Claudine Santos Badue Gonçalves

Projeto e Análise de Sistemas de Busca na Web

Tese de doutorado apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de doutor em Ciência da Computação

Belo Horizonte
27 de Fevereiro de 2007

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Aos meus pais Anuor e Celuta;
e ao meu esposo Sérgio, eterno amor da minha vida.

"Assim diz Deus, o Senhor, que criou os céus e os estendeu,
formou a terra, e a tudo quanto produz;
que dá fôlego de vida ao povo que nela está,
e o espírito aos que andam nela.
Guiarei os cegos por um caminho que não conhecem,
fá-los-ei andar por veredas desconhecidas;
tornarei as trevas em luz perante eles,
e os caminhos escabrosos, planos.
Estas cousas lhes farei, e jamais os desampararei."

Bíblia, no livro de Isaías 42:5,16

Agradecimentos

A Jesus Cristo, meu Senhor e Salvador, pelo Seu grande amor para comigo, que foi mostrado na extensão máxima quando Ele sacrificou Sua vida por mim na cruz. A Jesus o meu agradecimento por cumprir as Suas promessas na minha caminhada pelo meu curso de doutorado, guiando-me por um caminho que eu não conhecia, por uma estrada que eu nunca havia pisado antes, iluminando a escuridão que me cercava e aplainando os caminhos ásperos. A Ele entrego toda a minha vida! A Ele dou toda gratidão e honra e glória e louvor e adoração agora e para todo o sempre!!!

Ao Prof. Nivio Ziviani pela sua participação tão impactante na minha formação, tanto como profissional quanto como ser humano. A ele a minha gratidão pela sua excelente orientação, sólido suporte, grande experiência e extensa visão na arena da pesquisa, que foram determinantes no cumprimento desta tese. Também, seu incentivo constante ajudou-me a perseverar desde o começo até o final deste trabalho. Além disso, ele me ensinou a viver em excelência e em autoconfiança. Ele será o meu principal referencial profissional para sempre!

Ao Prof. Berthier Ribeiro-Neto pela sua participação tão importante na minha formação profissional e pessoal. A ele o meu agradecimento por apresentar-me a idéia desta tese e por tantas discussões que ajudaram a estabelecer direções para o desenvolvimento deste trabalho. Desde o início do meu curso de doutorado, sua alegria me contagiava muito; poucos minutos de conversa com ele eram suficientes para me encher de força e esperança para continuar, mesmo em meio a muita confusão. Também, aprendi com ele a considerar um problema como uma grande oportunidade e não como um obstáculo intransponível. Estou certa de que a sua participação na minha formação não irá nunca deixar de influenciar minha vida.

Ao Prof. Ricardo Baeza-Yates pelas valiosas idéias, críticas e sugestões durante o desenvolvimento desta tese. A ele a minha gratidão por receber-me em quatro períodos

diferentes no Departamento de Ciência da Computação na Universidade do Chile para trabalharmos no desenvolvimento deste trabalho. Além de serem muito produtivas tecnicamente, estas viagens propiciaram-me a oportunidade de conhecer as belezas do país chileno.

Ao Prof. Artur Ziviani pela sua participação decisiva na reta final do desenvolvimento desta tese. A ele o meu agradecimento por se disponibilizar horas e horas para discutirmos sobre o trabalho, por receber-me em vários períodos diferentes no Laboratório Nacional de Computação Científica (LNCC) para trabalharmos no desenvolvimento desta tese, e por cooperar com tanta excelência na redação dos artigos científicos. Seu vasto conhecimento da área de Ciência da Computação, sua competência profissional e sua paixão pela pesquisa foram determinantes na conclusão deste trabalho.

Ao Prof. Virgílio Almeida e à Profa. Jussara Almeida pela sua participação no desenvolvimento do modelo de desempenho apresentado nesta tese. Seu conhecimento da especialidade de análise e modelagem de desempenho de sistemas de computação contribuiu consideravelmente para a conclusão deste trabalho.

Ao Prof. Philippe Navaux e ao Prof. Edmundo de Souza e Silva, membros externos da banca examinadora, por terem aceitado participar da defesa desta tese e pelas valiosas contribuições para a melhoria deste trabalho.

