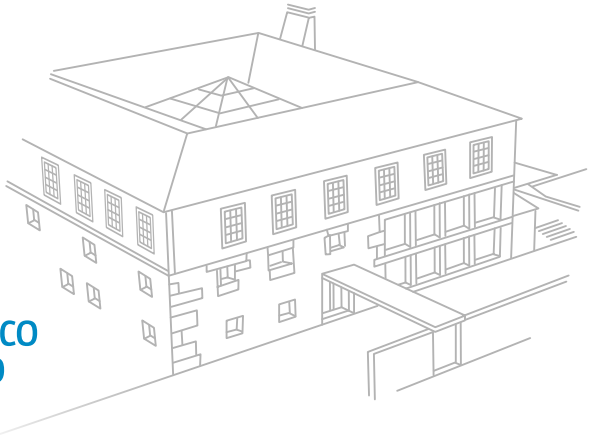


ESTGF | POLITÉCNICO DO PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Técnicas para melhoria de performance em aplicações CMS

DESIGNAÇÃO DO MESTRADO
Mestrado em Engenharia Informática

AUTOR
Armindo Carlos Martins Pereira

ORIENTADOR(ES)
Ricardo André Fernandes Costa
João Paulo Ferreira de Magalhães

ANO
2011

AGRADECIMENTOS

Gostaria de agradecer á ESTGF por me ter proporcionado as condições necessárias para efectuar a dissertação, numa área pela qual tenho um gosto especial, gostava também de agradecer a todas as pessoas do Centro de Informática, aos meus colegas do CIICESI e um agradecimento especial ao Professor João Paulo Magalhães, ao Professor Doutor Ricardo Costa e á Professora Doutora Dorabela Gamboa por terem possibilitado a elaboração da presente tese.

RESUMO

A *performance* das aplicações Web é um tópico extremamente importante. Estudos indicam que 37% a 49% dos utilizadores que sofrem de problemas de *performance* abandonam o *site* ou mudam para *sites* da concorrência. Destes, 77% partilham a sua experiência com outros prejudicando ainda mais a reputação da empresa. Para além do impacto na reputação há o impacto financeiro. Por exemplo um relatório tornado público recentemente indica que 100 milisegundos de perda de latência no acesso á Amazon contribui para uma quebra de 1% nas vendas.

Considerando o papel das aplicações Web para o negócio das empresas, é vital adoptar um conjunto de boas práticas ao nível da programação e ao nível da infra-estrutura de forma a garantir um bom desempenho da aplicação e consequentemente a satisfação dos utilizadores. São já vários os trabalhos feitos na área da optimização e análise de *performance* das aplicações Web. No entanto, e considerando a complexidade inerente às aplicações, envolvendo inúmeros componentes, tipos de *workload* dinâmicos e constantes mudanças ao nível do *software* e infra-estrutura torna-se difícil garantir os seus níveis de desempenho de forma constante.

Neste trabalho foca-se particularmente as questões de infra-estrutura que visam melhorar a *performance* das aplicações Web. O caso em estudo contempla técnicas como o balanceamento de carga, *caching* de conteúdo estático e *caching* de conteúdo dinâmico. Estas técnicas foram comparadas para avaliar o seu impacto na *performance* das aplicações Web. O tipo de aplicações Web escolhido foram os *Content Management Systems* - CMS. Para o processo de avaliação foi criado um ambiente de testes e utilizadas ferramentas que permitem a simulação de vários utilizadores a utilizar a aplicação. Por teste foi feita a recolha de dados que permite avaliar o nível de desempenho de cada uma das técnicas de forma individual e ainda o desempenho resultante da combinação entre as mesmas. As conclusões quer ao nível do esforço para adoptar as diferentes técnicas e o ganho de *performance* obtido por cada uma são interessantes e apresentadas ao longo do trabalho.

PALAVRAS-CHAVE

Aplicações Web, Balanceamento de carga CMS, *Caching*, Escalabilidade, *Performance*.

ABSTRACT

The performance of Web applications is topic of great relevance. Reports of the field indicate a straight relationship between the Web application performance and the profit and image of the companies. Recent studies, suggest that 37% to 49% of the users that experience slow response times while interacting with Web application will switch to competitors or abandon the site. Some of these will also share their negative experience with others. Besides the diminished reputation, the revenue impact is also considerable. According to a recent presentation a delay of just 100ms in the response time of Amazon web site corresponds to a reduction of 1% in sales.

Considering such negative impacts it is mandatory to adopt good practices of both: programming and infrastructure. Although, and considering the complexity behind applications this is a challenge task. Nowadays, applications involve multiple components provided by different entities, are subject of regular updates and different usage patterns. Such characteristics difficult the appropriate evaluation of the applications before they are deployed into production emphasizing the risk of performance issues.

In this project we compare different techniques applied at the infrastructure level. We adopted CMS applications and have compared the different techniques - load balancing, caching of static and dynamic content - to access their advantage in terms of performance-gain. All the tests were done using a test-bed environment conveniently prepared. The user requests are simulated from a real user usage capture. For each technique is captured and evaluated the response time experienced by the end-user and the number of requests attended by the web server per unit of time. A combination of techniques is also discussed.

The type of analysis and evaluation provided in this report is important to understand which sort of techniques contributes to improve the performance results considering the requests type and also provide IT staff with some techniques to pragmatically reduce the response time that some users may experience while interacting with the Web applications.

KEYWORDS

Web applications, Load balancing, CMS, Caching, Scalability, Performance

Índice

AGRADECIMENTOS 1

RESUMO 2

ABSTRACT 3

KEYWORDS 3

Índice Figuras 6

Índice Tabelas 7

Abreviaturas e Acrónimos 8

1.1. Estrutura do relatório 14

2.1. Optimização ao nível da programação 15

2.2. Web *Browsers* 17

2.3. *Tuning* do Sistema Operativo e de Rede 18

2.4. *Tuning* do Servidor Web 18

2.5. Base de dados 19

2.6. Optimização ao nível da infra-estrutura 20

2.7. Optimização ao nível do CMS 21

3.1. Ambiente de testes 24

3.2. Ferramentas Utilizadas 26

3.3. Cenários de teste 27

4.1. *Performance* antes da adopção das técnicas 31

4.1. Resultados da *performance* com a adopção das técnicas 34

4.1.1. Latência da *Index Page* do Drupal 36

4.1.2. Latência da *Search Page* do Drupal 39

4.1.3. Latência da *Index Page* do Joomla 42

4.1.4. Latência da *Search Page* do Joomla 45

4.2. Análise de resultados 48

A.1. Quadro comparativo entre Drupal e Joomla 57

B.1. Instalação e configuração de *software* 61

B.1.1. Apache + PHP 61

B.1.2.MySQL 61

B.1.3. HAProxy 61

B.1.3.Squid 63

B.1.4.Memcache 64

B.1.5.Optimizações 65

Índice Figuras

Figura 1 - Volume Mensal de *Queries* sobre CMS no Google (versão inglesa) 12
Figura 2 - Cenário Base..... 24
Figura 3 - Cenário Avançado..... 25
Figura 4 - Comparativo de Latência 32
Figura 5 - "Peso" da "Index Page" dos CMS..... 32
Figura 6 - Comparativo de *Throughput*..... 33
Figura 7 - Comparativo de Desvio Padrão 34
Figura 8 - Comparativo de Latência 36
Figura 9 - Comparativo de *Throughput*..... 37
Figura 10 - Comparativo de Desvio Padrão 38
Figura 11 - Comparativo de Latência 39
Figura 12 - Comparativo de *Throughput*..... 40
Figura 13 - Comparativo de Desvio Padrão 41
Figura 14 - Comparativo de Latência 42
Figura 15 - Comparativo de Throughput..... 43
Figura 16 - Comparativo de Desvio Padrão 44
Figura 17 - Comparativo de Latência 45
Figura 18 - Comparativo de Throughput..... 46
Figura 19 - Comparativo de Desvio Padrão 47
Figura 20 - Comparativo de Latência 48
Figura 21 - Comparativo de Throughput..... 49
Figura 22 - Comparativo de Latência 50
Figura 23 - Comparativo de Desvio Padrão 66
Figura 24 - Comparativo de *Throughput*..... 67
Figura 25 - Comparativo de Desvio Padrão 68

Índice Tabelas

Tabela I - Média de Download Semanais de cada CMS..... 12
Tabela II - Características Físicas do Ambiente de Testes 25
Tabela III - Tabela Comparativa entre Drupal e Joomla 60

Abreviaturas e Acrónimos

BIOS – Basic Input Output System

CDN – Content Delivery Network

CentOS – Community Enterprise Operating System

CI – Centro de Informática

CIICESI – Centro de Inovação e Investigação em Ciências Empresariais e Sistemas de Informação

COTS - Commercial Off-The-Shelf components

CMS – Content Management System

CPU – Central Processing Unit

CSS – Cascade Style Sheets

GB – Gigabyte

GHz – Giga Hertz

GIF – Graphics Interchange Format

HTML – Hyper Text Markup Language

HTTP – Hyper Text Transfer Protocol

IO – Input & Output

IP – Internet Protocol

IT – Information Technology

LB – Load Balancer

JPEG - Joint Photographic Experts Group

MC – Memcached

MPM – Multi-Processing Modules

MS – Milisegundos

OSI – Open Systems Interconnection

PNG – Portable Network Graphics

RAM – Random Access Memory

RP – Reverse Proxy

RSS – Really Simple Syndication

SW – Servidor Web

TCP – Transmission Control Protocol

Capítulo 1
Introdução

A *performance* das aplicações é vista como um aspecto crítico e representativo da qualidade, reputação e retorno financeiro das empresas. Estudos como [1] indicam que o desempenho financeiro das empresas cujo negócio depende de aplicações Web está estritamente ligado á qualidade de serviço na qual se inclui a *performance* das aplicações. Um tipo de aplicações onde a *performance* tem sido amplamente debatida é o das aplicações Web. Este debate centra-se essencialmente em três áreas de estudo:

1. a identificação de cenários e análise dos impactos derivados da perda de *performance*;
2. a detecção atempada de degradação de *performance* e respectiva recuperação;
3. a análise sobre o retorno (número de visitas, financeiro) obtido em função da *performance* da aplicação Web.

Em cada uma destas áreas encontramos referências que reforçam a importância do assunto. Por exemplo, e no que diz respeito aos impactos derivados da perda de *performance*, Greg Linden disse recentemente, numa apresentação pública [2], que uma perda de *performance* de apenas 100ms na utilização do *site* da Amazon representa uma quebra financeira de 1%, e de forma semelhante, um aumento no tempo de resposta da *index page* do Google implica uma perda de tráfego na ordem dos 20%. Sobre a detecção atempada e respectiva recuperação, é de salientar um estudo em [61] que indica que 65% das falhas reportadas pelos utilizadores, considerando três *sites* de referência, podem ser evitadas ou reduzidas se for feita uma detecção atempada dos problemas.

O retorno obtido pela melhoria de *performance* é também relevante. Por exemplo, Stoyan Stefanov da Yahoo anunciou em [60] que verificou um aumento do tráfego na ordem dos 9% por cada 400 milissegundos de melhoria obtida. Em [6] e [7] são também apresentados os ganhos, em termos de *performance* e escalabilidade, obtidos após a optimização da infra-estrutura de suporte ao Twitter e Youtube. Ainda, Paul Roy e o Alex Polak da Microsoft, focam as melhorias obtidas no número de páginas visualizadas do MSN (1.2% (search) e 0.5% (outras)) após reduzir o tempo de execução do *JavaScript* [59].

Por cada uma das área de estudo apresentadas, os esforços para maximizar a *performance* das aplicações Web divide-se entre melhorias de nível aplicacional e/ou

melhorias ao nível da infra-estrutura. Steve Souders é uma referência na área de *performance* de aplicações Web e nos seus livros [3,4] apresenta várias técnicas e desafios, sobre o desenvolvimento das aplicações Web que visam a sua melhoria em termos de *performance*. As recomendações são incontestáveis, no entanto, e motivado pelo crescente número de componentes que as aplicações integram (abordagem conhecida por COTS [5]), a dependência de serviços de terceiros e o crescente número de utilizadores faz com que muitas das vezes ocorram problemas de *performance* inesperados.

O papel da infra-estrutura que suporta as aplicações Web é algo que também não pode ser descurado. Mecanismos de *cache* e balanceamento de carga estão ao dispor e são explorados pelo *IT staff* para garantir uma maior escalabilidade e *performance* das aplicações que correm sobre a infra-estrutura.

Os já referidos exemplos do John Adams do Twitter [6], Cuong Do do Youtube [7], e Reza Behforooz do GTalk [8] são demonstrativos e cativantes para perceber o papel que os mecanismos de balanceamento de carga, *cache* de conteúdo estático e dinâmico têm na *performance* e escalabilidade dos serviços disponibilizados pelas aplicações.

Se do lado da programação há um conjunto de boas práticas a seguir, do lado da optimização de infra-estrutura é um enorme desafio saber em antemão qual é o custo/benefício na adopção de um mecanismo. Deste modo, importa saber e fornecer aos administradores de sistemas análises de custo/benefício que lhes permitam perceber as vantagens na adopção de um determinado mecanismo e ainda pela combinação de mecanismos. Neste trabalho vai-se focar este tipo de análise. Partindo de um tipo de aplicação Web muito utilizada hoje em dia, como é o caso dos *Content Managment Systems* (CMS), vai-se dotar a infra-estrutura de suporte aplicacional de mecanismos de balanceamento de carga, *cache* de conteúdo estático e *cache* de conteúdo dinâmico e auferir o ganho em termos de *performance* que se obtém com cada um dos mecanismos de forma isolada e pela combinação de mecanismos.

Estas aplicações são usadas pelas organizações de forma melhorar a comunicação entre utilizadores, reduzir a replicação de dados e controlar o acesso a estes. Para além disso os CMS também são utilizados como plataforma de suporte ao negócio das organizações. Um inquérito da empresa Water & Stone sobre a quota de mercado dos CMS [9] mostra de forma clara a quantidade de pessoas/entidades a adoptar ferramentas CMS e ainda o peso relativo dos diferentes tipos de CMS. Uma das análises efectuadas, é mostrada na Tabela I, e consiste na medição da quantidade de *downloads* semanais feitos para cada uma das ferramentas de CMS.

CMS	Valor médio de *downloads* por semana (em milhares)	Origem dos dados
Wordpress	433,767	Wordpress Download Center
Joomla	189,429	JoomlaCode.org
Drupal	62,500	Drupal Team
Umbraco	5,670	Umbraco Team
eZ Publish	5,612	eZ Publish Team
CMS Made Simple	4,903	Cmsmadesimple.org
SilverStripe	2,500	SilverStripe Team
e107	2,242	e107.org
Xoops	1,209	Xoops.org
TikiWiki	373	TikiWiki.org
phpWebSite	347	Phpwebsite.appstate.edu
Typo3	100	Sourceforge.net
Alfresco	57	Alfresco.com

Tabela I - Média de Download Semanais de cada CMS

Como se verifica pela tabela I o volume de *downloads* é elevado. Segundo o mesmo relatório, o número de projectos realizados por *freelancers*, para os três primeiros CMS apresentados na Tabela I, ascendeu a 10280 no ano de 2009. Outro indicador sobre a procura de CMS resulta da análise sobre o volume mensal de consultas capturadas pelo Google, (versão inglesa), entre Junho de 2008 e Junho de 2009. Tal como ilustrado na Figura 1, os *open* CMS lideram e são encabeçados de forma destacada pelos projectos Joomla [65] e Drupal [66].

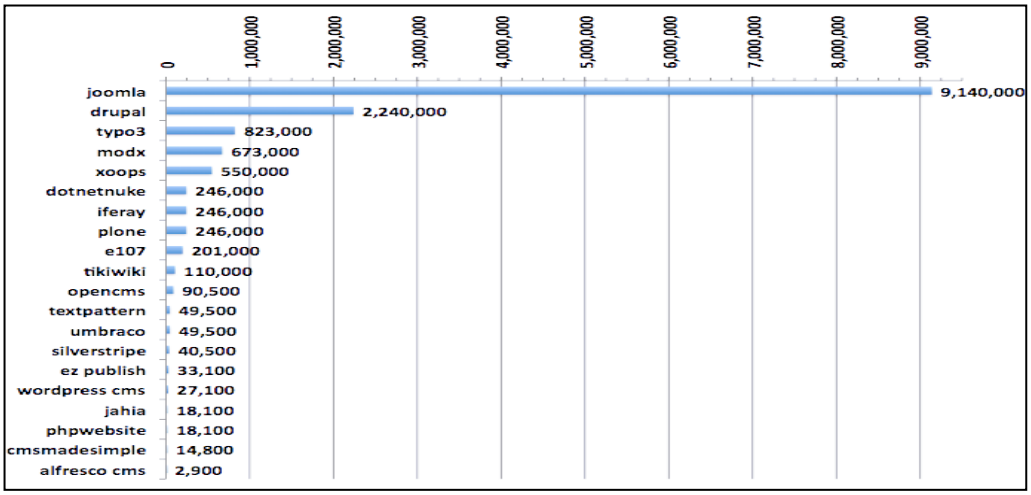


Figura 1 - Volume Mensal de *Queries* sobre CMS no Google (versão inglesa)

Realçado o interesse dos CMS como meio de suporte ao negócio das empresas, é importante analisar os aspectos de *performance* que se podem extrair deste tipo de aplicações. Através de uma simples pesquisa efectuada nos *bugtrackers* de duas das ferramentas de CMS mais utilizadas (Joomla e Drupal) verifica-se que existem aspectos que necessitam de atenção. A título de exemplo, no *bugtracker* do CMS Drupal [10] existe actualmente mais de 2800 registos abertos que reportam problemas de *performance* e no Joomla [11] também se verifica a existência de um elevado número de registos reportando o mesmo tipo de problema. É importante referir que alguns dos registos são repetidos, não obstante o elevado número e o tempo de resposta aos registos alertam para a adopção de alternativas que permitam melhorar a *performance* das aplicações e reduzir a sensação de "lentidão" reportada pelos utilizadores.

Pelo exposto acima nomeadamente o elevado número de projectos realizados por *freelancers* e o número de registos abertos associados á *performance*, focou-se a atenção em mecanismos que permitam melhorar a *performance* de aplicações do tipo CMS. Deste modo irão ser comparadas diferentes técnicas que actuam ao nível da infra-estrutura e destinam-se à melhoria da *performance* e escalabilidade de aplicações Web. A motivação para esta análise é a de perceber até que ponto a adopção dessas técnicas é vantajosa em termos de ganho de *performance* versus complexidade e custo relativo de adopção.

Em concreto vai-se:

- Analisar os problemas de *performance* dos CMS;
- Analisar a *performance* resultante na perspectiva do utilizador final;
- Adoptar mecanismos de balanceamento de carga;
- Adoptar mecanismos de *caching* de conteúdo estático;
- Adoptar mecanismos de *caching* de conteúdo dinâmico;
- Conjugar as diferentes técnicas e efectuar as respectivas análises.

O estudo começará por testar o funcionamento dos CMS partindo de uma configuração d*efault*. A forma típica de navegação dos utilizadores será capturada através de ferramentas que permitem gravar o comportamento dos utilizadores para mais tarde replicar. Os CMS serão exercitados, através do *workload* capturado, com diferentes níveis de utilização. Adicionalmente serão capturadas diversos parâmetros de sistema e da aplicação relacionados com a *performance* e utilização de recursos. Estes parâmetros serão usados para posterior análise.

Depois, e numa segunda fase, serão adicionados os mecanismos que visam a melhoria da *performance* e, através da execução repetida do *workload* será analisado o

ganho de *performance* obtido pela adopção de cada mecanismo. Numa terceira fase será efectuada a combinação de vários mecanismos. Da combinação pretende-se analisar até que ponto existem ganhos em combinar diferentes mecanismos e ainda explorar cenários em que a adopção de um determinado mecanismo pode anular os ganhos obtidos por outros.

Será feita uma análise dos resultados focando as vantagens/desvantagens dos diferentes mecanismos com o objectivo de proporcionar ao IT *staff* uma análise sumária que lhes permita perceber o custo/benefício da adopção destas soluções.

1.1. Estrutura do relatório

Este documento está organizado em cinco capítulos. No capítulo 2 são feitas referências ao trabalho da área da *performance* em aplicações Web. No capítulo 3 é apresentado o ambiente de testes e o plano de testes a executar. No capítulo 4 são apresentados os resultados por tipo de mecanismo utilizado. Finalmente no quinto capítulo é feita a conclusão sobre o trabalho realizado.

