) ;
  return OK ;
}
```

Galiausiai reikia sukurti galimybę, kad kiti moduliai, pasinaudodami šiomis funkcijoms, taip pat galėtų priimti bei gražinti prisijungimus į atminties vietą(pool). Kaip pavyzdį pateiksiu metodą mysql_acquire. Šis metodas iškviečiamas būtiniausiu atveju, kai prisijungimas naudojamas tiek kiek trunka užklausa[21].

```
MYSQL* mysql_acquire(request_rec* r, unsigned int flags) {
  mysql_request* req = (mysql_request*)
        ap_get_module_config(r->request_config, &mysql_module) ;
  if ( ! req ) {       /* use pool if and only if we haven't already got one */
    svr_cfg* svr = (svr_cfg*)
        ap_get_module_config(r->server->module_config, &mysql_module) ;
    req = (mysql_request*) apr_palloc(r->pool, sizeof(mysql_request) ) ;
    req->flags = flags ;
    req->dbpool = svr->dbpool ;
    if ( apr_reslist_acquire(svr->dbpool, (void**)&req->sql)
        != APR_SUCCESS ) {
      ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
        "Failed to acquire MySQL connection from pool") ;
      return NULL ;
    }
    if ( mysql_ping(req->sql) != 0 ) {
      ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, "MySQL: %s",
        mysql_error(req->sql) ) ;
      apr_reslist_invalidate(svr->dbpool, req->sql) ;
      return NULL ;
    }
    ap_set_module_config(r->request_config, &mysql_module, req) ;
    apr_pool_cleanup_register(r->pool, req,
        mysql_release, apr_pool_cleanup_null) ;
  } else {
    req->flags |= flags ;  /* ensure all required cleanups happen */
  }
  return req->sql ;
}
```

Toliau, viską suintegruodami į bendrą visumą ir pasinaudodami Apache 2 serverio teikiamais privalumais programiškai reguliuoti „connection pooling“, mes galime optimizuoti žiniatinklio ir duomenų bazės duomenų apsikeitimo spartą.

7.3 Serijinės užklausos(batch queries)

Serijinės užklausos(batch queries)- yra standartinių užklausų paketas. Jį dažniausiai sudaro daugybę SQL užklausų su skirtingomis reikšmėmis. Tai yra ruošinys iš anksto numatytiems sistemos užklausimams. Jis yra naudojamas daugialypėm užklausom įvykdyti. Jei, tarkim, vartotojas kelis kartus ieško skirtingų duomenų toje pačioje lentelėje, vidutinis užklausos greitis padidėja, nes numatytos užklausos jau būna paruoštos užklausų pakete. Jei reikia vartoti daug panašių veiksmų su skirtingais parametrais, serijinės užklausos gali pasiteisinti. Bet yra ir kita šio medalio puse. Priklausomai nuo duomenų bazės struktūros ir žiniatinklio aplikacijos pobūdžio, serijinės užklausos gali neatnešti jokios naudos, kadangi paruošiamas paketas veikia tik vienos sesijos metu ir pasiteisintų tik tuo atveju jei užklausos būtų atliekamos tame pačiame puslapyje kelis kartus. Tačiau dažniausiai būna atvirkščiai – kiekvieną užklausa iškviečiame skirtingame puslapyje. Todėl serijinės užklausos dažniausiai vartojamos ne žiniatinklinėse aplikacijose, o didelės valdymo sistemose. Pasinagrinėkime patį serijinių užklausų veikimą PHP programavimo kalboje prisijungiant prie MySql duomenų bazės:

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query  = " Select * from TestQBatch where col1 = 1";
$query  .= " Select * from TestQBatch where col1 = 2";
$query  .= " Select * from TestQBatch where col1 = 3";
$query  .= " Select * from TestQBatch where col1 = 4";
$query  .= " Select * from TestQBatch where col1 = 5";
$query  .= " Select * from TestQBatch where col1 = 10";

if ($mysqli->multi_query($query)) {
    do {
        if ($result = $mysqli->store_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
```

```
            $result->close();
        }
        if ($mysqli->more_results()) {
            .................
        }
    } while ($mysqli->next_result());
}

$mysqli->close();
?>
```

Kaip galima matyti iš šio pavyzdžio, PHP programavimo kalba kartu su MySql duomenų baze neturi išankstinio suderinto ir paruošto metodo, ar funkcijos. Kadangi pati idėja nėra sudėtinga, tai skriptas lengvai sukuriamas ir paprastai valdomas rankiniu būdu. Atskirai sukurtus serijinių užklausų metodus turi JDBC bei ADO.NET. Šiek tiek patogesnis jų vartojimas, tačiau principas išlieka toks pat.

VIII. Praktinis žiniatinklinės aplikacijos optimizavimas

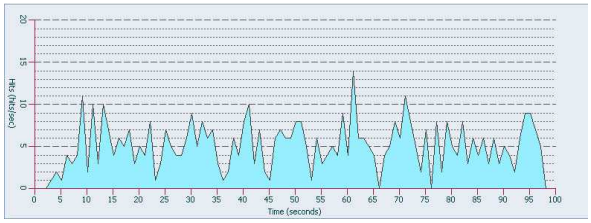
Žiniatinklio optimizavimui naudosime prieš tai išnagrinėtus tris greičio optimizavimo būdus: podėliavimas, „connection pooling“ ir serijinės užklausos. Pasinaudojant jais, pabandysime pasiekti optimaliausius rezultatus ir padaryti išvadas. Darydami išvadas, remsimės konkrečiais bandymais su optimizuojamojo žiniatinklio testų rezultatais.