Ao meu colega Coutinho do Laboratório *E-Speed* por compartilhar comigo seu conhecimento e experiência na especialidade de sistemas operacionais. Sua disponibilidade e paciência foram muito importantes para o desenvolvimento desta tese.

Ao meu colega Charles do Laboratório para Tratamento da Informação (LATIN) por orientar-me em procedimentos relacionados à instalação e configuração do sistema operacional *Linux*. Sua prontidão para ajudar foi muito importante para o desenvolvimento desta tese.

Aos meus colegas do Laboratório para Tratamento da Informação (LATIN) — Álvaro, Anísio, Charles, David, Fabiano, Marco Cristo, Marco Modesto, Maria de Lourdes, Pável e Thierson — pela sua amizade e companhia em muitas horas exaustivas de trabalho durante o meu curso de doutorado.

Aos meus colegas da empresa *Akwan Information Technologies* — Bruno, Paulo e Ramurti — pelas reuniões técnicas que esclareceram diversos aspectos sobre o comportamento de mecanismos de busca na Web.

Ao Camillo, meu grande colega e amigo especial, pela parceria nos trabalhos das disciplinas de qualificação do meu curso de doutorado.

À Daniela, minha amiga especial e irmã em Cristo, pela companhia durante o período do meu curso de doutorado. A ela o meu agradecimento por dedicar tempo para as nossas reuniões semanais de oração e meditação na Palavra de Deus. A presença de Deus manifestada e a Palavra de Deus proclamada nestas reuniões me sustentaram e me alimentaram durante o período do meu curso de doutorado.

À Linnyer, minha grande amiga e irmã em Cristo, pelo seu compromisso de estar comigo em oração sempre. As nossas reuniões semanais de oração e meditação na Palavra de Deus ensinaram-me a lançar sobre Deus em oração toda a minha ansiedade. A ela a minha gratidão por jejuar e orar pelo sucesso da minha tese de doutorado, e por crer por mim quando eu estava sem forças para crer que chegaria no final deste trabalho.

Aos meus irmãos e irmãs na fé da Igreja Batista da Graça em Belo Horizonte pela comunhão nas reuniões da célula e pelas celebrações nos domingos à noite durante o período do meu curso de doutorado.

Ao Pastor Geraldo por pastorear a minha vida com tanto amor durante o período do meu curso de doutorado.

Aos meus irmãos e irmãs na fé da Igreja Presbiteriana de Anápolis por orarem por mim desde o começo do meu curso de doutorado e por participarem comigo dos feriados cheios de paz e restauração em Anápolis.

Ao Pastor Larry e sua esposa Ida, meus pais americanos, por me suportarem em oração e por me ajudarem a permanecer firme na fé através de suas poderosas palavras de encorajamento durante o período do meu curso de doutorado.

À Adeílde, minha mãe em Belo Horizonte, por cuidar de mim com tanto carinho e por várias vezes receber-me em sua casa com tanto amor durante o período do meu curso de doutorado.

À Juliana, Karina e Susana, minhas amigas inseparáveis, por participarem comigo na vida diária e por compartilharem comigo tanto os momentos de alegria quanto os momentos de tristeza durante o período do meu curso de doutorado.

Aos meus sogros Paulo e Celma pelo seu amor, suporte, incentivo e orações tão importantes para a conclusão desta tese.

À minha tia Doraci, Wagner, Bruno e Letícia pelo amor e carinho que sempre demonstraram para comigo em todas as fases da minha vida.

Aos meus avós Raul e Dulce por participarem com tanto amor do meu crescimento e formação. Eles serão sempre grandes exemplos de fé e caráter para a minha vida.

Aos meus irmãos Cassius e Christian pelo seu amor e por poder contar com eles em todo tempo.

Aos meus pais Anuor e Celuta pelo seu amor profundo e suporte incondicional desde o meu primeiro suspiro neste mundo. A eles a minha gratidão por cuidarem com amor incondicional de todas as áreas da minha vida e por investirem ilimitadamente em todas as fases da minha educação. A sua presença tão forte e orações incessantes têm sido cura para a minha alma e têm trazido paz ao meu coração.