Capítulo 2
Estado da Arte

A optimização da *performance* de aplicações Web envolve diversas áreas entre as quais se destacam:

- Optimização ao nível da programação
- Web *Browser*
- *Tuning* do Sistema Operativo e Rede
- *Tuning* de aplicações
- Base de dados
- Optimização ao nível da infra-estrutura
- Optimização ao nível do CMS

2.1. Optimização ao nível da programação

Uma área de estudo na *performance* das aplicações Web é a da análise e melhoria de aspectos relacionados com a programação das aplicações Web. Steve Souders é uma referência na área e ao longo dos seus livros [3,4] destacam-se algumas das boas práticas a considerar para o desenvolvimento de aplicações Web. Estas práticas são hoje em dia auxiliadas por diversas ferramentas como por exemplo: o YSlow [12], ou o Google Page Speed [13] que sugerem que cuidados uma determinada página necessita, facilitando assim o trabalho de análise e determinação das partes de uma aplicação Web que mais carecem de atenção. Dentro do conjunto de boas práticas destacam-se as seguintes:

- Diminuição dos pedidos HTTP;
- Compressão dos diversos recursos utilizados na página Web;
- Incluir as folhas de estilo (CSS) no topo da página;
- Incluir as *scripts* (*Javascript*) no fundo da página;
- Adicionar cabeçalhos de expiração ao pedido HTTP;
- Optimizar as imagens utilizadas na página;
- *Preload* e *Postload* de componentes

Quando se visualiza uma página Web cerca de 80% [14] do tempo de espera de carregamento da página é gasto com o *download* dos vários componentes existentes na página desde imagens, folhas de estilos, *javascript* entre outros. Reduzindo o número de componentes a carregar irá naturalmente reduzir o número de pedidos HTTP e o tempo de carregamento das páginas. Uma forma de reduzir este número consiste em combinar as várias folhas de estilo utilizadas em apenas uma, e o mesmo para os vários *scripts* utilizados na página. Desta forma os vários pedidos serão agregados em dois pedidos optimizando o tempo de carregamento da página.

O tempo de carregamento de uma página está ligado ao tempo de *download* de cada componente e quanto maior for o componente em causa maior será o tempo de carregamento da página resultando numa menor *performance* da mesma. Uma forma de evitar que os componentes sejam descarregados com tamanho excessivo é utilizar técnicas de compressão, nomeadamente GZip. Reduzindo o tamanho dos componentes irá reduzir o tamanho da página resultando assim num carregamento mais rápido. É no entanto importante referir que a compressão faz com que a utilização do CPU suba podendo eventualmente reduzir a capacidade do mesmo. Este *trade-off* precisa de ser acautelado e está dependente do tipo de recursos disponíveis e da análise tempo de comunicação *versus* tempo de computação.

Incluir as folhas de estilo no topo da página faz com que as páginas carreguem progressivamente, isto é o *browser* irá mostrar o conteúdo ao qual já fez *download* dando desta forma um *feedback* progressivo aos utilizadores contribuindo para uma melhoria da utilização. O carregamento progressivo é particularmente importante para páginas com muito conteúdo ou para ligações á Internet lentas. Quanto ao código em *javascript*, é preferível colocar no fundo da página. Assim o *download* do *javascript* não interfere com o *feedback* visual da página, (pois quando os ficheiros de *javascript* são descarregados já todos os restantes componentes foram descarregados) diminuindo-se o tempo de carregamento da mesma.

Á medida que as páginas se tornam mais atractivas também se tornam mais "pesadas". Ao utilizar mais componentes faz com que estes tenham de ser descarregados aumentando o tempo de carregamento da página. Para melhorar o desempenho a este nível devem-se adicionar cabeçalhos com data de expiração permitindo assim que os componentes fiquem na *cache* do *browser* o que significa que em futuras visitas os componentes serão respondidos a partir de uma *cache* reduzindo-se o tempo de *download* e a sobrecarga sobre o servidor principal.

As imagens a ser utilizadas numa determinada página podem também ser optimizadas. Sobre a optimização das imagens deve-se considerar:

- se as imagens tiverem o formato GIF pode-se verificar se o tamanho da paleta de cores corresponde ao número de cores da imagem;
- converter GIFs para PNGs contribui para reduzir o tamanho ocupado pelas imagens, facilitando o tempo de transferência e *rendering* no Web *browser*;
- utilizar ferramentas ImageMagick [15], PNGCrush [16], e Smush.it [17] para optimização de PNGs e JPEGs.

O *preload* de componentes significa o carregamento de por exemplo imagens que não são necessárias na página em questão mas no entanto serão necessárias em outras páginas. O carregamento é normalmente efectuado quando o *browser* se encontra inactivo. Por *postload* de componentes entende-se que para o funcionamento mínimo de uma determinada página devem apenas ser carregados os componentes estritamente necessários á mesma. Neste tipo de carregamento os restantes componentes só são carregados após o carregamento da página principal. Por exemplo: um mecanismo de *drag&drop* apenas deve ser carregado após a página ser totalmente carregada.

2.2. Web *Browsers*

Sendo o Web *Browser* a aplicação através da qual se visualizam as páginas Web, é natural que estes também afectem os tempos de carregamento de uma determinada página. Desde a forma como processam *javascript* passando pela forma como interpretam o HTML e a forma como os vários *add-ons* interagem com a página. Nos estudos apresentados em [18,19] verifica-se através de vários testes a *performance* dos *browsers* bem como a sua gestão de memória, um estudo complementar em [20] mostra a forma como os *browsers* se comportam no que diz respeito á sua *cache*, javascript e *rendering* das páginas. Nos estudos referidos previamente são utilizadas diversas ferramentas [21,22,23,24,25,26] para testar a forma como os *browsers* influenciam os tempos de carregamento das páginas e a funcionalidade das mesmas. A título de exemplo verificou-se que a extensão Firebug [27] para o Firefox [28] torna a navegação em média 79% mais lenta [29].

2.3. *Tuning* do Sistema Operativo e de Rede

Por *tuning* entende-se a modificação de algo com o intuito de aumentar a sua *performance,* neste caso efectua-se a modificação ao sistema operativo de forma a incrementar a *performance* das aplicações que sobre este correm. Estas modificações ocorrem ao nível da gestão de memória, gestão da rede e sistema de ficheiros.

Ao nível de rede e sistema de ficheiros uma modificação importante tem a ver com o aumento do número máximo de "*file descriptors*" (detalhes no Anexo B, mais precisamente no ponto 1.5). Este aumento vai permitir que sejam abertos mais ficheiros, no qual se incluem as ligações de rede, ou seja, o número de clientes que se podem ligar ao servidor, no entanto podem ser efectuadas mais modificações de forma a incrementar ainda mais a *performance* do sistema operativo [30,31].

Outro parâmetro de rede que desempenha um papel importante na performance do sistema é o TCP *KeepAlive timeout*. Este parâmetro indica o número de segundos que determinada ligação TCP irá ficar á espera de novos pedidos. Se uma ligação já estiver activa então não há necessidade de perder tempo com o estabelecimento das mesmas, reduzindo o tempo de resposta. Num cenário em que o número de utilizadores é elevado este parâmetro tem um efeito muito positivo, porque para além do cliente não ter que esperar pelo estabelecimento de uma nova ligação, o servidor fica menos ocupado com as tarefas de gestão de ligações.

2.4. *Tuning* do Servidor Web

O *tuning* de aplicações consiste no ajuste dos parâmetros aplicacionais e de sistema, de forma a incrementar os níveis de escalabilidade e *performance* das mesmas. No âmbito deste trabalho irá ser abordado o *tuning* do servidor Web.

O servidor Web é o principal responsável por receber os pedidos dos utilizadores e tratar da resposta aos mesmos. Para fornecer uma resposta podem ainda ser envolvidos outros servidores/serviços. Desta forma a optimização do servidor Web é de extrema importância para que sejam admitidos múltiplos utilizadores em paralelo e que os tempos de processamento de cada um se mantenha o mais estável possível. Entre os vários servidores Web disponíveis neste trabalho adoptou-se o Apache [55]. O Apache dispõe de vários parâmetros [32,33,34,35] que devem ser ajustados em função do tipo de aplicação, infra-

estrutura e forma de utilização da aplicação. Entre os principais parâmetros destacam-se o *KeepAlive* já descrito, o MPM que determina a forma como o Apache irá funcionar se em modo "*multi-thread*" ou em modo "normal", outra modificação que se pode efectuar reside nos módulos carregados, ou seja carregar apenas os módulos necessários, também se deve alterar o parâmetro *MaxClients* que indica o número máximo de pedidos simultâneos suportados pelo servidor, existem outros parâmetros que podem ser alterados mas no entanto estes estão dependentes do MPM seleccionado.

2.5. Base de dados

O *tuning* do sistema de gestão de base de dados e da própria base de dados aplicacional são igualmente importantes. Ao nível do sistema de gestão da base de dados é de vital importância definir correctamente o número máximo de ligações aceites. Este número corresponde ao número máximo de ligações suportadas em simultâneo e caso se esgote faz com que o cliente fique á espera que uma ligação fique livre, aumentando desta forma o tempo de latência ou introduzindo erros por falta de ligações Além do correcto dimensionamento destes valores no S.O. é fundamental também verificar a configuração do próprio sistema de gestão de base de dados.

A par de um bom modelo de dados, a gestão corrente da base de dados é também relevante. Cenários como a falta de índices em tabelas, a concorrência excessiva e ainda a dimensão da base de dados são aspectos que devem ser acautelados para evitar degradações de *performance.*

As opções ao nível do armazenamento usado pela base de dados são também relevantes. Por exemplo, nas bases de dados MySQL [56] a escolha dos tipos de armazemanento (InnoDB [62], myISAM [62], Memory [62] ou Archive [62]) têm impacto na *performance* e escalabilidade do sistema. A escolha apropriada do tipo de armazenamento está relacionada com o tipo de base de dados e a forma como esta é usada [63,64]. Por exemplo caso seja necessário que a base de dados seja *transaction-safe* e possua uma boa *performance* então uma opção seria utilizar o mecanismo InnoDB, caso o objectivo seja a performance e a informação não necessite de ficar armazenada então pode-se optar pelo mecanismo Memory.

Apesar do referido acima, existem outras alterações às configurações do MySQL que se podem fazer de forma a melhorar a *performance* do mesmo desde modificações no funcionamento da *cache* passando pela alteração dos *buffers*.

2.6. Optimização ao nível da infra-estrutura

Exemplos sobre optimização ao nível da infra-estrutura e resultados obtidos evidenciados nas apresentações feitas por John Adams do Twitter [5], Cuong Do do Youtube [6] e Reza Behforooz do GTalk [7]. Ao longo das suas apresentações são discutidas diversas técnicas, entre as quais se destacam o *caching* de conteúdo dinâmico e estático, os mecanismos de balanceamento de carga e ainda a utilização de técnicas de *clustering*.

O *caching* de conteúdo dinâmico é utilizado por exemplo para fazer *cache* de consultas efectuadas á base de dados Entre as técnicas de *caching* de conteúdo dinâmico mais adoptadas encontra-se o Memcache [43] e o EHCache [43]. Através deste tipo de técnicas as aplicações perguntam aos servidores de *cache* se, para uma dada *query* os dados estão em *cache*. Caso o resultado não se encontre em *cache* então a aplicação consulta a base de dados. O Memcache foi utilizado com sucesso pela equipa do Twitter [6] que com o uso deste sistema de *cache* conseguir reduzir aos tempos de latência sentidos pelo utilizador

Em termos de mecanismos de *caching* de conteúdo estático existem várias ferramentas disponíveis, entre as quais se destacam o Squid [44], NginX [45], Varnish [46]. O funcionamento destas é relativamente simples:

- recebem os pedidos dos clientes e verificam a sua *cache* a ver se têm o pedido;
- caso o tenham a resposta é encaminhada para o cliente;
- caso o pedido não exista em *cache* então o pedido é redireccionado para o servidor especificado nas suas configurações. O resultado é armazenado em *cache* para ser utilizado em futuros pedidos.

O balanceamento de carga serve para redistribuir os pedidos por vários servidores de acordo com as políticas de balanceamento definidas. As políticas de balanceamento mais comuns são: *roundrobin* [47], *leastconn, source.* Na política *roundrobin* os pedidos são encaminhados para os servidores de forma sequencial, isto significa que o 1º pedido do cliente A será encaminhado para o servidor A, o 2º pedido será encaminhado para o

servidor B e assim sucessivamente até não haver mais servidores, recomeçando de seguida este processo. Na política leastconn os pedidos dos clientes são encaminhados para o servidor com menos ligações activas no momento. Na política *source* tem-se em consideração o endereço IP do cliente, fazendo com que um IP ou grupo de IPs seja atribuído um servidor específico. Este tipo de balanceamento de carga é útil por exemplo para lidar com questões de localização, na qual os endereços de uma determinada origem podem ser processador por um servidor mais próximo.

O balanceamento de carga é normalmente usado para diminuir a carga sobre as máquinas e simultaneamente introduzir mecanismos de tolerância a falhas dado que quando um dos servidores falha os pedidos continuam a poder ser atendidos por outro servidor. É importante ainda referir a existência de mecanismos de balanceamento de carga de nível 4 e nível 7 do modelo OSI. A principal diferença entre eles reside no modo como os reencaminhamentos se processam. Enquanto no balanceamento de carga efectuado no nível 7 do modelo OSI os reencaminhamentos são baseados no tipo de aplicação, informação protocolar do nível da aplicação (ex: HTTP com filtro por *status code*), no nível 4 os reencaminhamentos são efectuados tendo por base os endereços IP origem/destino.

A utilização de *clustering* [48] é também um aspecto a destacar. Os tipos de *cluster* mais frequentes são: *clusters* de alto desempenho e *clusters* de alta disponibilidade. Num *cluster* de alto desempenho, como o nome indica é um tipo de *cluster* em que existem vários servidores que estão a trabalhar em paralelo na execução de uma mesma tarefa. Os *clusters* de alta disponibilidade consistem normalmente em servidores activos e passivos. Caso o servidor activo fique indisponível a computação/serviços são assumidos pelo servidor que até então se designava por passivo. Existem também *clusters* de nível aplicacional. Estes *clusters* são frequentes em infra-estruturas Web e são fundamentais para assegurar a alta disponibilidade das sessões dos clientes. Assim, perante a falha de um servidor Web, as sessões encontram-se replicadas podendo a mesma ser restabelecida de outro servidor sem que o utilizador final se aperceba da falha.

2.7. Optimização ao nível do CMS

Com o objectivo de melhorar a própria *performance* das aplicações, o *software* é normalmente dotado de mecanismos que lhes permitem manter estado sobre as tarefas

mais executadas, acelerando assim o tempo de reposta. No que diz respeito aos CMS estes trazem de raiz mecanismos de controlo de *cache* que podem ser activados. A utilização destes serviços para além de simples permite que os dados mais frequentemente acedidos permaneçam numa *cache* e sejam respondidos mais rapidamente quando comparado com o acesso tradicional à fonte de dados.

Nos CMS Drupal e Joomla a activação de *cache* é bastante simples, e o seu funcionamento é transparente para o utilizador final. No caso do Drupal existem 2 modos de funcionamento da *cache*: normal e *agressive,* a diferença entre estes 2 modos consiste no *load* e *unload* dos módulos activos sendo que no modo *agressive* isto não acontece daí este modo fornecer um aumento de *performance* quando comparado com o modo normal mas no entanto pode causar problemas com módulos criados por terceiros [49,50,51], por outro lado no Joomla existem 3 modos de efectuar *cache: page, view e module* [52], no modo de *page* quando a página é visualizada pela primeira vez é criada uma cópia desta sendo que quando a página é visitada novamente é enviada a cópia ultrapassando-se assim o processamento de código e as ligações á base de dados, por outro lado no modo de *view* e *module* são criadas cópias de porções da página.

Para além dos mecanismos de cache, os CMS incluem funcionalidades que permitem a junção das várias CSS num único ficheiro e também a junção de várias funções de *javascript* em apenas um ficheiro, além disso estas aplicações têm um vasto conjunto de módulos que permitem aumentar os níveis de desempenho da aplicação [36,37,38].

Capítulo 3
Análise de Performance em Aplicações CMS

Ao longo deste capítulo irá ser apresentado o ambiente de testes e a metodologia adoptada para proceder à avaliação sobre o ganho de *performance*, introduzido por diferentes técnicas implementadas ao nível da infra-estrutura. A avaliação foca aplicações em produção, ou seja:

- aplicações cuja optimização de código como via para solucionar um problema seria morosa;
- aplicações que por falha na adequação da infra-estrutura necessitam de melhorar de forma rápida e eficiente o seu desempenho.

A experiência reportada em [6,7,8] aponta para melhorias significativas ao nível do desempenho e escalabilidade das aplicações. Sendo os CMS aplicações muito utilizadas, assumindo mesmo o papel de aplicações do tipo *mission-critical* para algumas empresas torna-se importante auferir que ganhos se podem obter com a adopção de técnicas, como o balanceamento de carga e *cache* de conteúdo estático e dinâmico. Do ponto de vista operacional este estudo é também importante, ele contribui para que os administradores de sistemas e responsáveis por aplicações Web percebam os benefícios que podem esperar pela adopção das diferentes técnicas e da combinação das mesmas.

As vantagens e desvantagens, em termos de ganho de performance, introduzidas pelas técnicas de balanceamento de carga, *cache* de conteúdo estático e conteúdo dinâmico serão analisadas quer em separado quer em conjunto. Para efeitos de avaliação de desempenho foi criado um ambiente de testes controlado, permitindo assim anular possíveis efeitos indesejados de rede que levassem a um enviesamento na análise dos resultados. Nas próximas secções será apresentado o ambiente de testes base bem como a sua adaptação para acomodar as diferentes técnicas. São também apresentadas as ferramentas adoptadas nos testes e ainda os cenários contemplados pelos testes.

3.1. Ambiente de testes

Os testes serão feitos num ambiente criado para o efeito. O ambiente de testes base é ilustrado na Figura 2. Ele contempla uma arquitectura Web típica composta três camadas na qual os vários utilizadores fazem pedidos ao servidor Web onde esta a correr o CMS. O servidor Web interage com um servidor de base de dados.

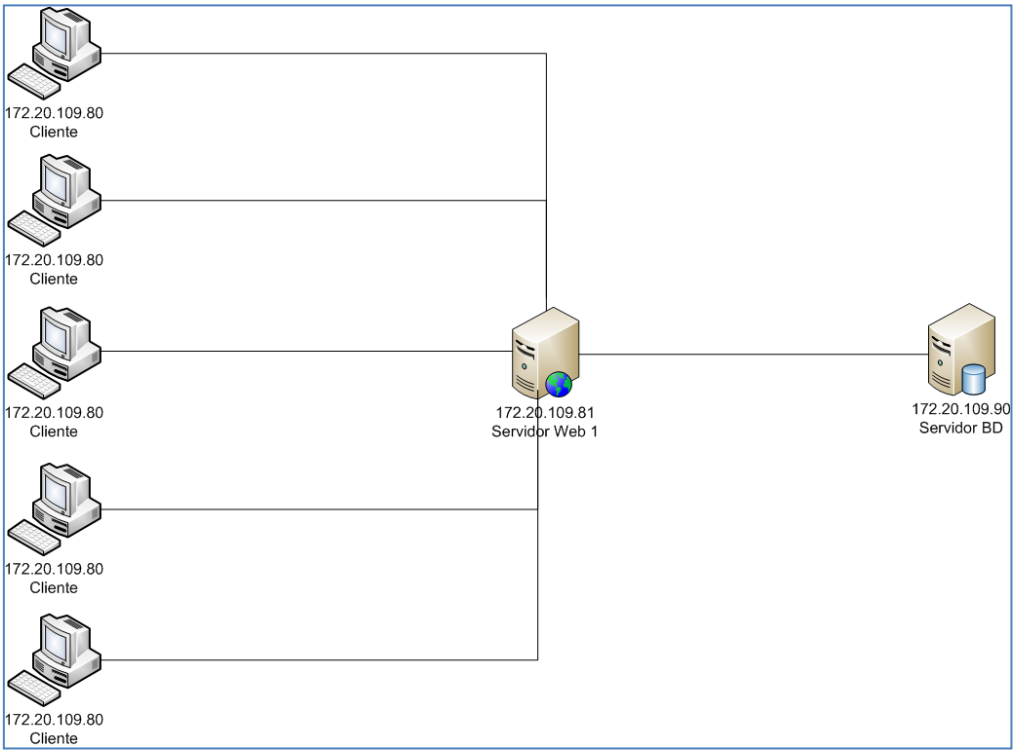


Figura 2 - Cenário Base

O cenário base será usado para avaliar a *performance* das aplicações antes de introduzir qualquer técnica de optimização de *performance*. À medida que se forem adicionando novos serviços a infra-estrutura de testes irá ser adaptada. Na Figura 3 é ilustrado o cenário final. Como se pode ver na figura o número de camadas sobe quando comparado com o cenário base, isto acontece pelo facto de ter sido adicionado um servidor para *cache* de conteúdo estático, um serviço de balanceamento de carga e serviços de *cache* de conteúdo dinâmico. Também se pode verificar que o número de servidores Web a servir a aplicação aumentou, permitindo assim testar o efeito de balanceamento de carga pelos servidores. A arquitectura apresentada na Figura 3 é a que se aproxima dos ambientes produtivos adoptados pela indústria.