- Optimizavimo objektas – www.supermodels.lt (žiūrėti priedus – 1,2) Šis žiniatinklio puslapis yra parašytas naudojant PHP programavimo kabą. Duomenys yra saugojami MySql 5.0 duomenų bazėje. Visas žiniatinklis yra palaikomas su Apache HTTP Server 2.0.61 serveriu.
- Optimizuojamo objekto puslapis http://www.supermodels.lt/lt/people-portfolios.php (registruotam vartotojuj, priedas - 2)
    - 36 KB HTML byla
    - 400 KB ~100 paveiksliukų
    - 11 KB CSS
    - 138 KB JS
- Puslapis yra skirtas informacijos peržiurai pagal atskirus kriterijus, t.y puslapyje atvaizduojama informacija – iš duomenų bazės yra pakraunami žmonių anketų duomenys su nuorodomis į jų nuotraukas. Pagal skirtingus kriterijus duomenys yra filtruojami ir generuojamos naujos užklausos.
- Testavimo priemonės JBlitz Pro 5.1.

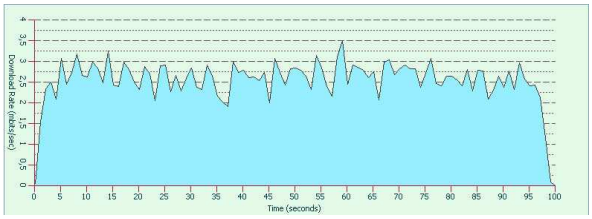
- Vidutinis testavimo laikas -140 sek.
- Virtualus vartotojų skaičius testavimo metu – 60.
- Testavimo sistemos parametrai – Windows XP/Linux AMD Athlon(tm) 64 X2 Dual, Core Processor 5600+, 2.81GHz, 2,00 GB of RAM , modemas 100Mbps. Tinklo greitis – 1536Kbps. Sistema yra testuojama neatsižvelgiant į tinklo trikdžius.

Prieš taikant optimizavimo metodus analizuojamam puslapiui, buvo atlikti pirminiai jo testavimo rezultatai:

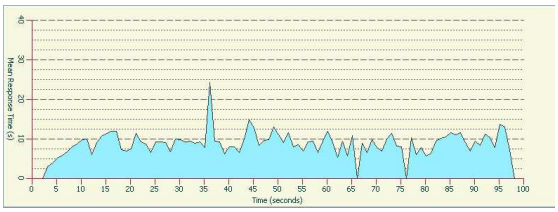
Prisijungimo sparta (kart/sek)



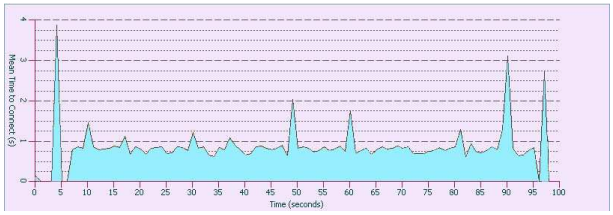
Duomenų siuntimo sparta (mbit/sek)



Vidutinis užklausos atsakymas (sek)



Vidutinis prisijungimo laikas (sek)



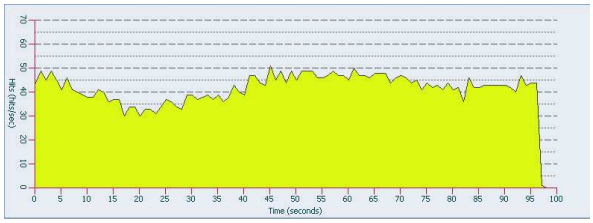
5 pav. Pirminių testavimo rezultatų grafikai

Prisijungimų skaičius	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
498	5,09	327.75	2.679,75	9357	791

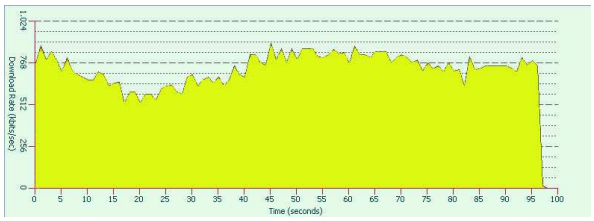
6 lentelė. Pirminiai testavimo rezultatai

Pats svariausias rodiklis mūsų puslapio lankytojui būtų prisijungimų dažnis bei prisijungimo laikas. Jais remiantis, galima daryti išvadas, koks yra puslapio apytikslis greitis. T.y kiek kartų per sekundę įmanoma prisijungti prie mūsų analizuojamo puslapio. Šis rodiklis parodo, kiek puslapis yra optimalus ir kiek jis gali dalinai apdoroti užklausų per tam tikrą laiką. Šiame bandyme mes matome prisijungimų skaičių 5,09 karų per sekundę. T.y rezultatas gautas be jokio papildomo optimizavimo metodo pritaikymo. Dabar paanalizuokime rezultatus pritaikant padėliavimo optimizavimo metodą :

Prisijungimo sparta (kart/sek)

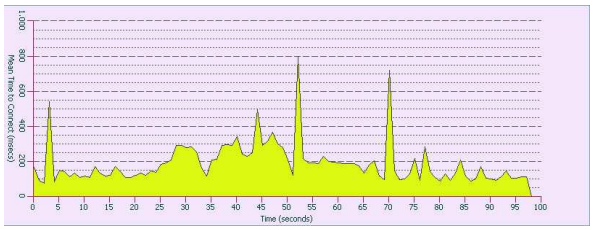


Duomenų siuntimo sparta (kbit/sek)



Vidutinis užklausos atsakymas (msek)