Ao Sérgio, meu esposo e melhor amigo, por simplesmente me completar! A ele o meu agradecimento por viver para mim, por me amar com tanta intensidade e por me fazer sentir tão amada. Em seus braços eu encontro um lugar seguro. A ele a minha gratidão também por viver comigo cada minuto durante o período de desenvolvimento desta tese. Ele realmente produziu este trabalho juntamente comigo. A ele a minha vida e o meu amor eterno!!!

Abstract

Web search engines are expensive to maintain, expensive to operate, and hard to design. Modern search engines rely on clusters of server machines for query processing. Thus, the performance of parallel query processing in a cluster of index servers is crucial for modern Web search engines. The objective of this thesis is to provide a performance framework for the design and analysis of the infrastructure of Web search engines. In this framework we (i) investigate and analyze the imbalance issue in a computational cluster composed of homogeneous index servers and (ii) propose a capacity planning model for Web search engines.

In a cluster of index servers, the response time basically depends on the service time of the slowest server to generate a partial ranked answer. Previous approaches investigate performance issues in this context using simulation, analytical modeling, experimentation, or a combination of them. Nevertheless, these approaches simply assume balanced service times among homogeneous index servers, a scenario that we did not observe in our experimentation. On the contrary, we found that even with a balanced distribution of the document collection among index servers, relations between the frequency of a query in the collection and the size of its corresponding inverted lists lead to imbalances in query service times at these same servers, because these relations affect disk cache behavior. Further, the relative sizes of the main memory at each index server (with regard to disk space usage) and the number of servers participating in the parallel query processing also affect imbalance of local query service times.

Predicting the performance of a Web search engine is usually done empirically through experimentation, requiring a costly setup. Thus, modeling is of natural appeal in this context. We introduce a capacity planning model for Web search engines that considers the imbalance in query service times among homogeneous index servers. Our model, which is based on a queueing network, is simple and yet reasonably accurate. We discuss how we tune it up and how we apply it to predict, for instance, the impact on the query response time when parameters such as CPUs and disks are changed. This allows the manager of the search engine to determine a priori whether a new configuration of the system will keep the query response under specified constraints. Our approach is distinct and, we believe, useful to predict the performance of real Web search engines.

Resumo

Mecanismos de busca na Web são caros para manter, caros para operar, e difíceis de projetar. Mecanismos modernos de busca contam com *clusters* de máquinas servidoras para processamento de consultas. Assim, o desempenho do processamento paralelo de consultas num *cluster* de servidores de índice é crucial para os mecanismos modernos de busca na Web. O objetivo desta tese é prover um arcabouço para o projeto e análise da infra-estrutura de mecanismos de busca na Web. Neste arcabouço (i) investigamos e analisamos a questão do desbalanceamento num *cluster* computacional composto por servidores de índice homogêneos e (ii) propomos um modelo de planejamento de capacidade para mecanismos de busca na Web.

Num *cluster* de servidores de índice, o tempo de resposta depende basicamente do tempo de serviço do servidor mais lento para gerar uma resposta ordenada parcial. Abordagens anteriores investigam questões de desempenho neste contexto usando simulação, modelagem analítica, experimentação, ou uma combinação delas. Entretanto, estas abordagens simplesmente assumem tempos de serviço balanceados entre os servidores de índice homogêneos, um cenário que não observamos em nossa experimentação. Ao contrário, verificamos que mesmo com uma distribuição balanceada da coleção de documentos entre os servidores de índice, relações entre a freqüência de uma consulta na coleção e o tamanho de suas listas invertidas correspondentes levam a desbalanceamentos nos tempos de serviço de uma consulta nestes mesmos servidores, porque estas relações afetam o comportamento do *cache* do disco. Além disso, os tamanhos relativos da memória principal em cada servidor de índice (com referência ao uso do espaço em disco) e o número de servidores que participam do processamento paralelo de consultas também afetam o desbalanceamento nos tempos locais de serviço de uma consulta.