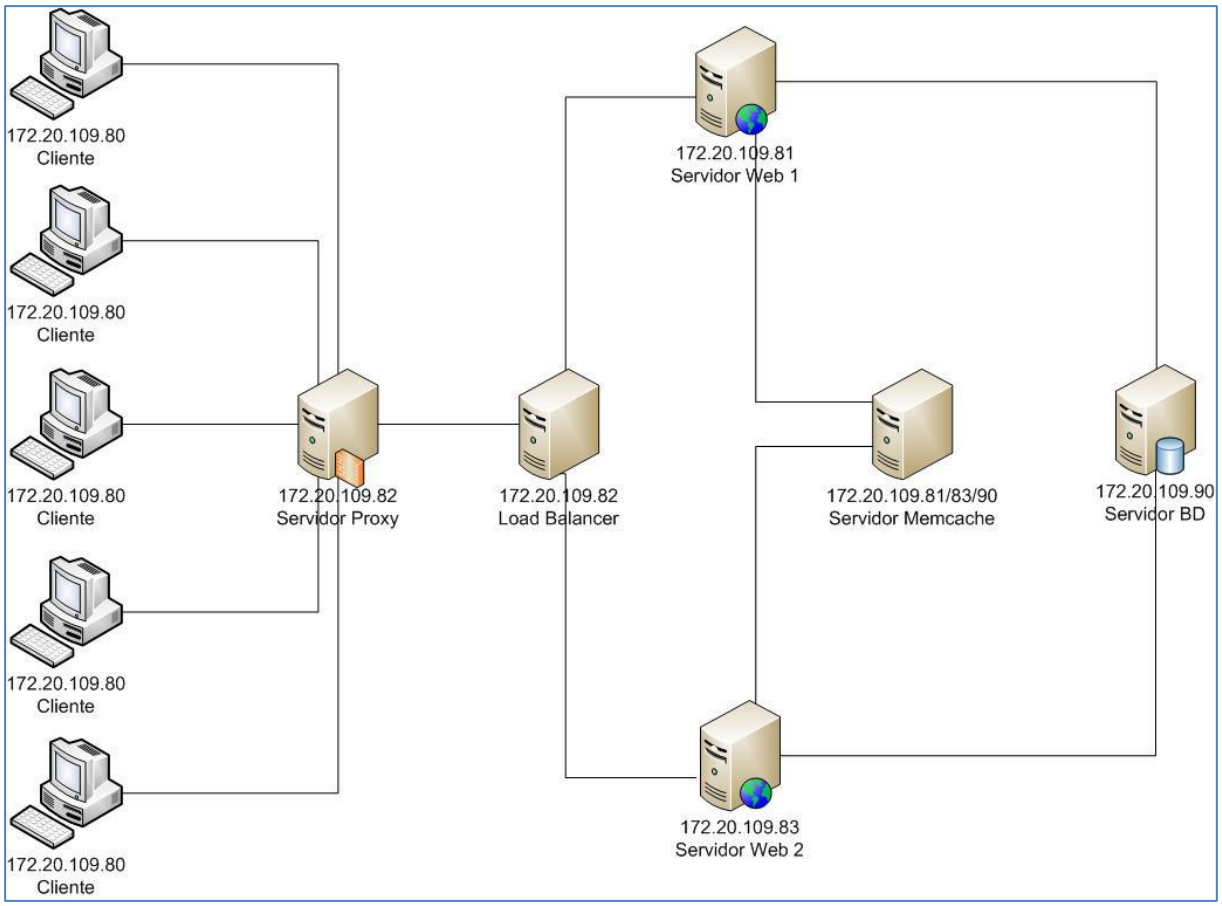


Figura 3 - Cenário Avançado

As características físicas das máquinas são apresentadas na tabela II. Os detalhes da instalação e configuração podem ser consultados no Anexo B.

	CPU	Memória	SO
Clientes	Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz	3,48 GB	Windows XP
Servidor(es) Web	Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz	4,0 GB	CentOS 5.4
Cache de conteúdo estático	Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz	4,0 GB	CentOS 5.4
Balanceador de carga	Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz	4,0 GB	CentOS 5.4
Base de Dados	Pentium III Xeon 1 GHz (x2)	3,2 GB	CentOS 5.4

Tabela II - Características Físicas do Ambiente de Testes

Como aplicações de *benchmark* adoptou-se o Drupal e o Joomla. Na base da escolha pesou o facto de ambos os CMS serem *open-source*, terem de acordo com a Figura 1 e Tabela I a maior quota de mercado, e ainda o facto destes dois CMS proporcionarem o mesmo tipo de funcionalidades (uma lista de funcionalidades sobre estes dois CMS é apresentada no Anexo A). Um CMS que reúne duas das condições (*open-source* e elevada quota de mercado) é o Wordpress, no entanto este foi excluído dos testes por se tratar de um CMS mais destinado para tarefas de *blogging*.

Os clientes foram simulados com recurso a uma aplicação denominada ApacheBench (AB) [53]. Como balanceador de carga foi utilizado o HAProxy [54] versão 1.4.4. Como servidor de *reverse proxy* foi instalado o Squid [44] versão 2.6. Os CMS correm sobre um servidor HTTP Apache [55] versão 2.0.46. O servidor de Memcache escolhido foi a versão 1.4.5 e a base de dados corre sobre MySQL [56] versão 5.0.77. A descrição destas ferramentas é apresentada na próxima secção.

3.2. Ferramentas Utilizadas

Os clientes foram simulados com recurso ao AB. Esta ferramenta permite simular vários utilizadores a desempenhar a mesma tarefa e recolher métricas a partir da execução dos *workloads* previamente gravados. Usufruindo dessa funcionalidade torna-se fácil colocar o CMS a ser utilizado por um determinado número de utilizadores em simultâneo. Para efeitos de teste, e em cada um dos cenários a testar, o número de utilizadores foi fixado em 5,10, 20, 30 e 50. Os testes consistiam em efectuar 25000 pedidos, tendo sido repetidos várias vezes para assegurar a consistência estatística dos resultados.

O Squid Reverse Proxy é um serviço que permite efectuar *cache* de conteúdo estático. Ele é colocado entre os clientes e os servidores de CMS e à medida que os utilizadores vão efectuando pedidos este servidor actua, verificando se os dados estão presentes na *cache*, se a *cache* possuir os dados devolve-os imediatamente ao cliente caso contrário o pedido é encaminhado para o servidor Web e na resposta o conteúdo estático é colocado em *cache* para futuros pedidos.

O HAProxy é um servidor de balanceamento de carga e foi configurado para aceitar pedidos dos clientes ou do Squid Reverse Proxy e encaminhar esses pedidos para o(s) servidor(es) Web usando o algoritmo *round-robin*. A configuração deste serviço é simples e com ele consegue-se para além da distribuição de carga, aumentar a tolerância a falhas do

site. O HAProxy pode ser complementado com serviços de *hearbeat*, o que lhe permite perceber se alguns dos servidores está em baixo e nesse caso encaminhar os pedidos apenas para os servidores disponíveis.

O servidor de Memcache permite efectuar *cache* de conteúdo dinâmico, nomeadamente resultados de consultas á base de dados. Ele actua entre o servidor Web e o servidor de base de dados. Através de um serviço cliente instalado em cada um dos servidores Web o Memcache *server* é questionado sobre a existência dos dados em *cache*. Se os dados existirem em *cache* são imediatamente devolvidos senão o servidor Web consulta a base de dados colocando de seguida os resultados no Memcache o que se torna útil para futuros pedidos. A configuração do Memcache requer algum *know-how* e algum tempo para configuração. O processo de determinar que *queries* serão alvo de *caching* e a definição do tempo que permanecem em *cache* são dois aspectos fundamentais para se retirar o máximo partido deste serviço.

3.3. Cenários de teste

Para comparar e auferir o ganho de *performance* pela utilização das várias técnicas foram seleccionados módulos comuns entre os CMS Drupal e Joomla. Os módulos escolhidos foram RSS, Poll, Search, Index Page e uma página com conteúdos. A comparação terá por base duas métricas comuns neste tipo de sistemas: latência e *throughput*. A latência representa o tempo final que o utilizador tem que esperar desde que solicita um pedido até que a resposta fica completa. O *throughput* mede a capacidade do servidor processar *N* pedidos por determinada unidade de tempo.

O plano de testes contempla três cenários principais. Para cada cenário há uma subdivisão de testes a realizar:

A. Avaliar a latência e o *throughput* para diferentes tipos de pedidos em cada um dos módulos/CMS, considerando o cenário base;
B. Adicionar ao cenário base cada uma das técnicas e medir desempenho (latência e *throughput*) obtido. Os serviços serão incluídos de forma isolada e comparados com o cenário base;
    1. o *Load Balancer*;
    2. o *Reverse Proxy;*
    3. o servidor de Memcache

C. Usar diferentes combinações das técnicas para comparar o desempenho obtido:
    1. *Load Balancer + Reverse Proxy*
    2. *Load Balancer* + Memcache
    3. *Load Balancer + Reverse Proxy* + Memcache

Cenário A: Avaliar o tempo de latência e *throughput*

Tal como já foi referido a latência e *throughput* são duas métricas muito comuns na avaliação de sistemas. Ao passo que a latência reflecte o que utilizador "experiencia" o *throughput avalia a capacidade do servidor.* Nos testes realizados a avaliação da latência consiste na medição do tempo de resposta sentido pelo utilizador final para cada um dos pedidos efectuados. Por facilidade de interpretação os valores serão agregados por média. Simultaneamente também será medido o *throughput*, representando este o número de pedidos por segundo que o servidor foi capaz de processar.

Neste primeiro cenário, serão simulados 5, 10, 20, 30 e 50 clientes em paralelo através do AB. Esses pedidos chegarão todos ao servidor Web que se encarrega de os processar e devolver a resposta aos clientes. No AB serão recolhidos os valores do teste, e estes serão usados como valores de referência para a comparação entre os diferentes cenários.

Cenário B.1: Incluindo um servidor de *Load Balancer*

Para avaliar o ganho de *performance* que se obtém com a introdução de um balanceador de carga colocou-se o AB a fazer pedidos através de um balanceador de carga. Assim que os pedidos chegam ao balanceador de carga, são distribuídos para os servidores Web. O algoritmo adoptado no balanceador de carga foi o *round-robin.* Com a distribuição dos pedidos por vários servidores de CMS não se espera observar um aumento efectivo do nível de *performance*, a não ser em termos de escalabilidade, isto é, quando o número de pedidos em simultâneo é elevado afectando a capacidade individual dos servidores de CMS.

Cenário B.2: Incluindo um servidor de *Reverse Proxy* (*cache* de conteúdo estático)

Neste cenário os pedidos dos clientes antes de atingir o servidor Web passam por um servidor de *cache* de conteúdo estático. Caso o conteúdo, ou parte do mesmo, esteja na

cache é imediatamente devolvido, caso contrário o pedido segue o ser percurso e na volta o conteúdo estático é armazenado em *cache*.

Com a inclusão desta técnica espera-se que a performance melhore em particular nos casos em que existe um vasto conteúdo estático (ex.: HTML estático, imagens).

Cenário B.3: Incluindo um servidor de Memcache (*cache* de conteúdo dinâmico)

O servidor de Memcache permite fazer *cache* de conteúdo dinâmico. Usufruindo de módulos existentes para os CMS e que suportam a comunicação com os servidores de Memcache, o serviço foi instalado entre os servidores de CMS e a base de dados. Com esta configuração activa as consultas à base de dados passam primeiro pelo servidor de Memcache. Caso este possua o resultado da *query* que está a ser invocada então devolve os dados de imediato. Caso contrário é consultada a base de dados e na volta os resultados são enviados em simultâneo para o cliente e para o servidor de Memcache.

Com esta *cache* espera-se obter um ganho significativo de *performance*, em particular nos casos em que o volume de dados a consultar na base de dados e o número de clientes a requisitar a mesma informação é elevado.

Enquanto que pela avaliação isolada de cada uma das técnicas se espera medir o efeito em termos de *performance* que cada uma introduz, pela combinação das mesmas espera-se ver se o efeito é proporcional, inferior ou superior à utilização isolada. Antes de qualquer teste acredita-se que haverá casos cuja combinação irá trazer uma perda de *performance* enquanto que noutros uma melhoria efectiva. Perceber estes fenómenos é de extremo interesse e potencia o desenvolvimento de algum tipo de mecanismo adaptativo, capaz de utilizar as técnicas existentes mediante a expectativa de melhoria.

Cenário C.1: *Load Balancer* e *Reverse Proxy*

Neste cenário o ambiente de testes conta com um servidor de *cache* de conteúdo estático que aceita os pedidos dos utilizadores. Se o conteúdo estiver em *cache* então é imediatamente devolvido, senão o pedido é encaminhado para um servidor de balanceamento de carga que mediante o algoritmo de escalonamento definido (*round-robin)* redistribui os pedidos pelos servidores Web.

Este cenário à partida favorece situações em que a aplicação dispõe de conteúdo estático e se encontra perante um elevado número de pedidos em simultâneo. Analisar o comportamento de cada um dos módulos é neste âmbito pertinente para perceber as reais vantagens que ser podem obter pela sua adopção.

Cenário C.2: *Load Balancer* e Memcache

O cenário em que o balanceador de carga e o serviço de *cache* de conteúdo dinâmico estão activos em paralelo é também analisado. Neste, os pedidos dos utilizadores são balanceados pelos dois servidores Web e quando é feita uma *query* à base de dados o pedido é em primeiro lugar efectuado ao servidor de Memcache.

Este tipo de infra-estrutura é útil quando existem muitos utilizadores em concorrência e os pedidos estão dependentes de um acesso á base de dados para obter informação de interesse comum aos vários utilizadores.

Cenário C.3: *Load Balancer* com *Reverse Proxy* e Memcache em paralelo

No último cenário são efectuados testes juntando os serviços de *cache* de conteúdo estático, *cache* de conteúdo dinâmico e balanceamento de carga. Este cenário é aquele que pelo junção das várias técnicas se espera trazer um benefício mais óbvio de *performance*, em particular quando existe uma boa quantidade de conteúdo estático, o número de clientes é elevado e partilham dados comuns vindos da base de dados.

Capítulo 4
Resultados experimentais

Neste capítulo apresentam-se os resultados obtidos após a experimentação para cada um dos cenários de teste descritos no capítulo anterior. A análise estabelece uma comparação entre a latência obtida, o *throughput* e o desvio padrão nos tempos de resposta. A latência reflecte o tempo desde que o cliente pede uma página até que a mesma lhe é disponibilizada. O *throughput* mede a quantidade de pedidos processados pelo servidor numa unidade de tempo. O desvio padrão permite medir a oscilação dos tempos de resposta para cada uma das páginas pedidas.

Tratando-se de um ambiente controlado, isto é, os clientes e o servidor partilham a mesma rede, a influência da rede é praticamente nula, tornando-se assim possível medir e estabelecer comparações entre os valores obtidos.

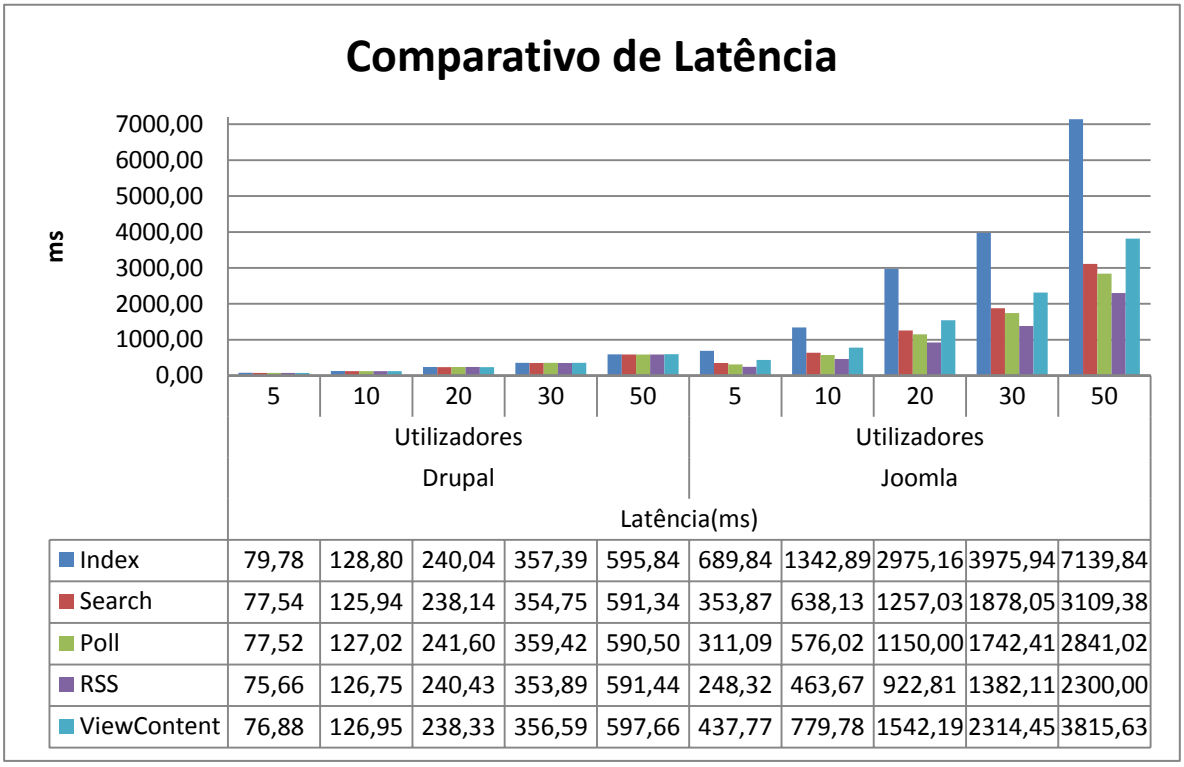
Ao longo das próximas secções vamos apresentar os resultados das análises sobre a latência, *throughput* e desvio padrão obtido nos nossos cenários de teste nomeadamente:

- antes da adopção das técnicas;
- com a adopção de cada uma das técnicas de forma isolada;
- com a combinação das diferentes técnicas.

4.1. *Performance* antes da adopção das técnicas

Nesta secção é efectuada a comparação entre os CMS em estudo antes da adopção das técnicas. Tal é relevante para saber como é que os CMS se comportam antes da introdução de mecanismos adicionais que visam o incremento de *performance.* Os valores obtidos permitem que sejam estabelecidas comparações entre o cenário base e os cenários que contemplam a adopção das diferentes técnicas.

Para auferir a latência “base”, ou seja o tempo que os clientes experienciam entre o momento em que fazem o pedido ao servidor e recebem a resposta utilizou-se a ferramenta AB. Através desta ferramenta simularam-se 5, 10, 20, 30 e 50 clientes que efectuam pedidos em simultâneo ao servidor Web. O valor de latência médio obtido (em milisegundos), para o Drupal e Joomla é ilustrado na figura 4.