Vidutinis prisijungimo laikas (msek)



7 pav. Podėliavimo metodo testavimo rezultatų grafikai

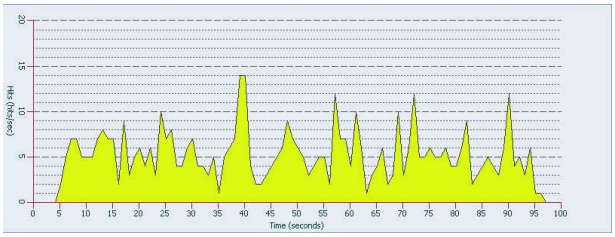
Prisijungimų skaičius	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
4093	**42,6**	8941	742,02	216	195

8 lentelė. Podėliavimo metodo testavimo rezultatai

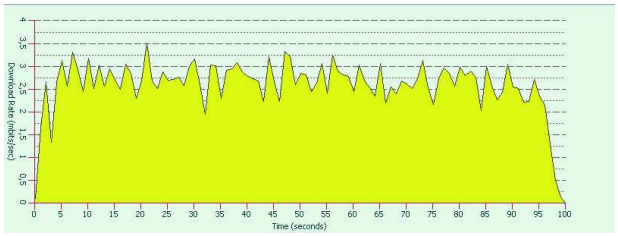
Panaudojant padėliavimo optimizavimo metodą matosi ryškus puslapio spartos padidėjimas. Vien tik kalbant apie prisijungimų dažnius per sekundę, galima pamatyti žymų skirtumą. Nuo

pradinio testavimo prisijungimų skaičius per sekundę padidėjo daugiau nei 8 kartus. Vidutinis prisijungimo laikas sumažėjo iki 3 kartų. Rezultatai jau džiugina. Pasinagrinėkime „connection pool“ metodą:

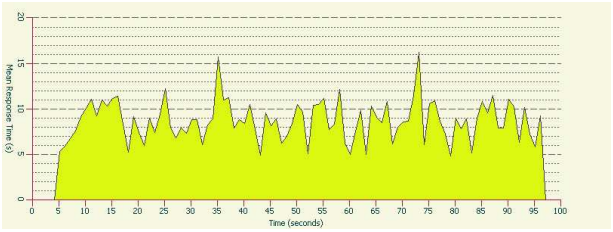
Prisijungimo sparta (kart/sek)



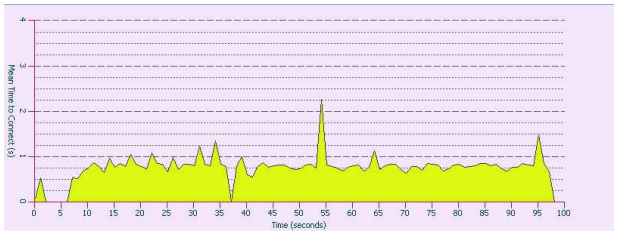
Duomenų siuntimo sparta (mbit/sek)



Vidutinis užklausos atsakymas (sek)



Vidutinis prisijungimo laikas (sek)



9 pav. „Connection pool“ metodo testavimo rezultatų grafikai

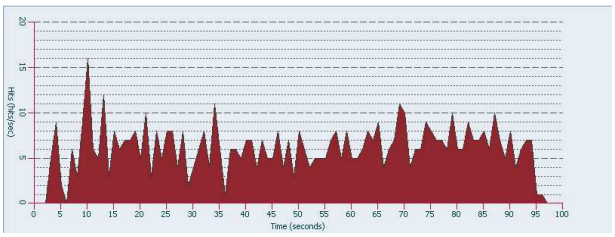
Prisijungimų skaičius	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
494	5,19	33290	2.799,24	8769	761

10 lentelė. „Connection pool“ metodo testavimo rezultatai

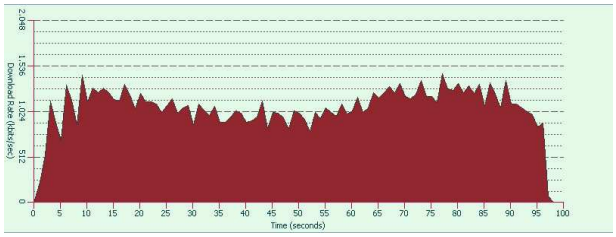
Įskaitant tinklo trikdžius ir užlaikymus, galima teigti jog „connection pool“ metodas nepasiteisino. Prisijungimo dažnis prie puslapio pasikeitė nežymiai. Prisijungimo laikas taip pat liko beveik nepakitęs. Nagrinėjant „connection pool“ veikimo principą ir jo panaudojimą, galima būtų daryti išvadas, kad tokio pobūdžio puslapiuose jis yra nepraktiškas. Tikriausiai dėl tos priežasties

Apache2 kūrėjai ir atsisakė paketinio „connection pool“ metodo. Apache2 serveris turi ApacheChild paprocesorį, kuris pagal nutylėjimą naudoja vieną prisijungimą. Šis procesų skaičius yra reguliuojamas. Be to, MySql duomenų bazė savyje taip pat globaliai podėliuoja duomenys, dėl to priežasties „connectio pooling“ tampa menkavertis. Pasianalizuojant nuodugniai „connection pooling“ metodą, galima drąsiai teigti jog šis metodas pasiteisins tik tuo atveju, kai puslapis bus užkraunamas ne mažiau kaip pora šimtų užklausų per sekundę. Užklausos turėtų susidaryti iš nesudėtingų trumpalaikių prisijungimų. Kitas optimizavimo metodas serijinės užklausos(batch queries):