A predição do desempenho de um mecanismo de busca na Web é usualmente feita empiricamente através de experimentação, requerendo uma configuração custosa. Assim, a modelagem tem um apelo natural neste contexto. Introduzimos um modelo de planejamento de capacidade para mecanismos de busca na Web que considera o desbalanceamento nos tempos de serviço de uma consulta entre os servidores de índice homogêneos. Nosso modelo, que é baseado numa rede de filas, é simples e razoavelmente preciso. Discutimos como ajustá-lo e como usá-lo para predizer, por exemplo, o impacto no tempo de resposta da consulta quando parâmetros tais como CPUs e discos são alterados. Isto permite ao gerente da máquina de busca determinar a priori se uma nova configuração do sistema irá manter o tempo de resposta sob determinadas restrições. Nossa abordagem é distinta e, acreditamos, útil para predizer o desempenho de mecanismos de busca reais.

Publicações

Artigos Publicados em Periódicos (Completo)

- C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Analyzing imbalance among homogeneous index servers in a web search system. *Information Processing and Management* (IP&M), 43(3):592-608, 2007.

Trabalhos em Eventos (Completo)

- C. S. Badue, R. Barbosa, P. Golgher, B. Ribeiro-Neto, and N. Ziviani. Distributed processing of conjunctive queries. In *ACM SIGIR Workshop on Heterogeneous and Distributed Information Retrieval*, Salvador, BA, Brazil, 2005.

- C. S. Badue, R. Baeza-Yates, W. Meira Jr., B. Ribeiro-Neto, and N. Ziviani. Distributed architecture for information retrieval. In *Proceedings of the 1st International Seminar on Advanced Research in E-Business (EBR'02)*, pages 114-122, Rio de Janeiro, RJ, Brazil, 2002.

- C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 10-20, Laguna de San Rafael, Chile, 2001.

Trabalhos em Eventos (Resumo)

- C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Modeling performance-driven workload characterization of web search systems. In *Pro-*

ceedings of the ACM 15th Conference on Information and Knowledge Management (CIKM'06), pages 842 - 843, Arlington, VA, USA, 2006.

- C. S. Badue, R. Barbosa, P. Golgher, B. Ribeiro-Neto, and N. Ziviani. Basic issues on the processing of web queries. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 577-578, Salvador, BA, Brazil, 2005.

Resumo Estendido

Introdução

Motivação

A World Wide Web data do fim da década dos anos 80 [17] e ninguém poderia ter imaginado seu impacto atual. A explosão no uso da Web e seu crescimento exponencial são agora bem conhecidos. Apenas a quantidade de dados textuais disponíveis é estimada para estar na ordem de *terabytes*. Adicionalmente, outros meios, tais como imagens, áudio, e vídeo, também estão disponíveis. Isto provoca a necessidade de ferramentas eficientes para administrar, recuperar, e filtrar informação desta base de dados enorme e diversificada.

Os mecanismos de busca transformaram-se numa ferramenta essencial e popular para lidar com a enorme quantidade de informação encontrada na Web. Um estudo feito por iProspect [30] revela que 35.1% dos usuários da Internet usam os mecanismos de busca pelo menos uma vez ao dia, 21.2% usam os mecanismos de busca quatro ou mais vezes ao dia, e 22.7% usam os mecanismos de busca múltiplas vezes por semana, o que indica que o uso de mecanismos de busca é um tipo popular de atividade *online*.

Os mecanismos de busca na Web requerem uma enorme quantidade de recursos computacionais para lidar com o tráfego entrante de consultas, que é caracterizado freqüentemente por altos picos. Além disso, o fato do número de documentos disponíveis na Web crescer constantemente — existem agora pelo menos 20 bilhões de documentos na Web [24] — torna o problema ainda mais desafiador. Para lidar com estas exigências, os mecanismos modernos de busca na Web contam com *clusters* de máquinas servidoras para o processamento das consultas [14, 18, 45].

A arquitetura de um mecanismo de busca típico na Web é composta por *clusters* de servidores de índice, com os documentos divididos entre eles (cada servidor de índice armazena uma parte da coleção de documentos e um índice para ela). Esta arquitetura é

geralmente referida como "particionamento por documento" e é preferida porque simplifica a manutenção, simplifica a geração do índice (que pode ser feita localmente), e degrada-se suavemente (porque a falha de um servidor de índice não impede que qualquer consulta seja respondida, embora o conjunto de resposta final possa não conter todos os documentos relevantes na coleção). O *cluster* inclui também um *broker* que se comunica com os vários servidores de índice, como ilustrado na Figura 1.