Index	79,78	128,80	240,04	357,39	595,84	689,84	1342,89	2975,16	3975,94	7139,84
Search	77,54	125,94	238,14	354,75	591,34	353,87	638,13	1257,03	1878,05	3109,38
Poll	77,52	127,02	241,60	359,42	590,50	311,09	576,02	1150,00	1742,41	2841,02
RSS	75,66	126,75	240,43	353,89	591,44	248,32	463,67	922,81	1382,11	2300,00
ViewContent	76,88	126,95	238,33	356,59	597,66	437,77	779,78	1542,19	2314,45	3815,63

Figura 4 - Comparativo de Latência

Como se pode observar pela figura 4 o CMS Drupal consegue obter tempos de latência inferiores ao Joomla. Uma análise mais detalhada para a diferença observada, sugere que o tempo de latência está associado ao tamanho da página e ao respectivo número de componentes que é carregado em cada um dos CMS. Na base desta conclusão estão os resultados obtidos através da ferramenta Yslow (figura 5), quando se analisou a *index page* de cada um dos CMS em estudo.

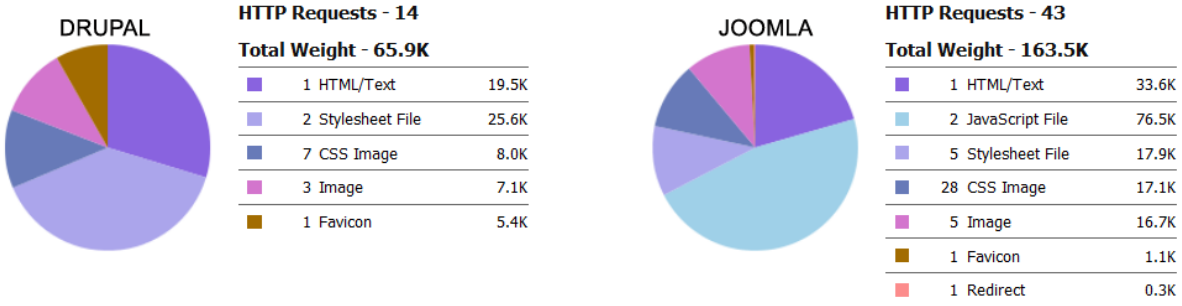
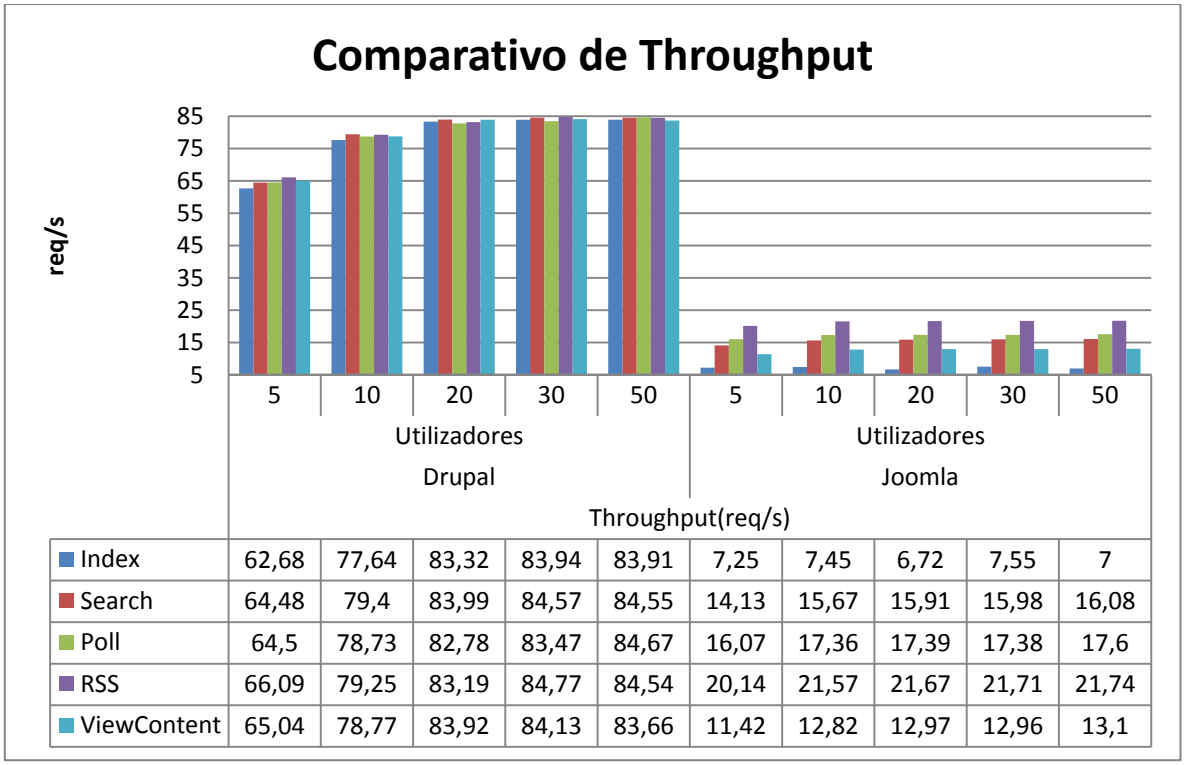


Figura 5 - "Peso" da "Index Page" dos CMS

O facto de a latência ser maior está relacionado com o volume de dados a processar (no servidor e no cliente) e a transferir via rede. Como se pode verificar pela figura 5 no

CMS Joomla são carregados mais componentes, em particular os componentes relacionados com a formatação da página. Estes resultados verificam-se nos restantes módulos testados, com excepção do módulo de RSS, caso em que ambos os CMS têm um número aproximado de componentes.

Do lado do servidor foi medido o *throughput,* isto é, o número de pedidos por segundo processados pelo servidor. O resultado desta análise é apresentado na figura 6.

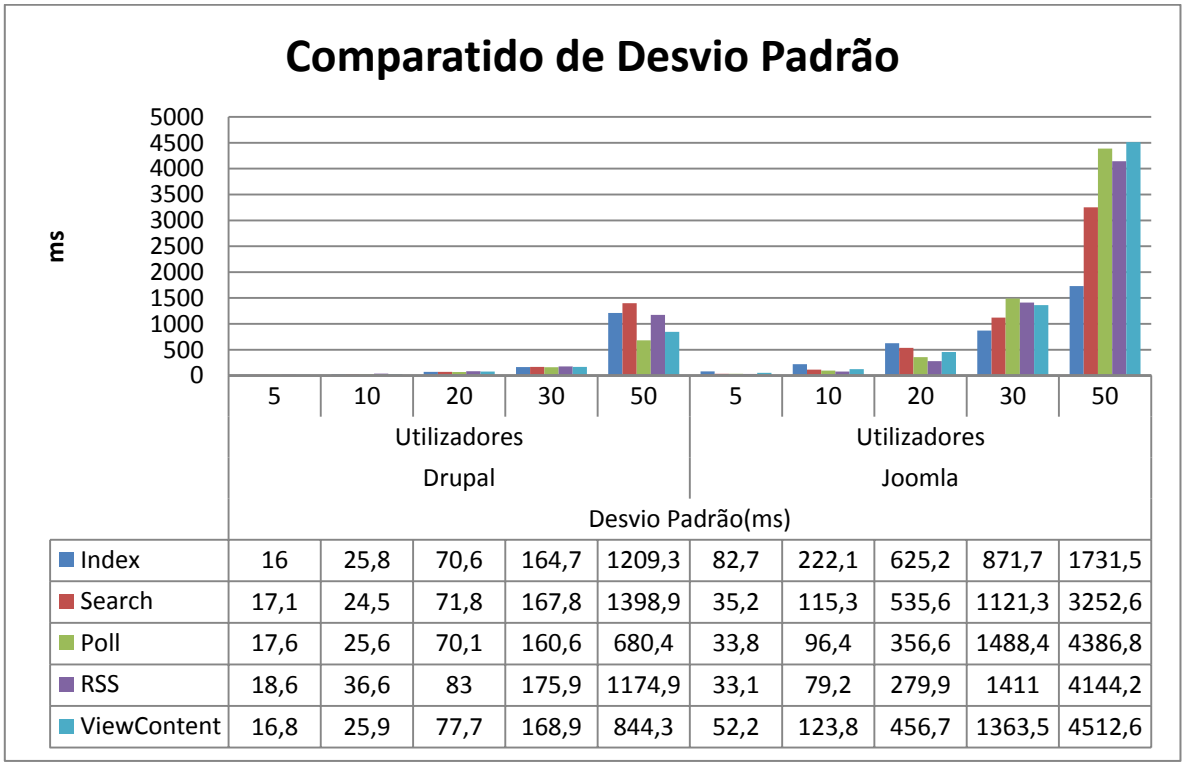


	5	10	20	30	50	5	10	20	30	50
Index	62,68	77,64	83,32	83,94	83,91	7,25	7,45	6,72	7,55	7
Search	64,48	79,4	83,99	84,57	84,55	14,13	15,67	15,91	15,98	16,08
Poll	64,5	78,73	82,78	83,47	84,67	16,07	17,36	17,39	17,38	17,6
RSS	66,09	79,25	83,19	84,77	84,54	20,14	21,57	21,67	21,71	21,74
ViewContent	65,04	78,77	83,92	84,13	83,66	11,42	12,82	12,97	12,96	13,1

Figura 6 - Comparativo de *Throughput*

Os resultados de *throughput* apresentados na figura 6 estão relacionados com os resultados de latência apresentados anteriormente. Á medida que o número de utilizadores em simultâneo aumenta o servidor demora mais a processar os pedidos. Com base nos testes efectuados, e tendo em consideração o ambiente de testes descrito anteriormente, a capacidade máxima do sistema em termos de *throughput* é de cerca de 83 *requests* por segundo para o Drupal e 21 *requests* por segundo no Joomla. A partir do momento em que o servidor atinge a sua capacidade máxima os tempos de latência sobem, isto é, os pedidos dos clientes permanecem mais tempo em fila de espera para serem atendidos. Pode-se ainda dar o caso de o tempo de resposta ultrapassar o valor de *timeout* parametrizado, fazendo com que os utilizadores experienciem erros.

A análise do desvio padrão é útil para auferir a dispersão dos tempos de resposta. Sendo a latência e *throughput* medida em valores médios é importante perceber se cada cliente esta próximo dessa média ou se apenas alguns dos clientes sentem tempos elevados afectando o valor médio. O resultado sobre o desvio padrão para os tempos de resposta é mostrado na figura 7.



	5	10	20	30	50	5	10	20	30	50
Index	16	25,8	70,6	164,7	1209,3	82,7	222,1	625,2	871,7	1731,5
Search	17,1	24,5	71,8	167,8	1398,9	35,2	115,3	535,6	1121,3	3252,6
Poll	17,6	25,6	70,1	160,6	680,4	33,8	96,4	356,6	1488,4	4386,8
RSS	18,6	36,6	83	175,9	1174,9	33,1	79,2	279,9	1411	4144,2
ViewContent	16,8	25,9	77,7	168,9	844,3	52,2	123,8	456,7	1363,5	4512,6

Figura 7 - Comparativo de Desvio Padrão

Pelos valores de desvio padrão apresentados figura 7 verifica-se que quando o número de clientes é baixo não existe muita volatilidade nos tempos. À medida que o número de clientes em simultâneo aumenta e a latência também aumenta, verifica-se que o desvio padrão também cresce. Isto significa que, perante os cenários de maior concorrência há clientes que continuam a ser processados com um tempo de resposta aceitável, mas no entanto há outros para os quais o servidor, por sobrecarga, não é capaz de providenciar a alocação imediata de recursos levando a um atraso no processamento dos pedidos.

4.1. Resultados da *performance* com a adopção das técnicas

Tendo em conta a análise anterior e os métodos escolhidos de optimização, descritos no capítulo anterior, bem como as possíveis combinações entre estes, procedeu-

se a uma análise para a página de entrada de ambos os CMS (índex *Page*) e para o seu módulo de pesquisa (*Search*). Os resultados dessa análise são apresentados ao longo desta secção.

Como forma de comparação foi adoptado o diferencial em percentagem do tempo "base" face aos ganhos/perdas obtidos pela aplicação das diferentes técnicas. Por tempo "base" entende-se os valores obtidos e já apresentados no início deste capítulo. Para tornar os gráficos mais simples de ler foram utilizadas as seguintes abreviaturas para os métodos de optimização:

- LB – *Load Balancer*
- RP – *Reverse Proxy*
- MC – *Memcache*
- MC(SW) - *Memcache* Servidor Web

Estando os métodos como balanceamento de carga (LB), *cache* de conteúdo estático (RP) e dinâmico (MC) referidos e explicados no capítulo anterior, convém relembrar que a localização do servidor de *Memcache* nos testes realizados coincidia com a localização do servidor de base de dados, enquanto que em MC(SW) o servidor de *Memcache* encontra-se localizado na(s) máquina(s) que alberga(m) o(s) servidor(es) Web.

4.1.1. Latência da *Index Page* do Drupal

O ambiente de testes foi devidamente preparado, para auferir a latência da *index page* do Drupal, considerando cada uma das técnicas e a combinação das mesmas. Os clientes foram simulados com recurso à ferramenta AB. Tal como já referido anteriormente, os resultados obtidos pela repetição de cada um dos cenários é sumariada em termos de percentagem de ganho ou perda obtido pela adopção de cada uma das técnicas. Na figura 8 são ilustrados os resultados obtidos.

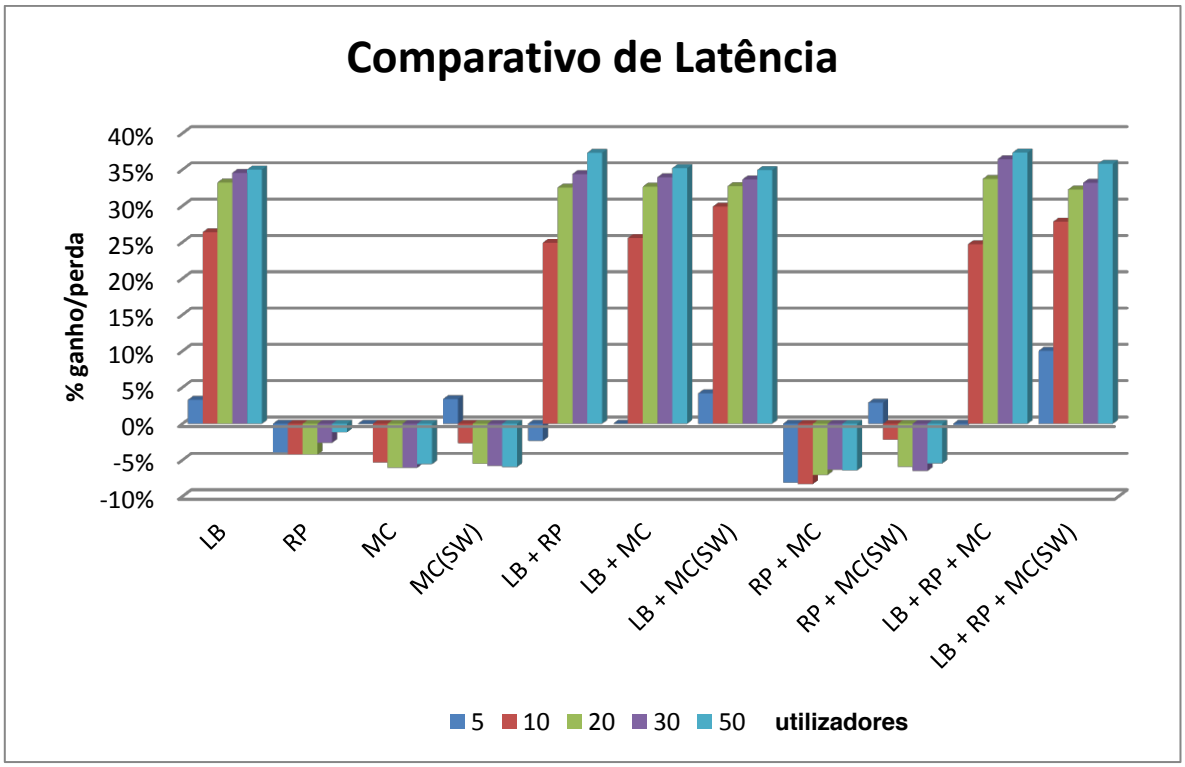


Figura 8 - Comparativo de Latência

Pelos resultados ilustrados na figura 8 verifica-se que o balanceamento de carga é importante para reduzir o tempo de resposta. Tal resultado era esperado, porque o aumento de capacidade do sistema dota-o de maior capacidade e escalabilidade, permitindo assim que um maior número de clientes possa ser atendido em simultâneo sem que a degradação do tempo de resposta seja tão acentuada. Também se verifica que a existência de um serviço de *cache* de conteúdo estático e de conteúdo dinâmico em nada contribuiu para aumentar a performance do sistema. Pela combinação das técnicas verificou-se que os níveis de desempenho se mantiveram ou foram ligeiramente afectados. Tal justifica-se pela

baixa quantidade de conteúdo estático que a página tem e ainda o facto de os dados requeridos à base de dados não serem comuns a uma boa parte dos utilizadores, pelo que a taxa de acerto na *cache* é baixa. Mais à frente neste relatório será apresentada uma análise mais detalhada sobre assunto.

Na figura 9 é ilustrado o resultado em termos de *throughput* obtido para a *index page* do Drupal.

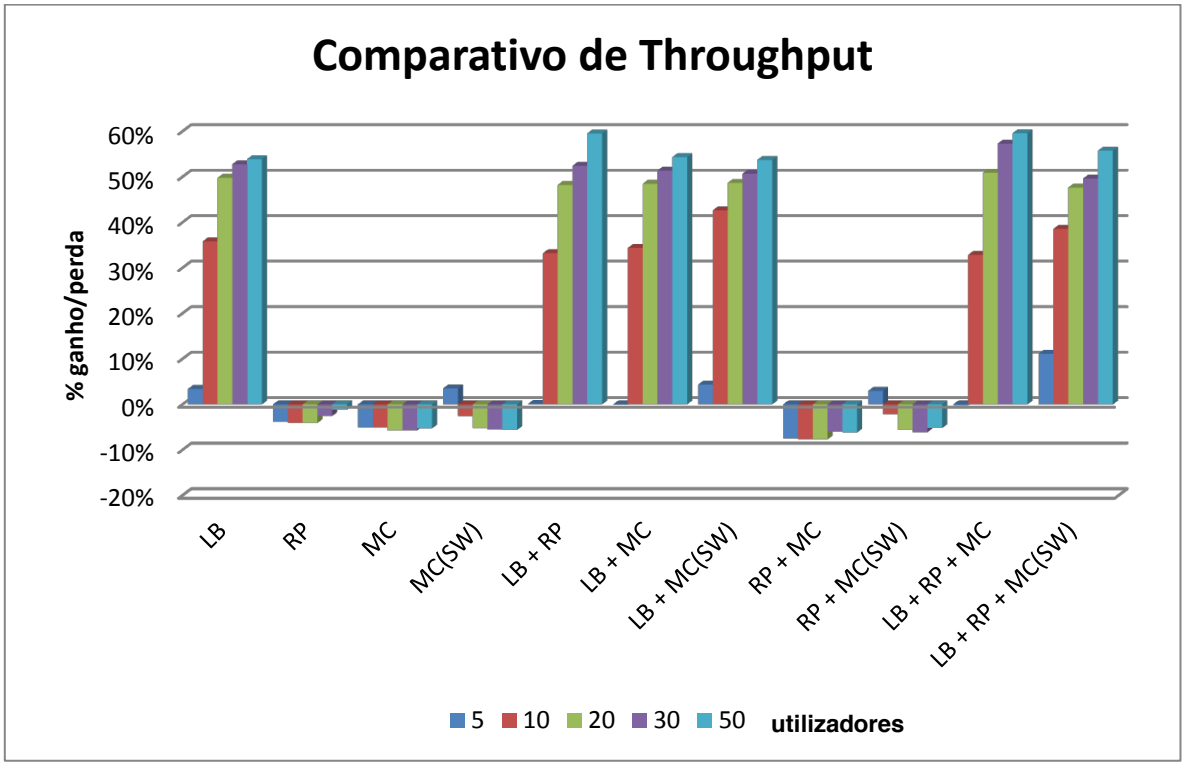


Figura 9 - Comparativo de *Throughput*

Como se pode verificar pela figura 9, o *throughput* sobe quando se introduz o mecanismo de balanceamento de carga. Os ganhos atingem níveis próximos dos 60%. Por outro lado, e como já verificamos na análise sobre a latência os mecanismos de *cache* de conteúdo estático ou *cache* de conteúdo dinâmico, bem como a conjugação de ambas as técnicas prejudica o *throughput* entre os 2% e os 8%. A inclusão destas técnicas quando não existe conteúdo estático ou a taxa de acertos em *cache* de conteúdo dinâmico é baixa faz com que os pedidos demorem ainda mais a ser processados. Esta conclusão é importante, porque realça a importância de conhecer bem a aplicação para tirar partido da adopção das técnicas disponíveis e também realça que não é linear o benefício entre a adopção das técnicas e os resultados esperados.