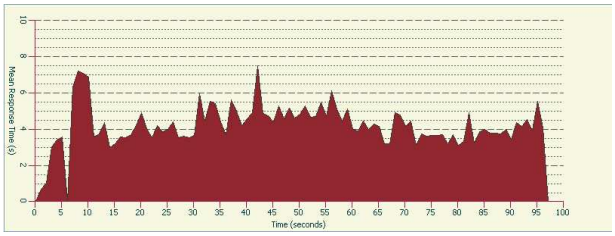
Prisijungimo sparta (kart/sek)



Duomenų siuntimo sparta (kbit/sek)



Vidutinis užklausos atsakymas (sek)



Vidutinis prisijungimo laikas (sek)

11 pav. Serijinių užklausų metodo testavimo rezultatų grafikai

Prisijungimų skaičius	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
501	5,9	31810	2.801,44	8261	720

12 lentelė. Serijinių užklausų metodo testavimo rezultatai

Serijinės užklausos taip pat nepasiteisino. Pagreitėjimas yra nežymus, siekiantis apytiksliai dešimties nuošimčių ribą. Kadangi serijinės užklausos veikia optimaliausiai tik tuo atveju, kai užklausos atliekamos po kelis kartus su tais pačiais duomenimis per vieną sesiją. Mūsų testuojamu atveju vyko priešingas procesas. Per vieną sesiją vyko vienas užklausimas.

Toliau detalesniam palyginimui ir konkrečių atveju analizavimui buvo padarytą kiekvienos atskirai optimizuojamųjų kombinacijų testavimas:

Per 100sek	**Prisijungimų skaičius**	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
Pradinis	498	5,09	327.75	2.679,75	9357	791
Podėliavimas	4093	42,6	8941	742,02	216	195
„Connection pooling“	494	5,19	33290	2.799,24	8769	761
Serijinės užklausos	536	5,9	31810	3.101,44	8261	720
Podėliavimas + „Connection pooling“	4328	45,5	9457	795,30	181	136
Podėliavimas + Serijinės užklausos	4101	42,7	8992	742,70	213	193
„Connection pooling“ + Serijinės užklausos	531	5,9	30710	3.100,12	8277	722
Podėliavimas + „Connection pooling“ + Serijinės užklausos	4351	45,6	9471	799,10	179	135

13 lentelė. Bendri visų metodų testavimo rezultatai

Iš visų rezultatų galima matyti, jog tarp mūsų nagrinėjamųjų optimizavimo metodų žymiausią įtaką turi - podėliavimas. Nuo pradinio žiniatinklio varianto, padėliavimo optimizavimas puslapiui suteikia ganėtinai didelę reikšmę. Mūsų analizuojamajam žiniatinkliui šis metodas

pranoksta visus kitus dešimteriopai. Tikriausiai pagrindinė priežastis buvo ta, jog žiniatinklis, kuris buvo optimizuojamas, yra vidutinio apkrovimo lygio. Jo kodas nėra itin sudėtingas, kreipimosi užklausos į duomenų baze taip pat nėra kompleksinės, todėl serijinės užklausos ir „connection pooling“ nepadarė didelės įtakos optimizavime. Kaip ir buvo minėta ankščiau, „connection pooling“ yra skirtas tik tiems puslapiams, kuriems būdinga būtų atlaikyti milžiniškus užklausų kiekius per sekundę. O kas liečia serijines užklausas, tai duomenų apsikeitimas tarp nagrinėjamo žiniatinklio ir duomenų bazės nėra toks sudėtingas ir kompleksinis, kad serijinių užklausų optimizavimas būtų ryškus. Be to, taisyklingai podėliuojant duomenys, serijinės užklauso būtinumas mažėja.

VIII. Pridėtiniai žiniatinklių greitikliai

Pridėtinis žiniatinklio greitiklis – programinė įranga, padedanti optimizuoti žiniatinklio veikimą nekeičiant esminio programinio kodo(griaučių). T.y programinė įranga, kuri išoriškai padeda susisteminti, suoptimizuoti veiksnius, kurie įtakotų tinklalapio veikimo greitį. Pagal jų veikimo principus galima būtų suskirstyti greitiklius į : grafikos optimizatorius ir sukompiliuotų bitų kodų optimizatorius.

Bitų kodo optimizatorius yra plėtinys pagreitinantis programinės įrangos veikimo spartą. Pats veikimo principas yra pagrįstas tuo, jog optimizatoriai podėliuoja sukompiliuotą skripto kodą tam, kad išvengti pasikartojančio kodo kompiliavimo bei tikrinimo. Kadangi kompiliavimas vyksta serverio pusėje, tad bitu kodo optimizatorius gali veikti tik nekompiliuojamos kalbos pagrindu sukurtose žiniatinkliuose. Tokiuose interpretuojamose kalbose kaip : PHP, Ruby, Python ir t.t. Toliau visi detalesni veiksniai priklauso nuo tam tikro pridėtinio žiniatinklio greitiklio. Kai kurie jų turi tiesiog skirtingus specifinius veikimo etapus, kuriose podėliuoja atmintį : į serverį kreipimosi metu, žiniatinklio paleidimo metu, informacijos siuntimosi metu ir t.t. Visi procesai kuriuose yra kompiliuojamas programos kodas serverio pusėje. Nuo programavimo kalbos rūšies priklauso ir pridėtinių žiniatinklių greitiklių struktūra. Dauguma programavimo kalbų turi būtent joms būdingus sukompiliuotų kodų fragmentus bei unikalią sintaksę. Todėl, dažniausiai pridėtiniai žiniatinklių greitikliai yra rašomi tik tam tikroms programavimo kalboms: JServer Accelerator; PHP Accelerator, XCache, eAccelerator. Zend Optimizer, ionCube PHP Accelerator, Net Accelerator ir t.t. Remiantis kūrėjų statistiniais duomenimis, pridėtiniai žiniatinklio greitikliai paspartina vartotojo veiksmus nuo 2 iki 10 kartų priklausomai nuo pačio žiniatinklio struktūros, pradinio kodo(griaučių), paskirties bei vartojimo specifikos. [26][27][28]