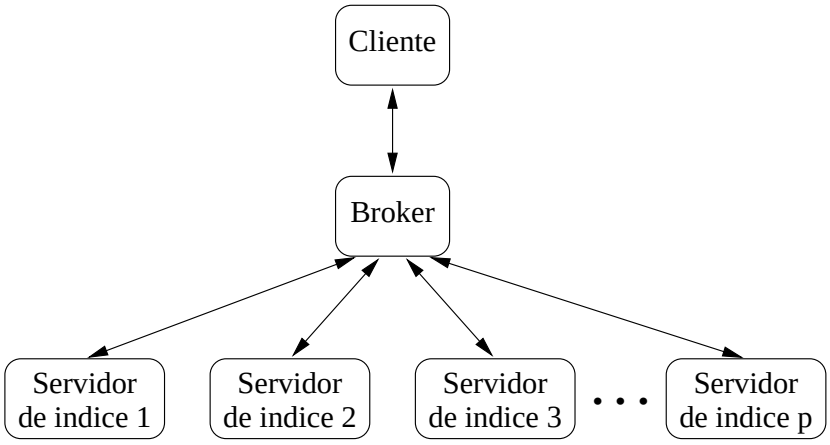


Figura 1: Arquitetura de um mecanismo de busca típico.

Uma consulta de usuário chega no mecanismo de busca através do *broker*, que envia uma cópia da consulta para cada servidor de índice para processamento local. Tipicamente, cada servidor de índice retorna seus 10 documentos mais relevantes para o *broker*, que executa uma mesclagem em memória para determinar as 10 respostas finais a serem enviadas para o usuário.

O processamento de uma consulta pode ser dividido em duas fases principais consecutivas [14]. Uma primeira fase que consiste em recuperar referências aos documentos que contém todos os termos da consulta[1] e ordená-los de acordo com alguma métrica de relevância (feito geralmente pelos servidores de índice). Uma segunda fase que consiste em tomar as respostas mais relevantes, tipicamente 10, e gerar fragmentos (do documento), título, e informação sobre a URL para cada uma delas (usualmente feito por um *cluster* de servidores de documento, cada servidor detendo uma parte da coleção de documentos). Enquanto a segunda fase tem custo aproximadamente constante, independente do tamanho da coleção de documentos, a primeira fase tem um custo que cresce com o tamanho da coleção. Portanto, o desempenho da primeira fase é crucial para manter a escalabilidade de

[1]Usar a interseção dos termos da consulta é agora prática padrão na Web

mecanismos modernos de busca que lidam com uma quantidade crescente de documentos da Web.

Dada a complexidade envolvida no projeto de mecanismos eficientes de busca na Web e o papel chave que tais sistemas desempenham no uso da Internet hoje em dia, é de importância máxima compreender o comportamento de mecanismos de busca na Web. Isto é essencial para a análise de desempenho e planejamento de capacidade de tais sistemas, a fim de permiti-los enfrentar adequadamente a demanda crescente dos usuários.

Objetivos e Contribuições

O objetivo desta tese é prover um arcabouço para o projeto e análise da infra-estrutura de mecanismos de busca na Web. Nosso objetivo é ter uma ferramenta simples e razoavelmente precisa que possa responder questões de planejamento de capacidade tais como:

(i) Dada uma coleção composta por n documentos distribuída através de p máquinas, que tipo de garantias de tempo médio de resposta de uma consulta que se pode esperar?

(ii) Que tipo de otimização nos recursos da máquina pode produzir uma redução no tempo médio de resposta de uma consulta para satisfazer um objetivo de nível de serviço definido pelo gerente do mecanismo de busca na Web?

(iii) Qual é o número mínimo de réplicas do *cluster* de servidores de índice que garantirá que, na média, o tempo de resposta de uma consulta num período de pico não excederá o limite definido pelo gerente do mecanismo de busca na Web?