A análise sobre o desvio padrão é apresentada na figura 10

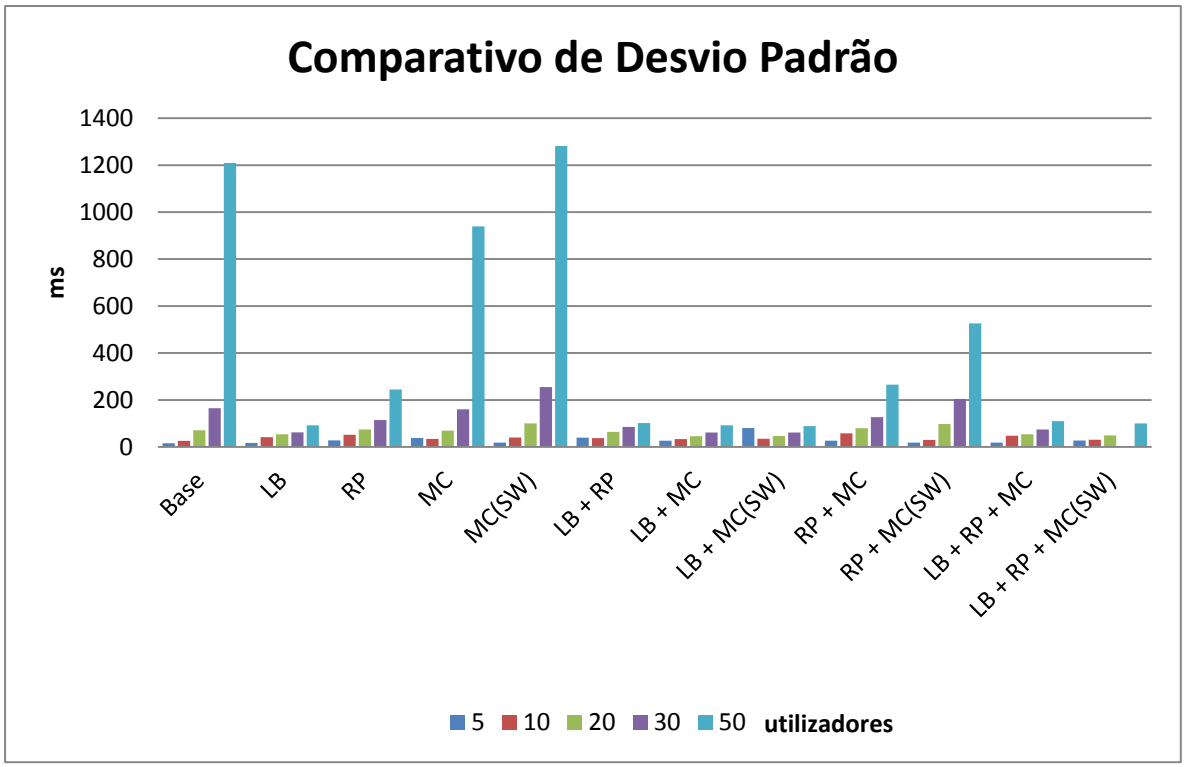


Figura 10 - Comparativo de Desvio Padrão

A figura 10 dá a conhecer a homogeneidade nos tempos de resposta. Como se verifica pela figura o desvio padrão torna-se mais elevado à medida que os níveis de concorrência sobem. Com a introdução do balanceador de carga, passa a ser possível processar mais pedidos por unidade de tempo e o tempo de resposta verificado pelos utilizadores é mais estável. Também se verifica que apesar do tempo de resposta não sair beneficiado pela introdução do *reverse proxy* (*cache* de conteúdo estático), os tempos passaram a ser muito mais estáveis. Isto advém do facto de o *reverse proxy* "amortecer" o processamento dos pedidos na sua fila de espera suavizando assim os tempos de resposta.

4.1.2. Latência da *Search Page* do Drupal

A estrutura da *search page* é diferente da *index page,* na medida em que as interacções com a base de dados são mais intensivas, daí o interesse em auferir qual o resultado obtido pela aplicação das diferentes técnicas. Na figura 11 é apresentado o comparativo de latência entre o cenário "base", a adopção individual e a combinação das diferentes técnicas.

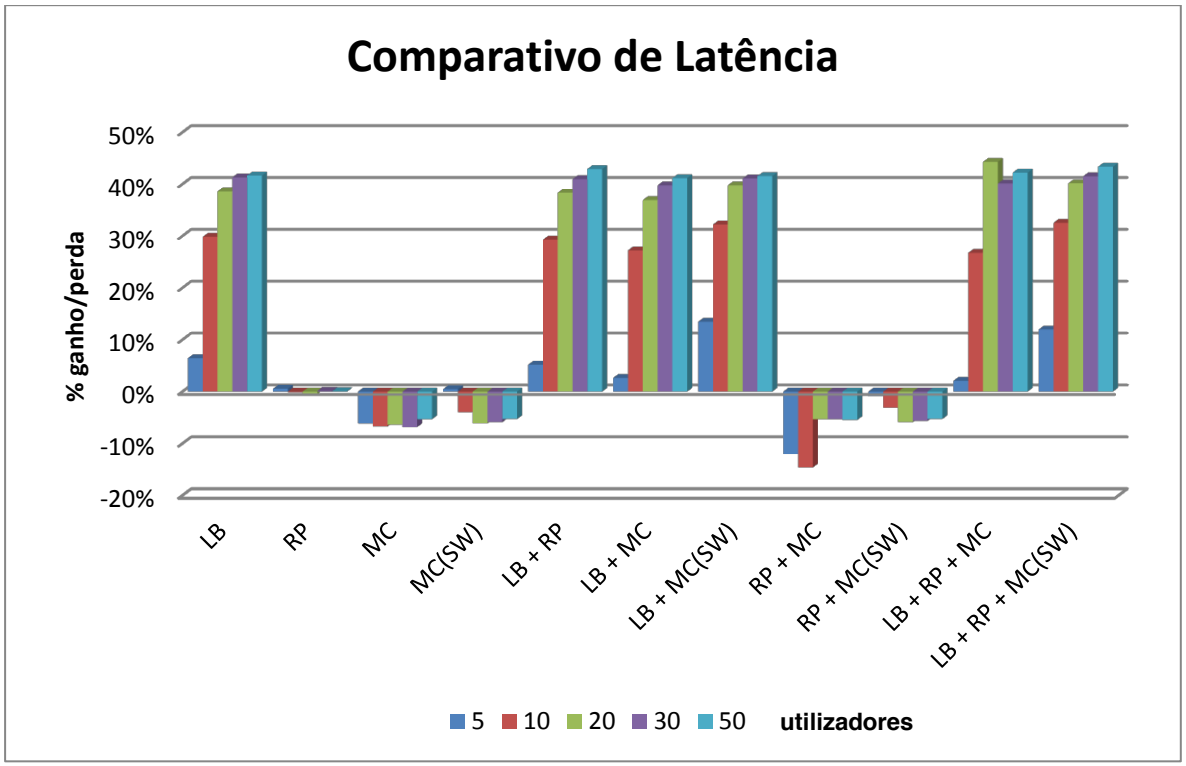


Figura 11 - Comparativo de Latência

Pelos resultados ilustrados na figura 11 verifica-se que o balanceamento de carga é importante para reduzir o tempo de resposta. Tal resultado era esperado, uma vez que a escalabilidade aumenta, permitindo que um maior número de clientes possa ser atendido em simultâneo sem que a degradação de *performance* seja tão acentuada. Também se verifica que a existência de um serviço de *cache* de conteúdo estático e de conteúdo dinâmico prejudicou a performance do sistema. Pela combinação das técnicas verificou-se que os níveis de desempenho se mantiveram ou foram ligeiramente afectados. Tal justifica-se pela baixa quantidade de conteúdo estático que a página tem e ainda o facto de os dados requeridos à base de dados não serem comuns a uma boa parte dos utilizadores, pelo que a

taxa de acerto na *cache* é baixa. Tal como já foi referido, mais à frente neste relatório será apresentada uma análise mais detalhada sobre assunto.

Na figura 12 é ilustrado o resultado em termos de *throughput* obtido para a *search page* do Drupal.

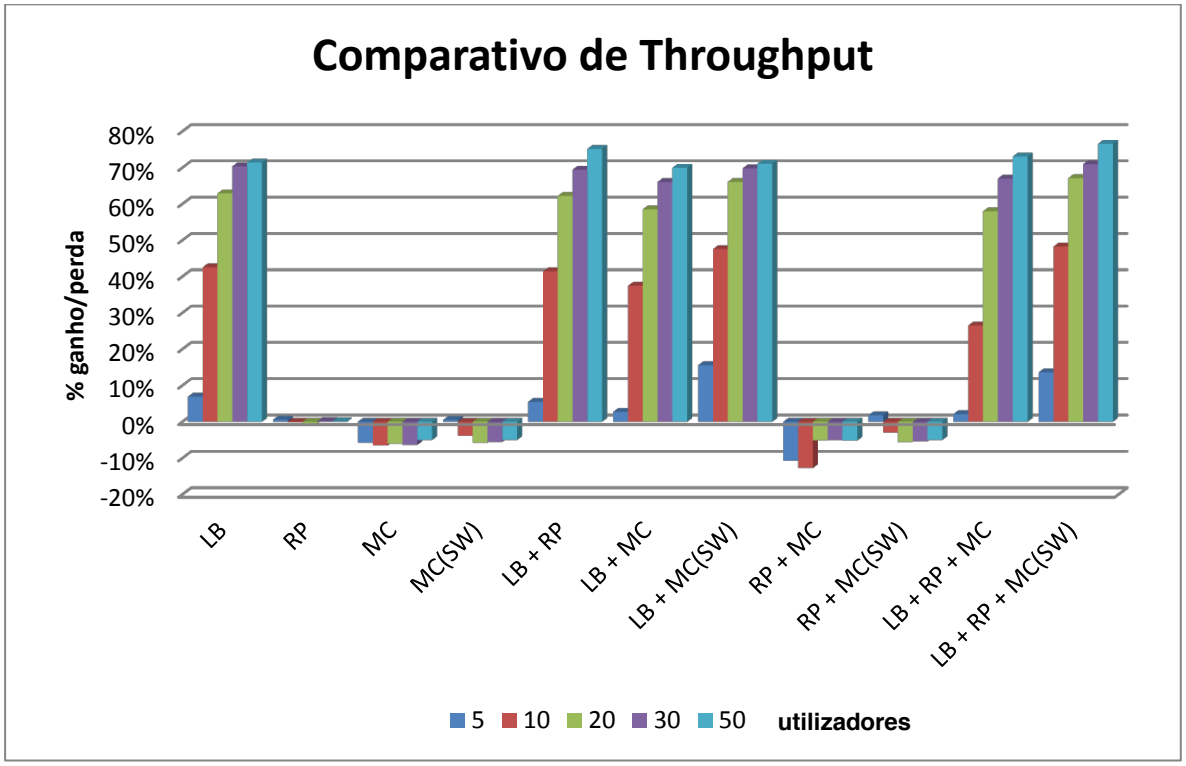


Figura 12 - Comparativo de *Throughput*

Como se pode verificar pela figura 12, o *throughput* sobe quando se introduz o mecanismo de balanceamento de carga. Os ganhos atingem níveis próximos dos 80%. Por outro lado, e como já verificamos na análise sobre a latência os mecanismos de *cache* de conteúdo estático ou *cache* de conteúdo dinâmico, bem como a conjugação de ambas as técnicas prejudica o *throughput* em valores entre os 2% e os 12%. A inclusão destas técnicas quando não existe conteúdo estático ou a taxa de acertos em *cache* de conteúdo dinâmico é baixa faz com que os pedidos demorem ainda mais a ser processados. Esta conclusão é importante, porque realça a importância de conhecer bem a aplicação para tirar partido da adopção das técnicas disponíveis.

Na figura 13 é mostrado a análise sobre o desvio padrão

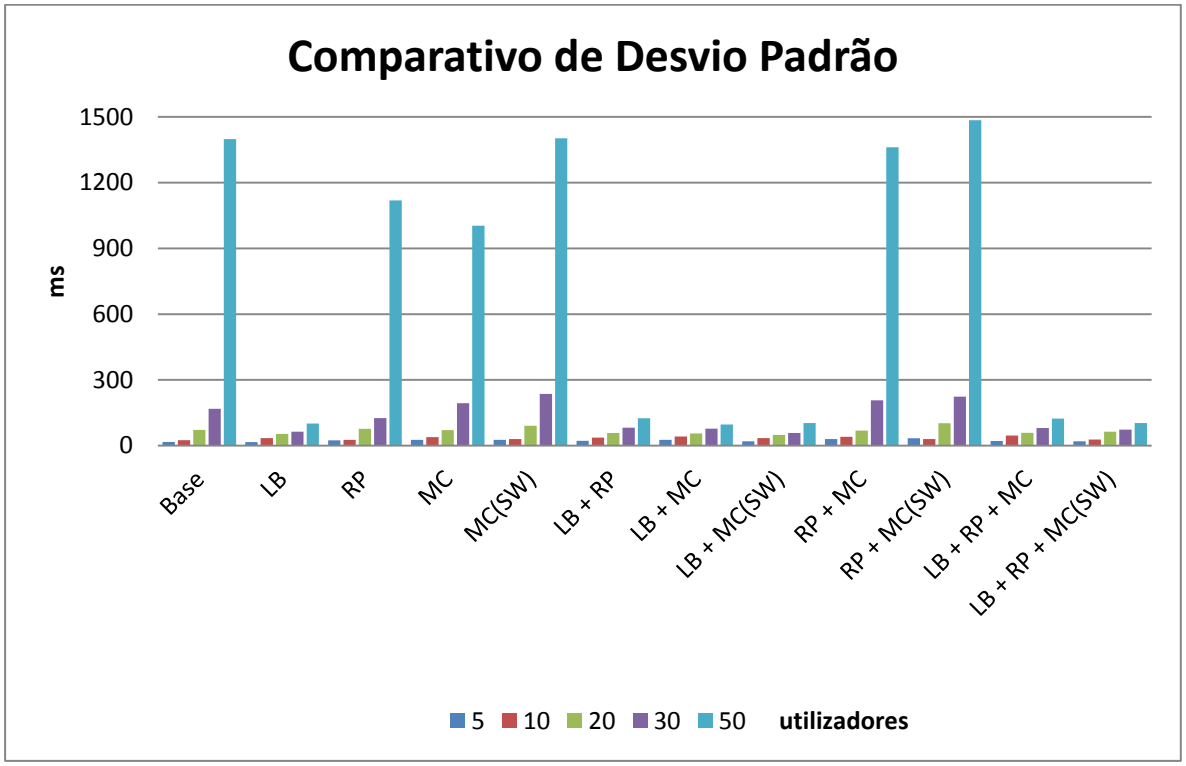


Figura 13 - Comparativo de Desvio Padrão

A figura 13 dá a conhecer a homogeneidade nos tempos de resposta. Como se verifica pela figura o desvio padrão torna-se mais elevado à medida que os níveis de concorrência sobem. Com a introdução do balanceador de carga, passa a ser possível processar mais pedidos por unidade de tempo e o tempo de resposta verificado pelos utilizadores é mais estável. Também se verifica que apesar do tempo de resposta não sair beneficiado pela introdução do *reverse proxy* (*cache* de conteúdo estático), os tempos passaram a ser muito mais estáveis. Isto advém do facto de o *reverse proxy* "amortecer" o processamento dos pedidos na sua fila de espera suavizando assim os tempos de resposta.

4.1.3. Latência da *Index Page* do Joomla

À semelhança do que foi feito com o CMS Drupal, foi verificada a latência, *throughput* e desvio padrão do CMS Joomla. Na figura 14 apresentam-se os resultados obtidos considerando a execução de 5, 10, 20, 30 e 50 utilizadores em simultâneo a efectuar pedidos sobre a *índex page* do Joomla

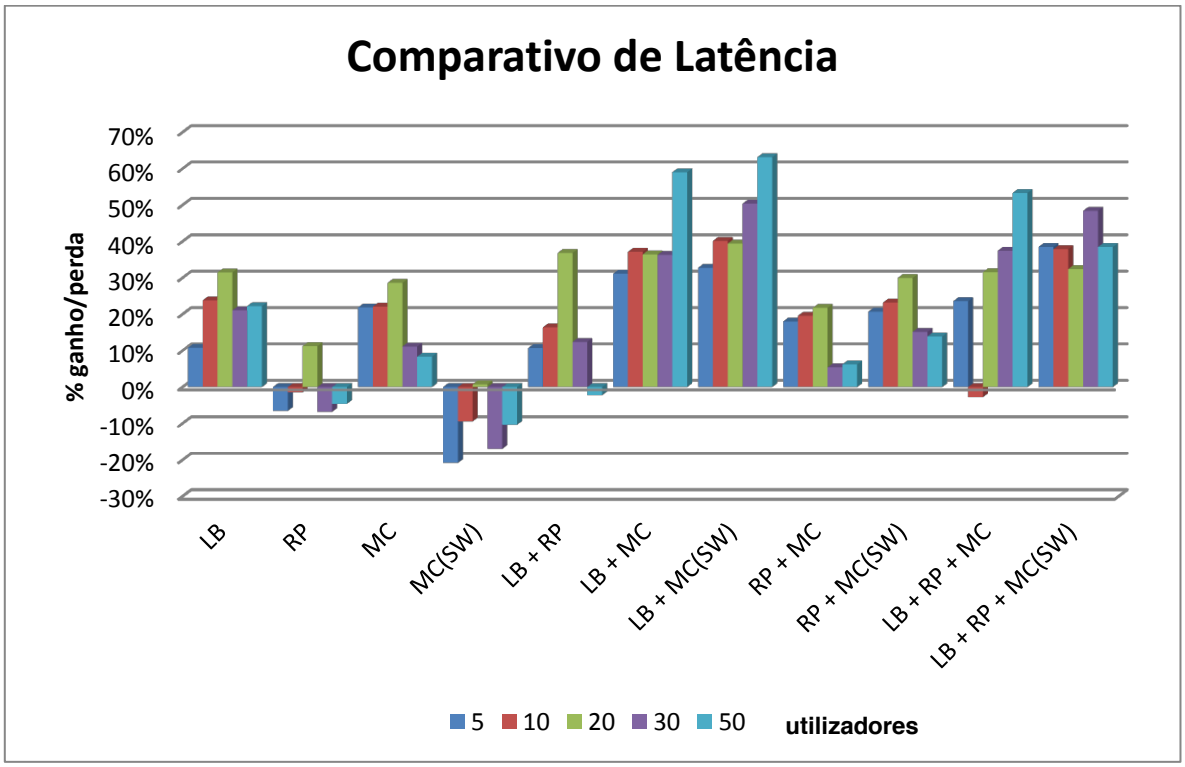


Figura 14 - Comparativo de Latência

Na figura 14 destacam-se as vantagens da adopção das diferentes técnicas. O balanceamento de carga e *cache* de conteúdo dinâmico resultaram em ganhos *performance* de 32% e 29%, respectivamente. A combinação das técnicas revelou também benefícios consideráveis. A combinação entre balanceamento de carga e *cache* de conteúdo de dinâmico reduziu a latência, quando o servidor se encontra a processar 50 clientes em simultâneo, em aproximadamente 60%. Esta é sem dúvida uma enorme vantagem alcançada pela combinação das técnicas. No que diz respeito a adopção de *cache* de conteúdo estático não foram verificadas vantagens significativas. Tal facto advém da baixa quantidade de conteúdo estático.

Na figura 15 é ilustrado o resultado em termos de *throughput* obtido para a *index page* do Joomla.