Tam, kad objektyviau įvertinti šiuo optimizatorių naudą, buvo pabandyta pritaikyti vienas iš populiaresnių pridėtinių bitų kodo optimizatorių anksčiau minėtam optimizuojamajam puslapiui:

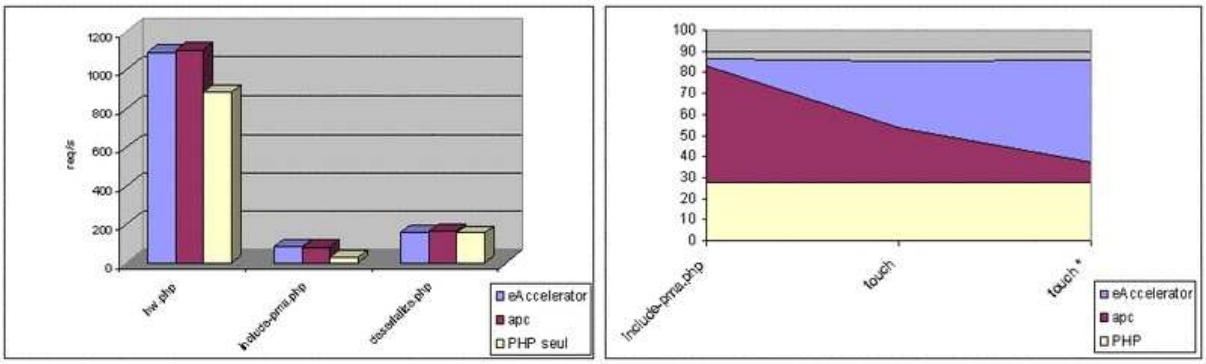
Per 100sek	**Prisijungimų skaičius**	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
Pradinis	498	5,09	327.75	2.679,75	9357	791
+ eAccelerator	**1059**	**12,5**	**689**	**6.179,5**	**4413**	**373**
Podėliavimas	4093	42,6	8941	742,02	216	195
+ eAccelerator	**6139**	**63,4**	**13411**	**1113**	**144**	**130**
Podėliavimas + „Connection pooling“	4328	45,5	9457	795,30	181	136
+ eAccelerator	**6501**	**68,2**	**14423**	**1212**	**132**	**124**
Podėliavimas + Serijinės užklausos	4101	42,7	8992	742,70	213	193
+ eAccelerator	**6141**	**63,5**	**13497**	**1190**	**143**	**131**
Podėliavimas + „Connection pooling“ + Serijinės užklausos	4351	45,6	9471	799,10	179	135
+ eAccelerator	**6605**	**69,7**	**14641**	**1287**	**130**	**120**

14 lentelė. Pridėtinio bitų optimizatoriaus testavimo rezultatai

Lyginant su kitų vartotojų internete [30] [31] [32] [33] pateikta bandymų statistika, mano atlikti rezultatai sąlyginai sutapo. Pateikta bendra statistika, jog atliekant bandymus žiniatinklio prisijungimo sparta padidėja nuo 2 iki 4 kartų, mano lentelėje sparta kinta tuo 1,5 iki 2 kartų . Galėtume palyginti su vienos iš internetinių technologijų kompanijos iPerSec pateikiamais duomenimis[30]

:

(HTTP užklausų įvykdymas per sekundę su eAccelerator)



15 lentelė. iPerSec patikiami testavimo rezultatai

Daugumos nešališkų kompanijų pateikiamų duomenų statistika daugmaž vienoda . Tad, remdamiesi visa surinkta medžiaga, galėtume daryti prielaidą, jog pridėtiniai bitų kodo optimizatoriai vidutiniam tinklalapiui tikrai pasiteisina.

Grafikos optimizatoriai yra plėtiniai kurie mažina paveikslėlių atminties apimtį, taip pagreitinantys jų parsisiuntimo laika. Šiuolaikinėje žiniatinklių erdvėje tikriausiai vargu ar rasime tinklalapį kuriame nebūtų turinio grafiniu pavidalu. Pagrindiniai dizaino aspektai, nuotraukos ir tinklalapių fono struktūra yra ne kas kita kaip pridėtiniai paveiksliukai. Tai sudaro vos ne didžiąją laiko dalį tinklalapio užkrovime . Todėl grafikos optimizatoriai kartais iš esmes išsprendžia visus tinklalapio greičio trūkumus. Yra taip pat sukurta daugybė grafikos optimizavimo programų, bet visų jų esm yra mažinti paveiksliukų apimtys. T.y :

1. Paveikslėlių suspaudimo algoritmo keitimas
2. Paveikslėlių rezoliucijos mažinimas;
3. Paveikslėlių spalvų skaičiaus mažinimas;
4. Paveiksliukų informacijos(angl. EXIF) šalinimas.