Nesta tese, analisamos o desempenho da recuperação dos documentos mais relevantes para uma dada consulta de um usuário, i.e., a primeira fase da tarefa de processamento da consulta. As principais contribuições desta tese são:

- Investigação e análise da questão do desbalanceamento num *cluster* computacional para processamento paralelo de consultas composto por servidores de índice homogêneos, como apresentado no Capítulo 4. Verificamos na prática um desbalanceamento consistente por consulta no tempo de serviço nos servidores de índice, apesar da distribuição dos tamanhos das listas invertidas nos vários servidores ser completamente balanceada. Este é um resultado experimental importante porque nossas descobertas

contradizem a suposição usual de tempos balanceados de serviço adotada pelos modelos teóricos anteriores encontrados na literatura [20,23,44]. Além disso, identificamos e analisamos as fontes principais de desbalanceamento: o uso do *cache* do disco, o tamanho da memória principal nos servidores de índice homogêneos, e o número de servidores de índice no *cluster*.

- Um modelo de planejamento de capacidade para mecanismos de busca na Web que considera o desbalanceamento nos tempos de serviço das consultas entre os servidores de índice homogêneos, como apresentado no Capítulo 5. Nosso modelo, baseado em redes de filas, é simples e razoavelmente preciso. Para ajustar os parâmetros do nosso modelo, executamos experimentos num *cluster* pequeno de servidores de índice. Uma vez que os parâmetros chave foram estimados, verificamos a precisão do modelo comparando suas predições com os resultados experimentais produzidos também usando o *cluster* pequeno de servidores de índice. Finalmente, ilustramos como usar nosso modelo para predizer o tempo de resposta de uma consulta ao adotarmos CPUs e discos mais rápidos do que aqueles em uso. Em nosso exemplo, consideramos um cenário realístico, onde uma coleção de 20 bilhões de documentos é distribuída através de 2,000 servidores de índice.

Analisando o Desbalanceamento entre Servidores de Índice Homogêneos

Nesta seção, investigamos e analisamos o desbalanceamento entre os servidores de índice homogêneos num *cluster* para processamento paralelo de consultas. Servidores de índice homogêneos têm a mesma configuração de *hardware* e *software*.

Na arquitetura para processamento paralelo de consultas, caracterizada por um particionamento local da coleção de documentos, o tempo de resposta de uma consulta é determinado pelo tempo de serviço do servidor de índice mais lento. Como conseqüência, desbalanceamento nos tempos de serviço entre o servidores de índice aumenta o tempo de resposta de uma consulta executada pelo *cluster* de servidores.

Uma medida comum contra o desbalanceamento é distribuir a coleção inteira de documentos entre os servidores de índice homogêneos de forma balanceada, tal que cada servidor manipule uma quantidade similar de dados para processar uma dada consulta. Como conseqüência, espera-se que os tempos de serviço nos servidores de índice homo-

gêneos também sejam aproximadamente balanceados. De fato, este cenário idealizado de tempos de serviço balanceados é uma suposição usual considerada por modelos teóricos para mecanismos de busca na Web [20, 23, 44]. Entretanto, num cenário real, relações entre as freqüências das consultas e os tamanhos das listas invertidas correspondentes levam a desbalanceamentos nos tempos de serviço das consultas.

Nesta seção, investigamos e analisamos a questão do desbalanceamento num *cluster* computacional composto por servidores de índice homogêneos. Como principal contribuição, verificamos que o cenário idealizado de tempos de serviço balanceados nos servidores de índice homogêneos com volumes de dados similares não é provável de ser encontrado na prática. Este é um resultado experimental importante porque nossas descobertas contradizem a suposição usual que é obviamente considerada como válida por modelos teóricos anteriores. Além disso, identificamos e analisamos as principais fontes de desbalanceamento: o uso do *cache* do disco, o tamanho da memória principal nos servidores de índice homogêneos, e o número de servidores no *cluster*.

Para os experimentos relatados nesta seção, usamos um *cluster* de 7 servidores de índice homogêneos. A coleção de teste é composta por 10 milhões de páginas Web coletadas pelo mecanismo de busca TodoBR da Web brasileira em 2003. O conjunto de consultas usados em nossos testes é composto por 100 mil consultas, extraídas de um registro parcial de consultas submetidas ao mecanismo de busca TodoBR em Setembro de 2003.