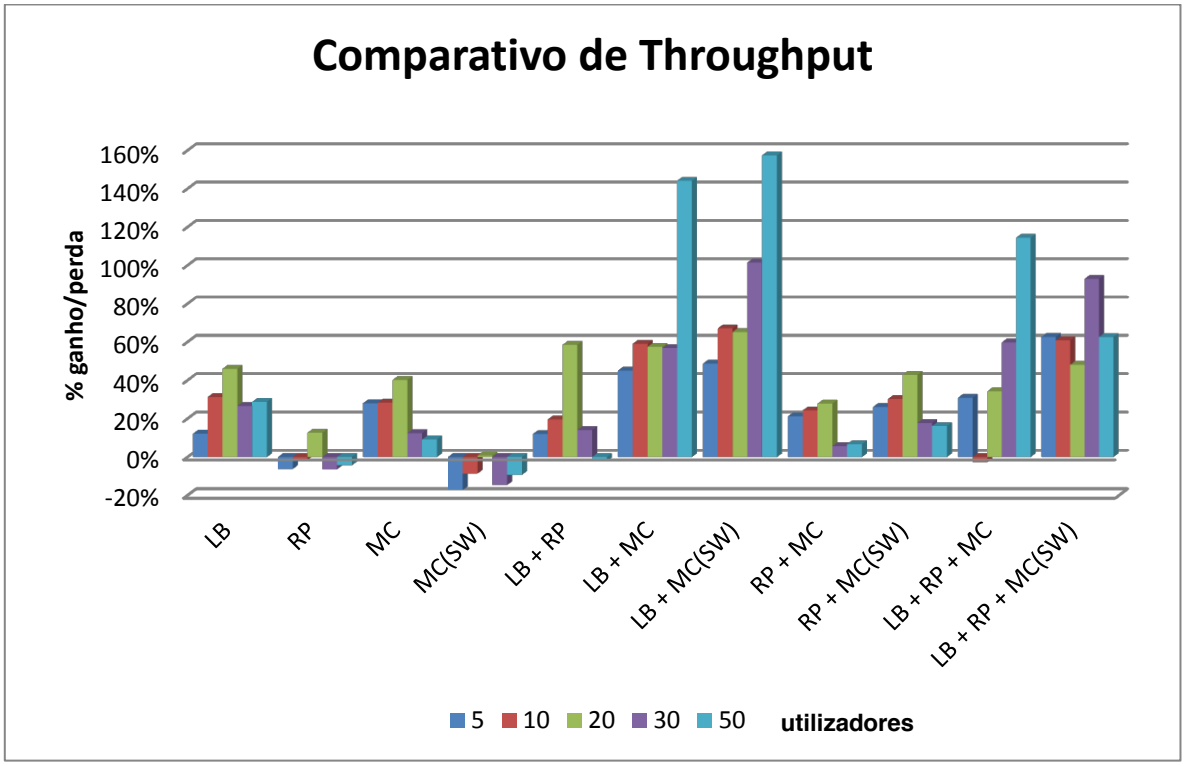


Figura 15 - Comparativo de Throughput

Como expectável, pela figura 15 observa-se que o *throughput* sobe quando se introduz o mecanismo de balanceamento de carga. Neste caso o incremento atinge níveis próximos dos 45%. A combinação entre as técnicas de balanceamento de carga e *cache* de conteúdo dinâmico (quando estas se encontram configuradas no servidor Web) permitiram ganhos na ordem dos 160%, considerando um volume de 50 utilizadores em paralelo.

A análise sobre o desvio padrão nos tempos de resposta verificada pelos clientes consta na figura 16

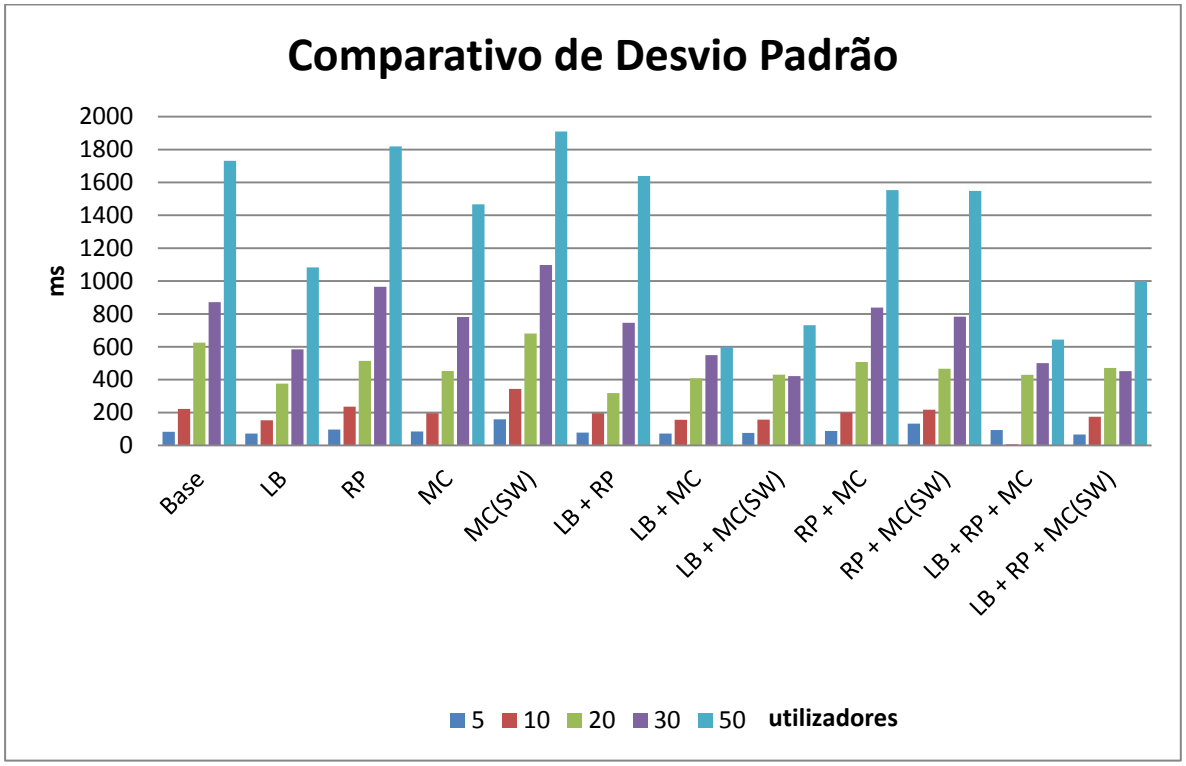


Figura 16 - Comparativo de Desvio Padrão

De acordo com os resultados ilustrados na figura 16, verifica-se que a volatidade dos tempos de resposta é menor quando as técnicas de balanceamento de carga e *cache* de conteúdo dinâmico estão activas. Estes resultados sugerem que os tempos de latência verificados são bastantes próximos uns dos outros, dando assim ênfase às vantagens das técnicas. Não deixa de ser relevante referir que, mesmo para os casos em que são adoptadas estas técnicas, os tempos chegam a ter uma variação de aproximadamente um segundo.

4.1.4. Latência da *Search Page* do Joomla

O propósito da *search page* é claramente diferente da *index page.* Sobre a *search page* espera-se um maior número de interacções com a base de dados. Por isso é importante auferir qual o resultado obtido pela aplicação das diferentes técnicas, em particular as de *cache* de conteúdo de dinâmico. No gráfico 17 é apresentado o comparativo de latência entre o cenário "base" e a aplicação das diferentes técnicas.

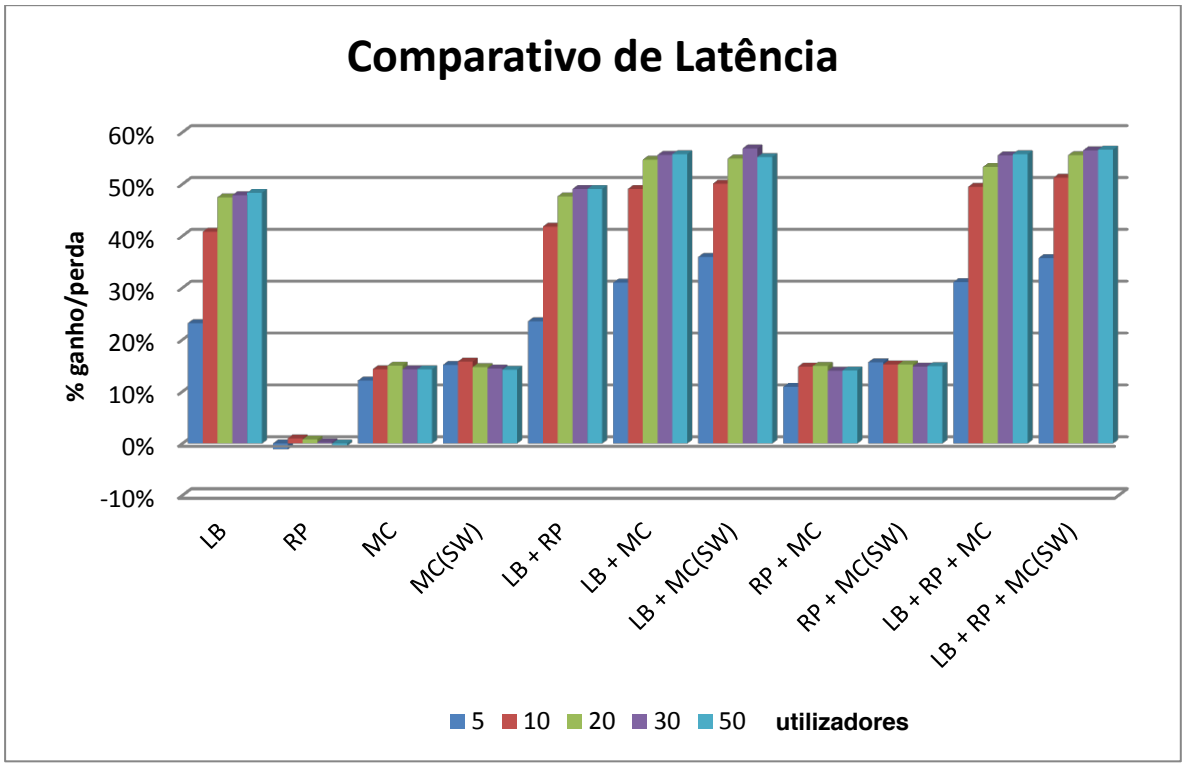


Figura 17 - Comparativo de Latência

Pela figura 17 verifica-se que de uma maneira geral as técnicas surtem o efeito desejado. A excepção é a da *cache* de conteúdo estático na qual os resultados são semelhantes aos resultados base. Com a adopção de mecanismos de balanceamento de carga consegue-se ganhos na ordem dos 50% enquanto que com a utilização da técnica de *cache* de conteúdo dinâmico obtemos ganhos superiores a 14%. Pela combinação das técnicas de balanceamento de carga e *cache* de conteúdo dinâmico obtiveram-se ganhos na ordem dos 55%. Este é sem dúvida um resultado importante e demonstrativo da utilidade da adopção das técnicas. Os resultados evidenciam a necessidade de conhecer a estrutura da aplicação, antes de adoptar as técnicas sob pena de o tempo e investimento neste tipo de técnicas poder revelar-se infrutífero.

Na figura 18 é ilustrado *throughput* obtido para a *search page* do Joomla.

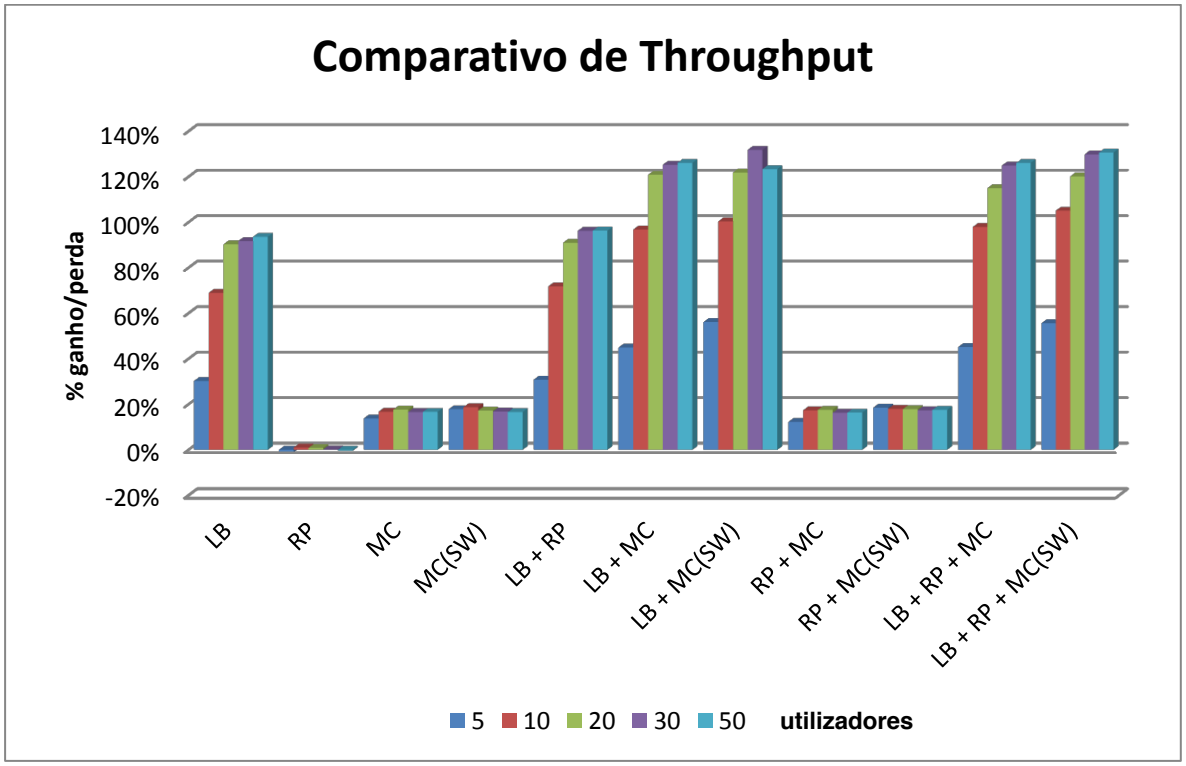


Figura 18 - Comparativo de Throughput

Como mostra a figura 18, e dento do que seria de esperar, os níveis de *throughput* sobem quando se introduz o mecanismo de balanceamento de carga. Neste caso a capacidade do serviço quase duplica. Adicionalmente pela adopção de mecanismos de *cache* de conteúdo dinâmico verificaram-se ganhos na ordem dos 20%. A conjugação de ambas as técnicas elevaram o *throughput* em aproximadamente 130%.

Na figura 19 é mostrado o resultado sobre o desvio padrão dos tempos de resposta. Como já foi dito anteriormente esta análise permite perceber a dispersão dos valores de latência face ao valor médio.

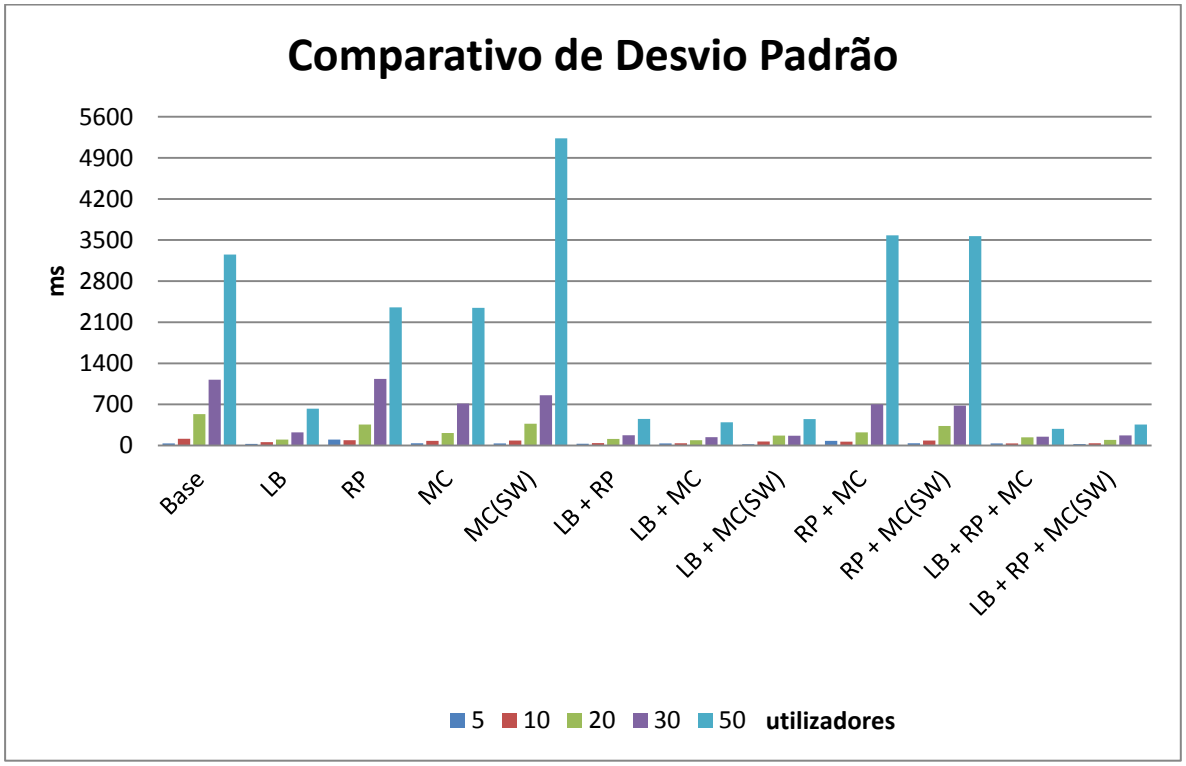


Figura 19 - Comparativo de Desvio Padrão

Como se verifica pela figura 19, a adopção de balanceamento de carga, bem como a combinação entre balanceamento de carga com *cache* de conteúdo estático e *cache* de conteúdo dinâmico faz com que a variação entre os tempos de resposta seja menor. No caso da *cache* de conteúdo dinâmico, partilha o mesmo servidor do CMS (MC(SW)) verifica-se um elevado desvio padrão (aproximadamente 5,2 segundos). Esta situação repete-se em todos os testes realizados permitindo deduzir que não é uma boa prática ter o servidor de cache de conteúdo dinâmico a partilhar o mesmo servidor físico com o CMS.

4.2. Análise de resultados

Pelos resultados obtidos verifica-se que a adopção de balanceamento de carga é vital para aumentar a escalabilidade do serviço, permitindo que mais utilizadores possam ser servidos com tempos de resposta mais baixos. Tal como é ilustrado na figura 20, as técnicas de balanceamento de carga trazem, considerando os cenários em testes, uma vantagem significativa a partir dos 10 clientes em simultâneo, destacando-se claramente os períodos em que os níveis de concorrência aumentam.

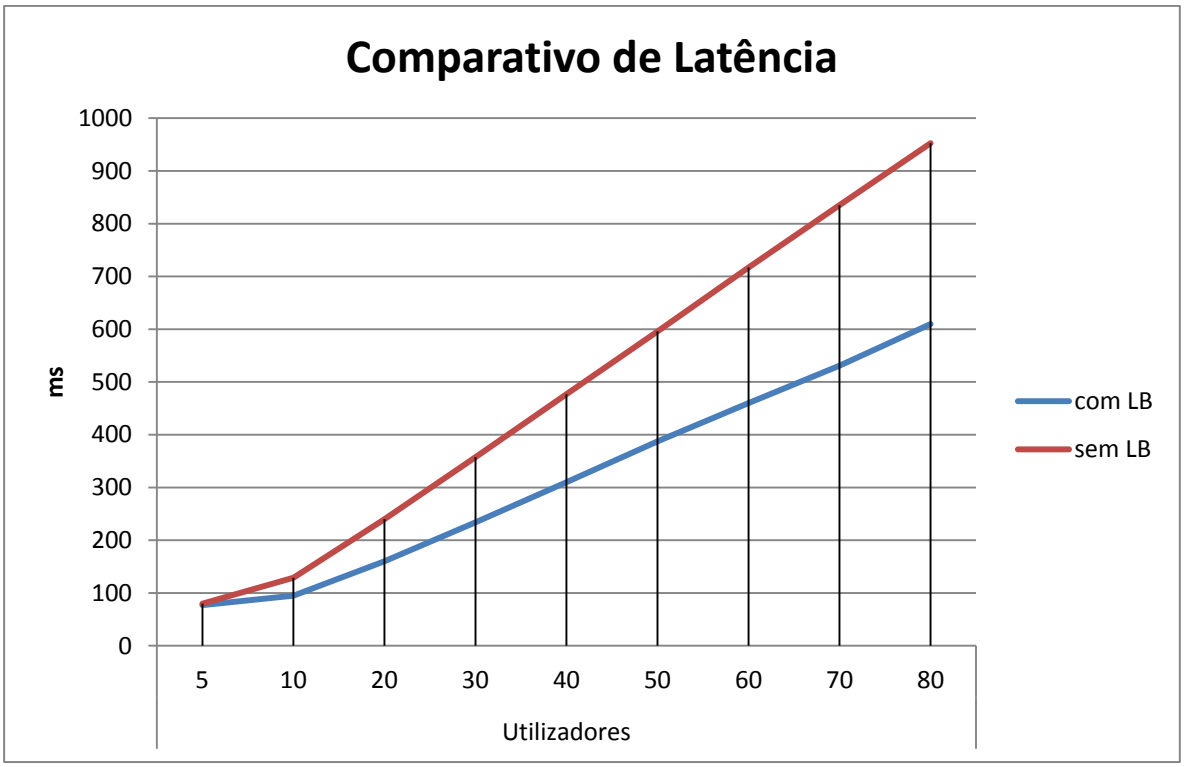


Figura 20 - Comparativo de Latência

Na figura 21 evidenciam-se as vantagens do balanceamento de carga trás sob o ponto de vista dos ganhos em *throughput*.