Varijuojant tarp šių trijų galimybių, įmanoma pasiekti tikrai puikius rezultatus. Vartotojui yra ne itin būtini pirminiai parametrai. Jam yra svarbiau bendras vaizdas kartu su jo gavimo sparta. Kadangi, kaip jau buvo paminėta, visų grafinių optimizatorių veikimo principas yra gana panašus, tad pasinagrinėkime vieno iš plačiau paplitusio grafinio optimizatoriaus GifBot pateikiamų rezultatų:

- Optimizuojamo objekto puslapis http://www.supermodels.lt/lt/people-portfolios.php
  - 36 KB HTML byla
  - 400 KB ~100 paveiksliukų
  - 11 KB CSS
  - 138 KB JS
- Optimizuojamas paveiksliukas

  http://www.supermodels.lt/pics/105x137/2621-2c06873a773c4311.jpg
- Testavimo priemonė GifBot 1.1.

Originalas: 4027 baitų H: 137 W: 103 	Dydis: 3774 baitų Opt: **6 %** 	Dydis: 2255 baitų Opt: **44 %** 	Dydis: 1351 baitų Opt: **66 %** 
Dydis: 3994 baitų Opt: **2%** 	Dydis: 3040 baitų Opt: **24 %** 	Dydis: 1915 baitų Opt: **52 %** 	Dydis: 953 baitų Opt: **76 %** 
	Dydis: 2395 baitų Opt: **40 %** 	Dydis: 1710 baitų Opt: **57 %** 	

16 lentelė. Optimizuojamo paveiksliuko testavimo rezultatai

Rezultatai rodo, jog grafinis optimizatorius yra gana efektingas. Optimizuojant nuotrauka iki 50 nuošimčių skirtumas plika akimi yra sunkiai pastebimas. Tad optimizuojant visus esančius paveiksliukus šiame puslapyje įmanoma sumažinti puslapio dydį iki 30 nuošimčių. O tai reiškia jog mūsų optimizuojamas puslapis dabar parsisius pusantro karto greičiau. Aišku žiniatinklio optimizavimo rodiklis sąlyginai kis nuo paveiksliukų užimamos dalies visame tinklalapio turinyje. Pagal [29] puslapyje pateiktus duomenys, grafinis optimizatorius vidutiniškai penktadalių sumažina tinklalapių turinio apimtį. Yra aprašomi atvejai, kai tinklalapių apimtis gali kisti dešimteriopai, bet tuomet visas tinklalapio turinys turi būti grindžiamas vien tik grafininėmis priemonėmis. Palyginkime statistinius duomenys su mano optimizuojamojo žiniatinklio tinklalapiu, šiuo atveju pridekime dar ir GifBot 52 nuošimčių optimizavimo pritaikymą :

Per 100sek	**Prisijungimų skaičius**	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
Pradinis	498	5,09	327.75	2.679,75	9357	791
+ GifBot	**846,8**	**8,65**	**557,15**	**4.554,00**	**5504**	**466**
+ eAccelerator **+ GifBot**	**1810**	**22,1**	**1170**	**10.450,65**	**2674**	**257**
Podėliavimas	4093	42,6	8941	742,02	216	195
+GifBot	**4905**	**47,1**	**10121**	**854,20**	**167**	**128**
+ eAccelerator **+ GifBot**	**6597**	**69,4**	**14402**	**1270**	**130**	**119**

17 lentelė. GifBot testavimo rezultatai

Kaip matome pirmu atveju GifBot pasiteisina. Pasiekiama yra daugiau kaip pusantro karto didesnė sparta. Vadinasi tiesioginio turinio dydžio mažinimas be abejo padidina prisijungimo skaičių kiekvienam serveriui. Bet antrame atvejyje skirtumas matosi visai nežymus. Tai yra todėl, jog pritaikant podėliavimą, podėliuojamųjų paveiksliukų apimtis labai didelės įtakos nebeturi pasikartojančiam užklausimui. To priežastis žymiai nepadidėjęs prisijungimų skaičius neunikaliems vartotojams. Šiuo atveju mes gauname naudingumą padidėjusį vidutiniškai apie 12 nuošimčių. Tai

nėra itin daug lyginant su galimą paklaidą (paklaida aprašyta bandymų pradžioje). Toliau atliekant statistinius stebėjimus podėliuojamajam puslapiui ir mažinant grafinio turinio kiekį tinklalapyje gautas rezultatas, jog dešimties nuošimčių naudingumo riba jaučiasi tik tuomet, kai grafinio turinio pasiskirstymas puslapyje viršija 2/3 viso turinio. Tad grafinio optimizatoriaus naudingumas su pritaikytu padėliavimu yra vartotinas tik su 2/3 ir daugiau grafinio tūrinio visame puslapyje.

Išvados

Apibendrinant šio darbo rezultatus galima būtų teigti, kadangi dauguma šiuolaikinių žiniatinklių technologijų yra pritaikytos vidutiniam vartotojų poreikiui, visos optimizavimo priemonės, kurios yra nustatytos pagal nutylėjimą, yra būtent akcentuotos vidutiniam optimaliausiam vartojimui. T.y jog kartų su žiniatinklių serveriais, duomenų palaikymo bazėmis, yra kuriamos ir optimizavimo priemonės jose. Dėl to, pagalbiniai ar pašaliniai optimizavimo metodai yra būtini tik išskirtiniais ar specialiai numatytais atvejais. Kas liečia PHP+MySql+Apache vienintelis pasiteisinantis metodas yra podėliavimas kartu su pridėtiniais bitų kodo optimizatoriais. Dėl grafinio optimizavimo yra šiek tiek sudėtingiau. Naudojant jį kaip vienintelį optimizatorių jis būtinai pasiteisins. O jei vartojimas yra kartu su padėliavimu, tuomet reikia atsižvelgti į procentinį grafinio turinio kiekį optimizuojamajame žiniatinklyje.