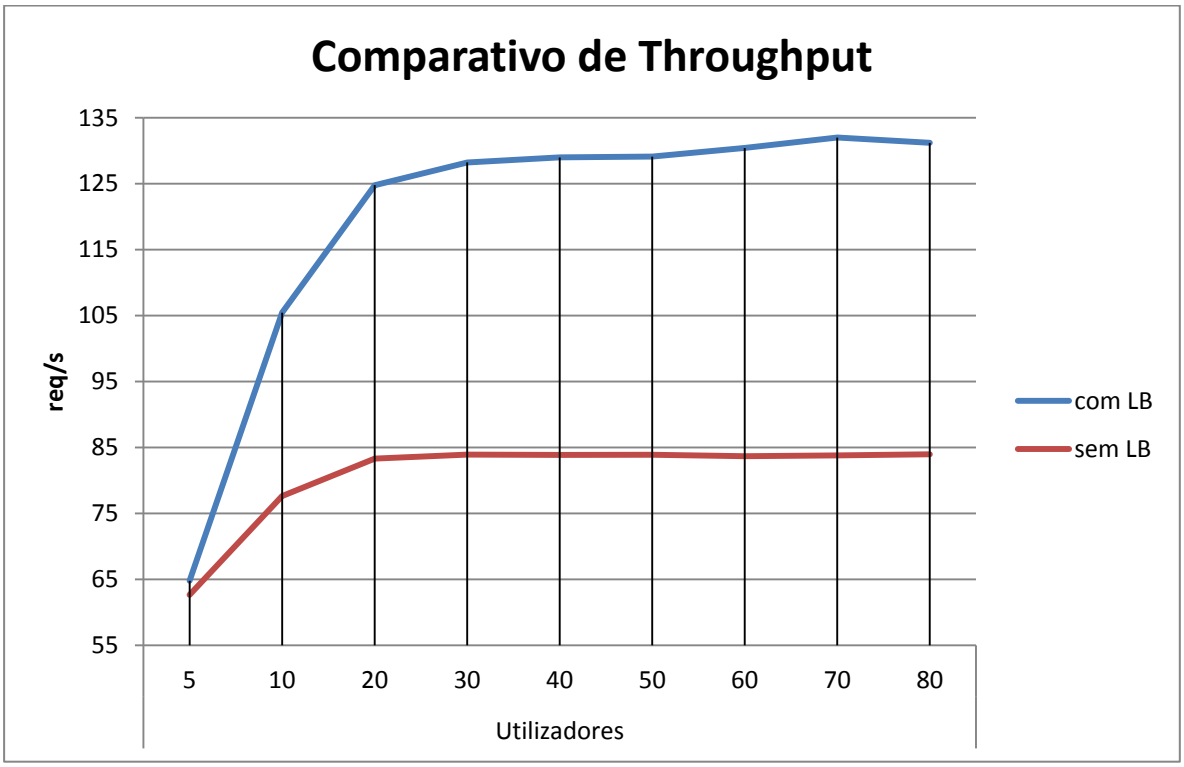


Figura 21 - Comparativo de Throughput

O que é ilustrado nas figuras 20 e 21 aplica-se de forma geral a todos os cenários testados, tendo-se verificado ganhos que variam entre os 4% e os 50% (o gráfico relativo ao desvio padrão encontra-se no Anexo C).

De forma geral a adopção de *cache* para conteúdo estático não revelou ganhos em termos de latência e *throughput.*

A adopção de *cache* de conteúdo dinâmico (Memcache) apenas produziu os efeitos esperados no CMS Joomla. A sua adopção trouxe ganhos que variam entre os 14% e os 30%. No caso do CMS Drupal esta técnica teve um efeito contrário ao esperado, tendo-se registado perdas de performance no tempo de resposta na ordem dos 6%. Para melhor auferir este caso, foi realizado um teste adicional. Este teste consistiu na realização de pesquisas sobre o Drupal (*search page*). A pesquisa é sempre a mesma, no entanto e para melhor perceber os efeitos de escala foi feita uma variação sobre a quantidade de registos devolvidos pela mesma (250, 500, 1000 e 15000). Cada pesquisa foi efectuada 25000

vezes. Na figura 22 apresentam-se os resultados sobre a latência verificada na pesquisa envolvendo um servidor de *cache* de conteúdo dinâmico.

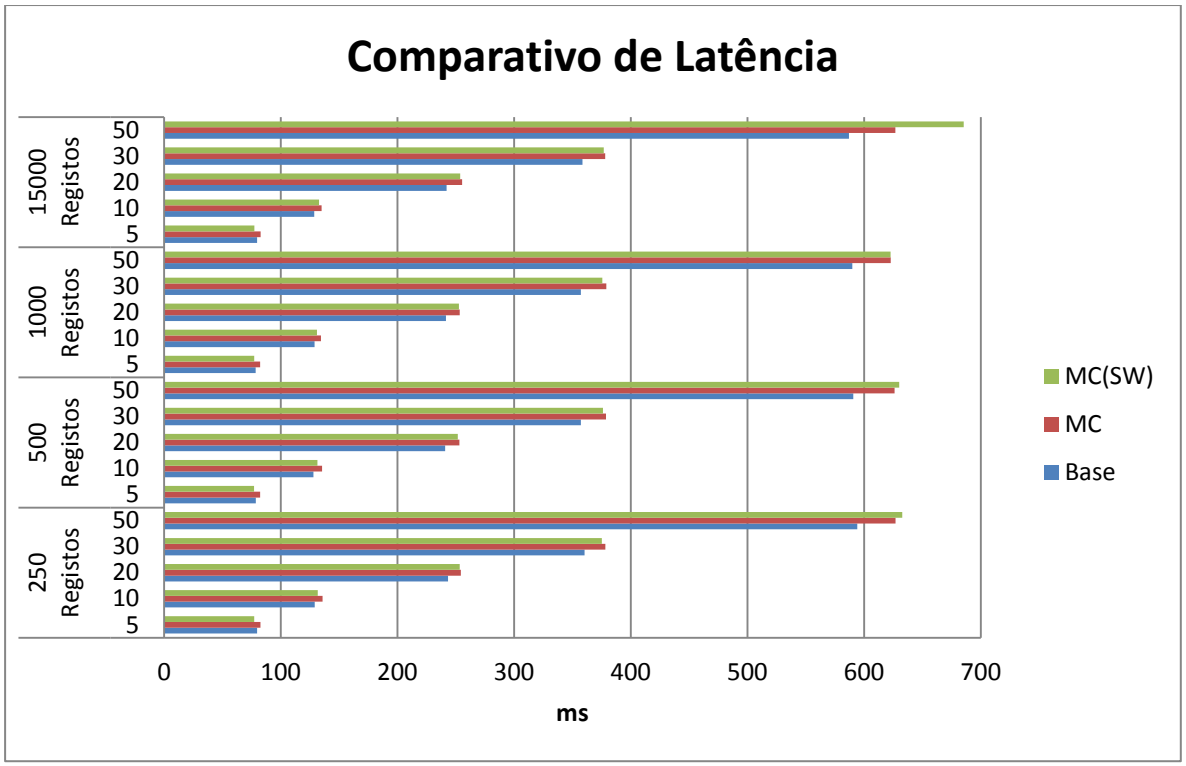


Figura 22 - Comparativo de Latência

Como se pode observar pela figura 22 a utilização do Memcache não se destaca do cenário base, no qual as consultas são realizadas directamente sobre a base de dados. A adopção entre *cache* de conteúdo dinâmico colocada ao nível do servidor de CMS ou num servidor dedicado também não revela diferenças significativas. Destes resultados concluiu-se que a *cache* de conteúdo dinâmico não se traduz, ao contrário do que seria de esperar numa vantagem para o módulo de *search* do Drupal. Os respectivos gráficos contendo o *throughput* e o desvio padrão são apresentados no Anexo C.

Face à situação verificada, analisou-se o comportamento do mecanismo de *cache* tendo em consideração o *input* e *output* resultante das *queries* que estavam a ser submetidas, estando os *logs* fornecidos pelo módulo de Memcache do Drupal presentes no Anexo C. Começou-se por verificar que a cache estava completamente vazia pelo que a taxa de *hit rate* era de 0%. Após se verificar que os dados na *cache* de conteúdo dinâmico eram reutilizados novamente, procedeu-se então á averiguação do motivo de não haver grandes melhorias de *performance,* estando isto relacionado com a forma como os resultados da pesquisa são retornados e como este mecanismo funciona no Drupal [57,58].

Capítulo 5
Conclusões

Ao longo deste trabalho foram aplicadas diferentes técnicas ao nível da infra-estrutura de suporte das aplicações do tipo CMS e foi feita uma avaliação sobre ganho/perda de *performance* obtido pela aplicação das mesmas. As técnicas consistiram na adopção de balanceamento de carga (HAProxy), *cache* de conteúdo estático (Squid Reverse Proxy) e Memcache para *cache* de conteúdo dinâmico (resultados de consultas á base de dados). Estas técnicas foram avaliadas de forma isolada e de forma integrada utilizando um *workload* que simula utilizadores a fazer pedidos ao servidor Web, considerando níveis de concorrência diferentes.

De acordo com os testes realizados verificou-se que a técnica de *cache* de conteúdo estático não trouxe benefícios para ambas as aplicações testadas. Uma das conclusões que se pode tirar de ambas as aplicações é que dado a baixa quantidade de conteúdos estáticos, os sistemas de *cache* não se traduzem numa vantagem. No que se refere a *cache* de conteúdo dinâmico é importante separar os resultados obtidos pelo Drupal e pelo Joomla. Enquanto que no Drupal, pela forma como as *queries* são processadas e os dados devolvidos não há vantagem na adopção de *cache* de conteúdo dinâmico, no Joomla a sua adopção traduziu-se em ganhos de performance, particularmente quando o número de utilizadores a pesquisar pelo mesmo conteúdo é elevado.

Uma vantagem transversal às aplicações adoptadas é a adopção de mecanismos de balanceamento de carga. Estes mecanismos tornam as aplicações mais escaláveis, pelo que um maior número de utilizadores beneficia de tempos de resposta mais estáveis.

No que diz respeito á complexidade e tempo de configuração de cada uma destas técnicas, importa dizer que para além do investimento necessário em *hardware* extra a configuração é simples. O tempo de *setup* das técnicas não é elevado:

- *Setup* do *Load Balancer* - ½ dia de técnico sénior;
- *Setup* do *Reverse Proxy* - ½ dia de técnico sénior;
- *Setup* do Memcache para CMS adoptando os módulos existente - ½ dia. Caso seja necessário programar o cliente de Memcache estima-se que seja necessário de 3 dias de um técnico sénior.

Uma conclusão que se retira dos testes efectuados e que merece toda a nossa atenção no trabalho futuro é a de que “uma medida não serve a todos”, ou seja dependendo do tipo de pedido, do número de pedidos simultâneos e do tipo de dados que são retornados (estático *versus* dinâmico) pode fazer todo o sentido criar um sistema adaptativo capaz de autoseleccionar os serviços (balanceamento de carga, *cache* estática e *cache* dinâmica) envolvidos no processamento do pedido. Por exemplo caso o serviço verifique que as consultas a um servidor de *cache* de conteúdo dinâmico não estão a surtir o efeito desejado para um determinado tipo de *queries*, então para essas queries o servidor de *cache* de conteúdo dinâmico deve ser evitado, sendo o pedido processado exclusivamente pela base de dados. Desta forma não só se poupa tempo com a pesquisa a mais serviços, como também se libertam recursos para os casos em que as *queries* realmente beneficiam da *cache* de conteúdo dinâmico.

Bibliografia

[1] S. Power, “Metrics 101: What to Measure on Your Website“, http://en.oreilly.com/velocity2010/public/schedule/detail/14327#

[2] G.Linden. “Make data useful”. http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazoncom

[3] S. Souders, “High Performance Web Sites”. O’Reilly, ISBN 978-0-596-52930-9, 2007

[4] S. Souders, “Even Faster Web Sites”. O’Reilly, ISBN 978-0-596-52230-8, 2009

[5] COTS - http://en.wikipedia.org/wiki/Commercial_off-the-shelf

[6] J. Adms. “Fixing Twitter: Improving the Performance and Scalability of the World’s Most Popular Micro-blogging site”, http://assets.en.oreilly.com/1/event/29/Fixing_Twitter_Improving_the_Performance_and_Scalability_of_the_World_s_Most_Popular_Micro-blogging_Site_Presentation%20Presentation.pdf

[7] YouTube Scalability - http://video.google.com/videoplay?docid=-6304964351441328559#

[8] GTalk: lessons in building scalable systems - http://video.google.com/videoplay?docid=6202268628085731280#

[9] Water & Stone Report, http://www.waterandstone.com/downloads/2008OpenSourceCMSMarketSurvey.pdf

[10] Drupal Bug Tracker, http://drupal.org/project/issues

[11] Joomla Bug Tracker, http://joomlacode.org/gf/project/joomla/tracker/?action=TrackerItemBrowse&tracker_id=32

[12] YSlow, http://developer.yahoo.com/yslow/

[13] Google Page Speed, http://code.google.com/intl/pt-PT/speed/page-speed/

[14] http://developer.yahoo.com/performance/rules.html

[15] ImageMagick, http://www.imagemagick.org/

[16] PNGCrush, http://pmt.sourceforge.net/pngcrush/

[17] Smush.it, http://www.smushit.com/ysmush.it/

[18] http://www.favbrowser.com/internet-explorer-9-ie9-vs-firefox-4-vs-google-chrome-10-vs-opera-11-vs-safari-5/

[19] http://www.ghacks.net/2011/03/17/web-browser-benchmark-results-comparison/

[20] http://sixrevisions.com/infographs/browser-performance/

[21] http://html5test.com/

[22] http://acid3.acidtests.org/

[23] http://www.webkit.org/perf/sunspider/sunspider.html

[24] http://ie.microsoft.com/testdrive/Performance/FishIETank/Default.html

[25] http://v8.googlecode.com/svn/data/benchmarks/v6/run.html

[26] http://clients.futuremark.com/peacekeeper/index.action

[27] Firebug, http://getfirebug.com/

[28] Firefox, http://www.mozilla-europe.org/pt/

[29] https://addons.mozilla.org/en-US/firefox/performance/ (consultado em 04/05/2011)

[30] http://www.performancewiki.com/linux-tuning.html

[31] http://people.redhat.com/alikins/system_tuning.html

[32] http://www.howtoforge.com/configuring_apache_for_maximum_performance

[33] http://onlamp.com/pub/a/onlamp/2004/02/05/lamp_tuning.html

[34] http://www.serverwatch.com/tutorials/article.php/10825_3436911_1/Apache-Server-Performance-Optimization.htm

[35] http://beginlinux.com/blog/2010/07/apache-performance-tuning/

[36] http://drupalmodules.com/category/Performance+and+Scalability

[37] http://extensions.joomla.org/extensions/site-management/site-performance

[38] http://extensions.joomla.org/extensions/site-management/content-networking

[39] http://www.linuxweblog.com/tune-my.cnf

[40] http://www.mysqlperformanceblog.com/2006/09/29/what-to-tune-in-mysql-server-after-installation/

[41] http://www.howtoforge.com/tuning-mysql-performance-with-mysqltuner

[42] Memcached, http://memcached.org/

[43] EHCache, http://ehcache.org/

[44] Squid, http://www.squid-cache.org/

[45] NginX, http://www.nginx.org/

[46] Varnish, http://www.varnish-cache.org/

[47] Roundrobin, http://en.wikipedia.org/wiki/Round-robin#Computing

[48] http://en.wikipedia.org/wiki/Computer_cluster

[49] http://pt.scribd.com/doc/20525007/How-Drupal-Works

[50] http://cruncht.com/95/drupal-caching/

[51] http://books.tag1consulting.com/book/export/html/5

[52] http://www.theartofjoomla.com/joomla-caching-explained.html

[53] ApacheBench, http://httpd.apache.org/docs/2.0/programs/ab.html

[54] HAProxy, http://haproxy.1wt.eu/

[55] Apache, http://www.apache.org/

[56] MySQL, http://www.mysql.com/

[57] http://acquia.com/blog/drupal-search-how-indexing-works

[58] http://drupal.org/documentation/modules/search

[59] Paul Roy e Alex Polak, “Performance Measurement and Case Studies at MSN”, http://ebookbrowse.com/performance-measurement-and-case-studies-at-msn-presentation-pptx-d145271668

[60] Stoyan Stefanov, “YSlow 2.0”,http://www.slideshare.net/stoyan/yslow-20-presentation

[61] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, “Why do internet services fail, and what can be done about it?” in Proceedings of the 4th conference on USENIX Symposium on

Internet Technologies and Systems - Volume 4, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003

[62] http://dev.mysql.com/doc/refman/5.0/en/storage-engines.html

[63] http://dev.mysql.com/doc/refman/5.1/en/storage-engines.html

[64] http://mikebernat.com/blog/MySQL_-_InnoDB_vs_MyISAM

[65] Joomla - http://joomla.org/

[66] Drupal - http://drupal.org/

Anexo A

A.1. Quadro comparativo entre Drupal e Joomla

System Requirements	Drupal 6.10	Joomla! 1.5.10
Application Server	Apache	CGI
Approximate Cost	Free	Free
Database	MySQL	MySQL
License	Open Source	Open Source
Operating System	Platform Independent	Platform Independent
Programming Language	PHP	PHP
Root Access	No	No
Shell Access	No	No
Web Server	Apache	Apache
Security	Drupal 6.10	Joomla! 1.5.10
Audit Trail	Yes	No
Captcha	Free Add On	Free Add On
Content Approval	Yes	Yes
Email Verification	Yes	Yes
Granular Privileges	Yes	No
Kerberos Authentication	No	No
LDAP Authentication	Free Add On	Yes
Login History	Yes	Yes
NIS Authentication	No	No
NTLM Authentication	Free Add On	No
Pluggable Authentication	Yes	Yes
Problem Notification	No	No
Sandbox	No	No
Session Management	Yes	Yes
SMB Authentication	No	No
SSL Compatible	Yes	Yes
SSL Logins	No	Yes
SSL Pages	No	Yes
Versioning	Yes	Free Add On
Support	Drupal 6.10	Joomla! 1.5.10
Certification Program	No	No
Code Skeletons	Yes	No
Commercial Manuals	Yes	Yes
Commercial Support	Yes	Yes
Commercial Training	Yes	Yes

Developer Community	Yes	Yes
Online Help	Yes	Yes
Pluggable API	Yes	Yes
Professional Hosting	Yes	Yes
Professional Services	Yes	Yes
Public Forum	Yes	Yes
Public Mailing List	Yes	No
Test Framework	Free Add On	Yes
Third-Party Developers	Yes	Yes
Users Conference	Yes	Yes
Ease of Use	Drupal 6.10	Joomla! 1.5.10
Drag-N-Drop Content	Free Add On	No
Email To Discussion	Free Add On	Free Add On
Friendly URLs	Yes	Yes
Image Resizing	Free Add On	Yes
Macro Language	Free Add On	Yes
Mass Upload	Free Add On	Yes
Prototyping	Limited	Yes
Server Page Language	Yes	Yes
Site Setup Wizard	Limited	No
Spell Checker	Free Add On	No
Style Wizard	Limited	No
Subscriptions	Free Add On	Costs Extra
Template Language	Limited	Yes
UI Levels	No	Yes
Undo	Limited	No
WYSIWYG Editor	Free Add On	Yes
Zip Archives	No	No
Performance	Drupal 6.10	Joomla! 1.5.10
Advanced Caching	Yes	Yes
Database Replication	Limited	No
Load Balancing	Yes	Yes
Page Caching	Yes	Yes
Static Content Export	No	No
Management	Drupal 6.10	Joomla! 1.5.10
Advertising Management	Free Add On	Yes
Asset Management	Yes	Yes
Clipboard	No	No
Content Scheduling	Free Add On	Yes
Content Staging	Free Add On	No
Inline Administration	Yes	Yes
Online Administration	Yes	Yes
Package Deployment	No	No
Sub-sites / Roots	Yes	Yes