Per 100sek	**Prisijungimų skaičius**	**Prisijungimai / sekundę**	**Duomenų kiekis (Kbits)**	**Duomenų siuntimo dažnis(Kbits/sek)**	**Vidutinis užklausos atsakymo laikas(ms)**	**Vidutinis prisijungimo laikas(ms)**
Pradinis	498	5,09	327.75	2.679,75	9357	791
Podeliavimas eAccelerator GifBot	**6597**	**69,4**	**14402**	**1270**	**130**	**119**

18 lentelė. Galutiniai testavimo rezultatai

Įvertinant visas paklaidas ir trikdžius, viso mokslinio darbo eigoje taikyti optimaliausi optimizavimo būdai galiausiai pasiteisino dešimteriopai. Iš galutinių rezultatų lentelės mes matome, jog analizuojant dauguma optimizavimo būdu, man pavyko suoptimizuoti puslapį ir paspartinti jo veikimą dešimčia kartų nuo pradinės jo būsenos. Manau nagrinėjant skirtingas programavimo kalbas, skirtingas duomenų bazes kartu su skirtingais palaikymo serveriais, rezultatus, aišku, gausime skirtingus. Bet ne žymiai tiek, kad daryti skirtingas išvadas. Kompanijos „NetMechanic“ yra paskaičiuota[34], jog mes prarandame 1/3 lankytojų vien tik dėl to, jog žiniatinklio puslapis užkraunamas per lėtai. Remiantis ir vadovaudamiesi moksliniu tiriamuoju darbu, žiniatinklių optimizuotojai išvengs šios gana liūdnos statistikos. Dėl aiškių priežasčių nebuvo galimybės patikrinti, kokie optimizavimo būdai yra optimaliausi didesniems žiniatinklių portalams. Procentiškai visas optimizavimo našumas priklauso ne nuo to, su kokiais programavimo įrankiais

mes dirbame, o kokį kieki duomenų bandome apdoroti. Kuo duomenų kiekis yra didesnis, kuo duomenų pateikimas yra sudėtingesnis, tuo skirtingi optimizavimo būdai bus našesni.