Themes / Skins	Yes	Yes
Trash	No	Yes
Web Statistics	Yes	Yes
Web-based Style/Template Management	Yes	Yes
Web-based Translation Management	Yes	Free Add On
Workflow Engine	Limited	No
Interoperability	Drupal 6.10	Joomla! 1.5.10
Content Syndication (RSS)	Yes	Yes
FTP Support	Limited	Yes
iCal	Free Add On	No
UTF-8 Support	Yes	Yes
WAI Compliant	Limited	No
WebDAV Support	No	No
XHTML Compliant	Yes	No
Flexibility	Drupal 6.10	Joomla! 1.5.10
CGI-mode Support	Yes	Yes
Content Reuse	Limited	Yes
Extensible User Profiles	Yes	Yes
Interface Localization	Yes	Yes
Metadata	Yes	Yes
Multi-lingual Content	Yes	Free Add On
Multi-lingual Content Integration	Free Add On	Free Add On
Multi-Site Deployment	Yes	Free Add On
URL Rewriting	Yes	Yes
Built-in Applications	Drupal 6.10	Joomla! 1.5.10
Blog	Yes	Yes
Chat	Free Add On	Free Add On
Classifieds	Free Add On	Free Add On
Contact Management	Free Add On	Yes
Data Entry	Free Add On	Free Add On
Database Reports	No	Free Add On
Discussion / Forum	Yes	Free Add On
Document Management	Limited	Free Add On
Events Calendar	Free Add On	Free Add On
Events Management	Free Add On	Free Add On
Expense Reports	No	Free Add On
FAQ Management	Yes	Yes
File Distribution	Free Add On	Free Add On
Graphs and Charts	No	Free Add On
Groupware	Free Add On	Free Add On
Guest Book	Free Add On	Free Add On
Help Desk / Bug Reporting	Free Add On	Free Add On
HTTP Proxy	No	No
In/Out Board	No	No

Job Postings	Free Add On	Free Add On
Link Management	Free Add On	Yes
Mail Form	Free Add On	Yes
Matrix	No	No
My Page / Dashboard	Free Add On	No
Newsletter	Free Add On	Free Add On
Photo Gallery	Free Add On	Free Add On
Polls	Yes	Yes
Product Management	Free Add On	Yes
Project Tracking	Free Add On	Free Add On
Search Engine	Yes	Yes
Site Map	Free Add On	Free Add On
Stock Quotes	Free Add On	Free Add On
Surveys	Free Add On	Free Add On
Syndicated Content (RSS)	Yes	Yes
Tests / Quizzes	Free Add On	Free Add On
Time Tracking	Free Add On	No
User Contributions	Yes	Yes
Weather	Free Add On	Free Add On
Web Services Front End	Limited	Yes
Wiki	Free Add On	Free Add On
Commerce	Drupal 6.10	Joomla! 1.5.10
Affiliate Tracking	Free Add On	Free Add On
Inventory Management	Free Add On	Free Add On
Pluggable Payments	Free Add On	Free Add On
Pluggable Shipping	Free Add On	Free Add On
Pluggable Tax	Free Add On	Free Add On
Point of Sale	No	Free Add On
Shopping Cart	Free Add On	Free Add On
Subscriptions	Free Add On	Free Add On
Wish Lists	Free Add On	Free Add On

Tabela III - Tabela Comparativa entre Drupal e Joomla

Anexo B

B.1. Instalação e configuração de *software*

B.1.1. Apache + PHP

- `yum install php php-devel php-gd php-imap php-ldap php-mysql php-odbc php-pear php-xml php-xmlrpc curl curl-devel perl-libwww-perl ImageMagick libxml2 libxml2-devel php-mbstring php-mcrypt`
- `chkconfig --levels 235 httpd on`
- `/etc/init.d/httpd start`

B.1.2.MySQL

- `yum install mysql mysql-devel mysql-server`
- `chkconfig --levels 235 mysqld on`
- `/etc/init.d/mysqld restart`

B.1.3. HAProxy

- `cd /opt`
- `wget http://haproxy.1wt.eu/download/1.4/src/haproxy-1.4.4.tar.gz`
- `tar xvfz haproxy-1.4.4.tar.gz`
- `cd / haproxy-1.4.4`
- `make TARGET=linux26 ARCH=i386`[1]
- `mv haproxy /opt`
- `. /haproxy -f /opt/haproxy.conf`
-

Posteriormente procedeu-se á alteração do ficheiro de configuração para efectuar a distribuição de carga pelos dois servidores Web, de notar que foram usadas duas configurações uma para receber pedidos encaminhados pelo servidor *proxy*, Squid, outra para receber directamente os pedidos do cliente.

Configuração 1

```
global
      maxconn 4096
      pidfile /var/run/haproxy.pid
      daemon
defaults
      mode http
      retries 3
      option redispatch
      maxconn 100000
      contimeout 900000
      clitimeout 900000
      srvtimeout 900000
      balance roundrobin
listen WEBS 172.20.109.82:8081
      option httpclose
      option forwardfor
      option httpchk
      server WEB_SERVER_1 172.20.109.81:80
      server WEB_SERVER_2 172.20.109.83:80
```

Configuração 2

```
global
      maxconn 4096
      pidfile /var/run/haproxy.pid
      daemon
defaults
      mode http
      retries 3
      option redispatch
      maxconn 100000
      contimeout 900000
      clitimeout 900000
      srvtimeout 900000
      balance roundrobin
listen WEBS 172.20.109.82:80
      option httpclose
      option forwardfor
      option httpchk
      server WEB_SERVER_1 172.20.109.81:80
      server WEB_SERVER_2 172.20.109.83:80
```

B.1.3.Squid

- `yum install squid`

Posteriormente procedeu-se á alteração do ficheiro de configuração do Squid situado em /etc/squid/squid.conf, de seguida seguem-se as configurações usadas para o Squid, a primeira configuração reencaminha os pedidos para o *Load Balancer*, a segunda reencaminha os pedidos para um dos servidores Web.

Configuração 1

```
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localhost
http_access allow  all
icp_access allow all
http_port 80 accel  defaultsite=172.20.109.82
cache_peer 172.20.109.82 parent 8081 0 no-query originserver
hierarchy_stoplist cgi-bin ?
acl QUERY urlpath_regex cgi-bin \?
cache deny QUERY
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern .               0       20%     4320
acl apache rep_header Server ^Apache
broken_vary_encoding allow apache
visible_hostname localhost
forwarded_for on
emulate_httpd_log on
redirect_rewrites_host_header off
```

Configuração 2

```
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localhost
http_access allow  all
icp_access allow all
http_port 80 accel  defaultsite=172.20.109.83
cache_peer 172.20.109.83 parent 80 0 no-query originserver
hierarchy_stoplist cgi-bin ?
acl QUERY urlpath_regex cgi-bin \?
cache deny QUERY
refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern .               0       20%     4320
acl apache rep_header Server ^Apache
broken_vary_encoding allow apache
visible_hostname localhost
forwarded_for on
emulate_httpd_log on
redirect_rewrites_host_header off
```

B.1.4.Memcache

- wget http://apt.sw.be/redhat/el5/en/i386/rpmforge/RPMS/rpmforge-release-0.3.6-1.el5.rf.i386.rpm
- RPM –install rpmforge-release-0.3.6-1.el5.rf.i386.rpm
- yum install memcached
- yum install libevent-devel
- memcache –u root –d –m 512 –p 11211

Para se instalar o módulo de Memcache para PHP seguiram-se os seguintes passos

- `yum install php-pecl-memcache`

B.1.5.Optimizações

De forma a poder aumentar o número de ligações que o sistema operativo suporta, neste caso CentOS, e também a melhorar a *performance* do servidor Apache, procederam-se ás seguintes modificações no sistema operativo:

- `ulimit -HSn 4096`
- `echo 1024 32768 > /proc/sys/net/ipv4/ip_local_port_range`

As alterações efetuada á configuração do servidor Apache, estando o ficheiro localizado em /etc/httpd/conf/httpd.conf foram as seguintes:

```
KeepAlive On
KeepAliveRequests 500
KeepAliveTimeout 2
StartServers 4
MinSpareServers 2
MaxSpareServers 3
MaxClients 100
MaxRequestsPerChild 5000
AllowOverride None
HostnameLookUps Off
EnableMMap On
EnableSendFile On
```

Anexo C

Gráfico representativo dos valores desvio padrão para o teste sobre a técnica de balanceamento de carga.

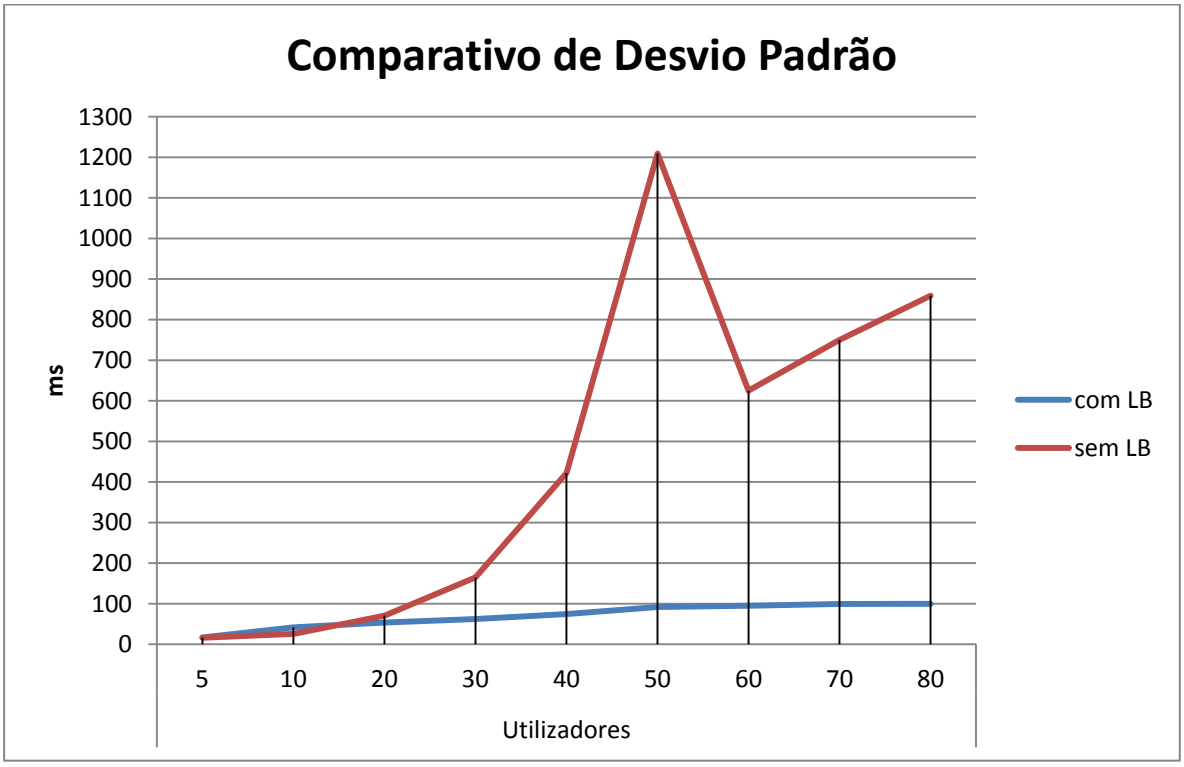


Figura 23 - Comparativo de Desvio Padrão

Gráfico representativo dos valores throughput para o teste efectuado sobre o módulo de pesquisa do CMS Drupal, com a adopção de *cache* de conteúdo dinâmico.

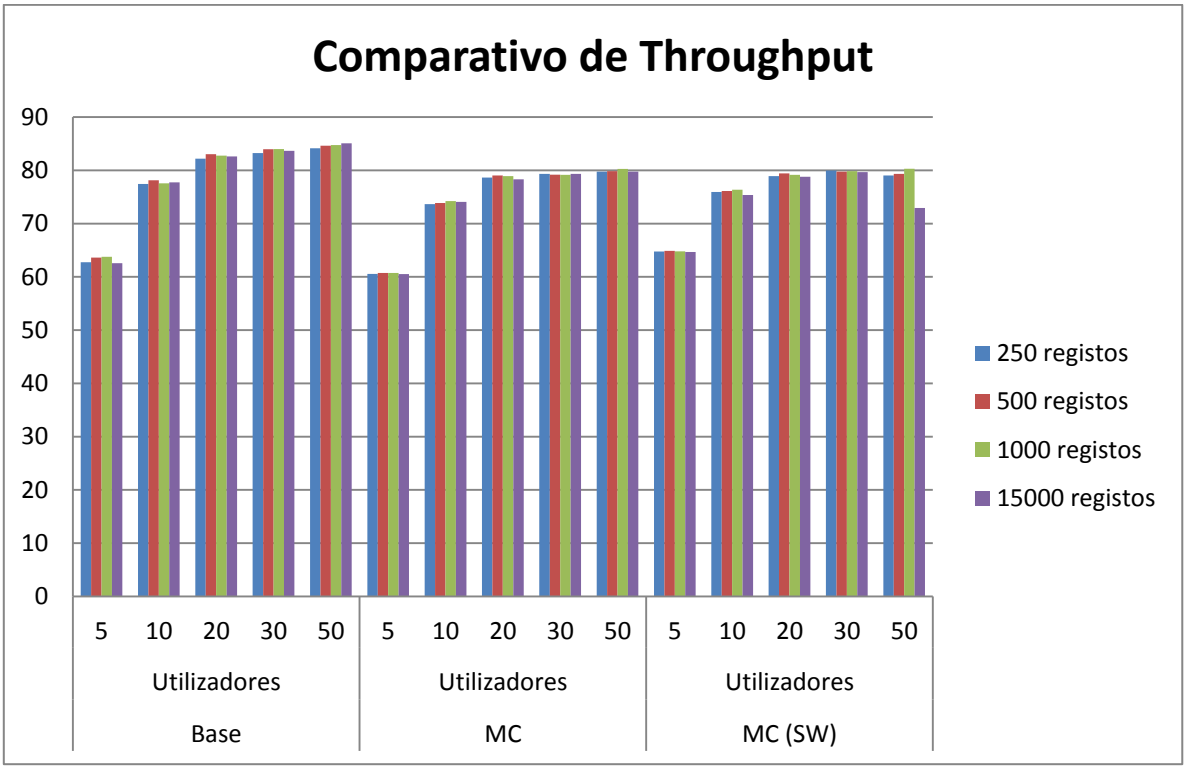


Figura 24 - Comparativo de *Throughput*

Gráfico representativo dos valores desvio padrão para o teste efectuado sobre o módulo de pesquisa do CMS Drupal, com a adopção de *cache* de conteúdo dinâmico.

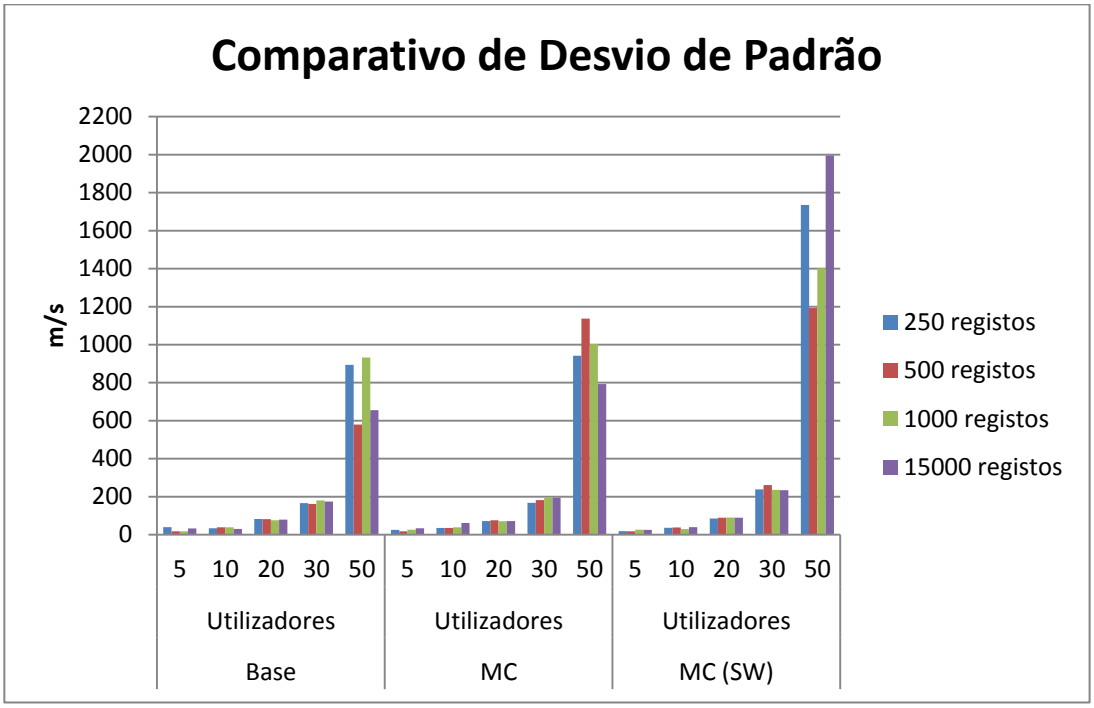


Figura 25 - Comparativo de Desvio Padrão

Logs referentes a um pedido ao servidor de *cache* de conteúdo dinâmico quando este se encontra vazio.

```
set:

variables
schema
1:6cc761aa00d44f1a585b2044bbe8feb7
theme_registry:garland
1:eb421d4c662445f5490a458a2144cee8
1:ba78f45b8e3a51ef99cea0cf15693ac4
1:8f7048e9b1c9182b60d5a1441ed8c35f
1:e842f8c253cc3b048a7427a7a22625b2
1:99f34a28c5fa5e67a1872ce943b11d64
1:902966c69eb1caf055bf149630f8d420
1:8dc577dbaec60d0d8a73a7afe081ae08
1:02563975f65816f86ed038bdbf10fddb
1:8d00632e79c6d1070a42330f0d44a396
links:navigation:tree-data:4667b392ba00a67fc17dc89e3957bbdf
links:navigation:page-cid:search/node/node:1
form_form-1111f1058b28a4afe2c7975b9d98bb91
links:primary-links:tree-data:5d6d3aaaaef5fba302ce62698fa37bbe
links:primary-links:page-cid:search/node/node:1
links:secondary-links:tree-data:5d6d3aaaaef5fba302ce62698fa37bbe
links:secondary-links:page-cid:search/node/node:1

hit:
```

Logs referentes a um pedido ao servidor de *cache* de conteúdo dinâmico quando este se encontra preenchido.

```
set:

form_form-34c8d9ae133a57febb401bb96dd774f4

hit:

variables
schema
1:6cc761aa00d44f1a585b2044bbe8feb7
theme_registry:garland
1:eb421d4c662445f5490a458a2144cee8
1:ba78f45b8e3a51ef99cea0cf15693ac4
1:8f7048e9b1c9182b60d5a1441ed8c35f
1:e842f8c253cc3b048a7427a7a22625b2
1:99f34a28c5fa5e67a1872ce943b11d64
1:902966c69eb1caf055bf149630f8d420
1:8dc577dbaec60d0d8a73a7afe081ae08
1:02563975f65816f86ed038bdbf10fddb
1:8d00632e79c6d1070a42330f0d44a396
links:navigation:page-cid:search/node/node:1
links:navigation:tree-data:4667b392ba00a67fc17dc89e3957bbdf
links:primary-links:page-cid:search/node/node:1
links:primary-links:tree-data:5d6d3aaaaef5fba302ce62698fa37bbe
links:secondary-links:page-cid:search/node/node:1
links:secondary-links:tree-data:5d6d3aaaaef5fba302ce62698fa37bbe
```