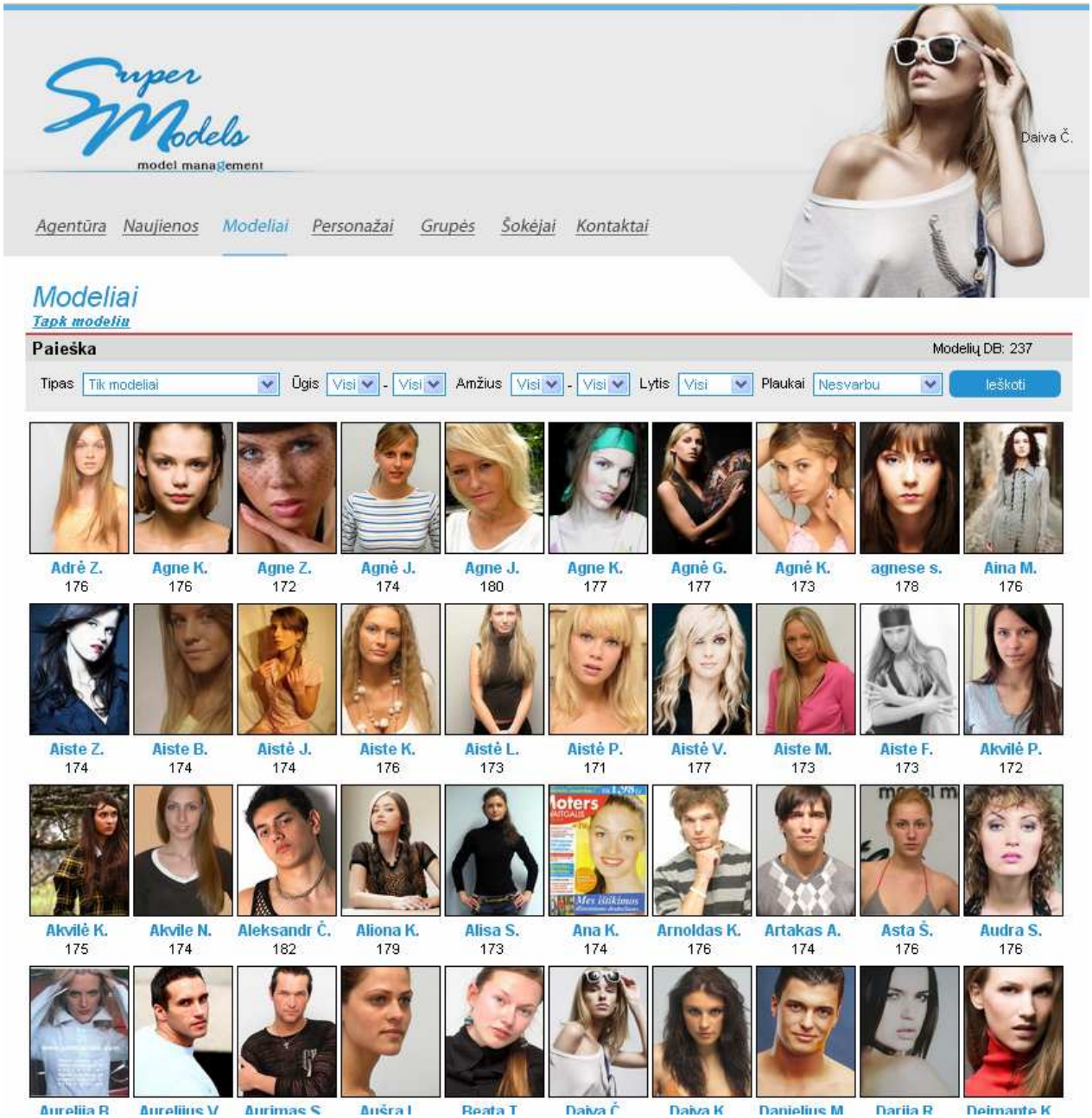
Literatūros sąrašas

1. Microsoft Official Course – 2310b Developing Microsoft ASP.NET Web Application Using Visual Studio .NET, USA - 20020921
2. Owens, Kevin - Programming Oracle Triggers And Stored Procedures, USA California – 20040206
3. http://www.microsoft.com/learning/syllabi/en-us/2310Bfinal.mspx
4. http://www.oracle.com/technology/pub/articles/hull_asp.html
5. http://www.mssqlcity.com/Articles/Compare/sql_server_vs_oracle.htm#part_5_1
6. https://www.microsoft.co.ke/lietuva/developertools/vs.mspx
7. http://www.vbip.com/books/1861003927/chapter_3927_12.asp
8. http://lt.wikipedia.org/wiki/Turinio_valdymo_sistema
9. http://www.seosite.lt/svetaines-patogumas/ivadas-i-tinklalapiu-patoguma.html
10. www.postgresql.org
11. http://www.microsoft.com/sql/prodinfo/compare/ibm/db2v8.mspx
12. http://lt.wikipedia.org/wiki/oracle
13. http://lt.wikipedia.org/wiki/mysql
14. http://moxliukas.esu.as/ruby/knyga/index.html
15. http://gedmin.as/study/python/index-lt.html
16. http://lt.wikipedia.org/wiki/python
17. http://www.lkka.lt/~g.rastauskiene/paskaita/internet/3_6.htm
18. http://amix.dk/blog/viewEntry/101
19. http://lt.wikipedia.org/wiki/Transakcija
20. http://en.wikipedia.org/wiki/Connection_pool
21. http://www.apachetutor.org/dev/reslist
22. http://www.devshed.com/c/a/PHP/Output-Caching-with-PHP/
23. http://www.mysqlperformanceblog.com/2006/11/12/are-php-persistent-connections-evil/
24. http://www.mysql.com/news-and-events/newsletter/2002-11/a0000000086.html
25. http://www.apachetutor.org
26. http://forums.devshed.com/
27. http://www.youngcoders.com/
28. http://www.programmersresource.com/forum/

29. http://www.netmechanic.com/products/powerpackFAQ.shtml
30. http://www.ipersec.com/index.php/2006/05/30/benchmarking-php-accelerators/
31. http://www.php-accelerator.co.uk/performance.php
32. http://itst.net/654-php-on-fire-three-opcode-caches-compared
33. http://itst.net/wp-content/uploads/2006/10/PHP%20Bytecode%20Cacher%20Review.html
34. http://www.netmechanic.com/news/vol7/beginner_no9.shtml

Priedai

1. Priedas. Žiniatinklinis tinklalapis www.supermodels.lt



2. Priedas. Žiniatinklinio tinklalapio http://www.supermodels.lt/lt/portfolios.php kodas

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>Modeliai iš agentūros "Supermodels"</title><meta name="keywords"
content="modeliai, manekenės, manekenes, foto modeliai, fotomodeliai" /><meta
name="description" content="modeliai, manekenės, foto modeliai iš agentūros
'Supermodels'" /><meta name="robots" content="index,follow" /><meta
name="copyright" content="UAB 'Elektroniniai sprendimai', Lietuva" />
<meta name="author" content="Erikas Matijosius, Lietuva" />
<link rel="stylesheet" type="text/css"
href="http://www.supermodels.lt/lt/css/main.css" media="screen" />
<!--[if lt IE 7]>
<link rel="stylesheet" type="text/css"
href="http://www.supermodels.lt/lt/css/ie6.css" media="screen" />
<![endif]-->
<link rel="stylesheet" type="text/css"
href="http://www.supermodels.lt/lt/css/lightbox.css" media="screen" />
<script type="text/javascript"
src="http://www.supermodels.lt/lt/js/js.js"></script>
<script type="text/javascript"
src="http://www.supermodels.lt/lt/js/prototype.js"></script>
<script type="text/javascript"
src="http://www.supermodels.lt/lt/js/scriptaculous.js?load=effects"></script>
<script type="text/javascript"
src="http://www.supermodels.lt/lt/js/lightbox.js"></script>
<script type="text/javascript"
src="http://www.supermodels.lt/lt/js/progress.js"></script>
<script src="http://www.google-analytics.com/urchin.js" type="text/javascript">
</script>
<script type="text/javascript">
_uacct = "UA-1830851-8";
urchinTracker();
</script>
<!--[if lt IE 7.]>
 <script defer type="text/javascript">
  var site_url = 'http://www.supermodels.lt/';
 </script>
 <script defer type="text/javascript"
src="http://www.supermodels.lt/js/pngfix.js"></script>
 <![endif]-->
</head>
<body>
<div class="top-line"><center><div class="content">
<a href="http://www.supermodels.lt" class="logo"><img
src="http://www.supermodels.lt/lt/img/logo.gif"></a>
<div class="jurgita">
<img src="http://www.supermodels.lt/img/kampai/daiva.gif" alt="" />
</div>
<div class="menu">
<div class="agency"><a
href="http://www.supermodels.lt/lt/agency.php"><em></em>Agency</a></div>
<div class="news"><a
href="http://www.supermodels.lt/lt/news.php"><em></em>News</a></div>
```