Definimos o "desbalanceamento de uma dada consulta" como a razão entre o tempo de serviço máximo e o tempo de serviço médio dos servidores de índice que participam do processamento paralelo desta consulta em particular. Esta métrica de desbalanceamento é igual a 1 num cenário perfeitamente balanceado que produz um tempo de serviço máximo exatamente igual ao tempo de serviço médio. À medida que a métrica de desbalanceamento torna-se progressivamente maior que 1, existe uma forte indicação de que o tempo de resposta de uma consulta é dominado por um tempo de serviço muito maior de um único servidor de índice.

Para evitar desbalanceamento entre os servidores de índice, optamos por balancear as distribuições do tamanho das listas invertidas que compõem os índices invertidos locais. Para isto, simplesmente atribuímos cada documento a um servidor de índice de forma aleatória. A Figura 2 ilustra a função de probabilidade do tamanho das listas invertidas que compõem os 7 índices invertidos locais em nosso *cluster* com 7 servidores de índice. Observamos que as distribuições do uso do armazenamento são muito similares através dos diferentes servidores de índice — na verdade, elas se sobrepõem na Figura 2, indicando que

a atribuição aleatória dos documentos nos servidores funciona muito bem para balancear o uso do armazenamento entre os servidores.

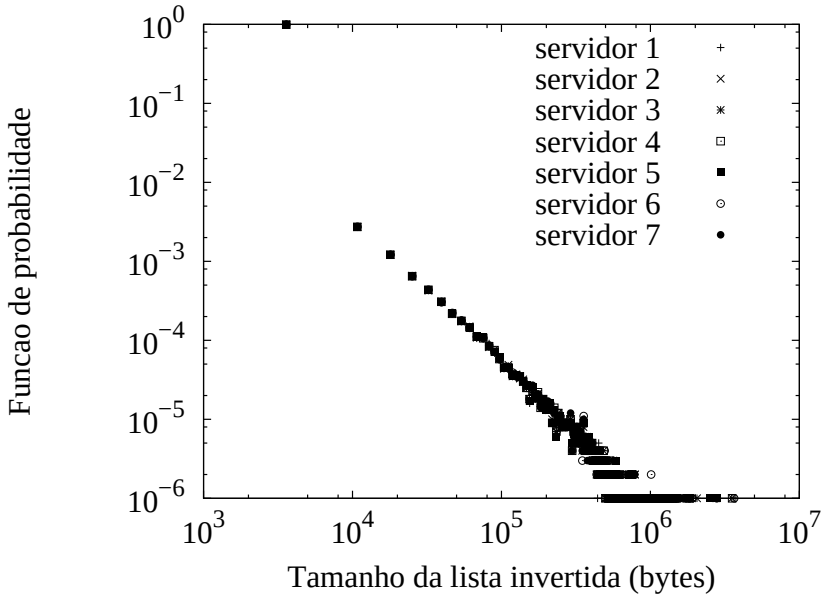


Figura 2: Função de probabilidade do tamanho das listas invertidas.

Embora a utilização do espaço em disco nos servidores de índice seja balanceada, como mostrado na Figura 2, investigamos se este uso balanceado do armazenamento entre as subcoleções reflete em tempos locais de serviço balanceados entre os servidores de índice, ou não. A Figura 3 ilustra as distribuições dos tempos (médio, máximo, e mínimo) locais de serviço por consulta. As barras de intervalo representam os tempos mínimo e máximo de serviço para cada consulta.

Como resultado dos nossos experimentos, verificamos na prática um desbalanceamento consistente por consulta nos tempos de serviço entre os servidores de índice, apesar da distribuição do tamanho das listas invertidas nos vários servidores de índice ser completamente balanceada. Motivados por este resultado inesperado, que contradiz a suposição usual de tempos balanceados de serviço adotada pelos modelos teóricos encontrados na literatura, conduzimos uma análise experimental detalhada para investigar as fontes do desbalanceamento observado. Conseqüentemente, identificamos as fontes principais de desbalanceamento: o uso do *cache* do disco, o tamanho da memória principal nos servidores de índice homogêneos, e o número de servidores no *cluster*.

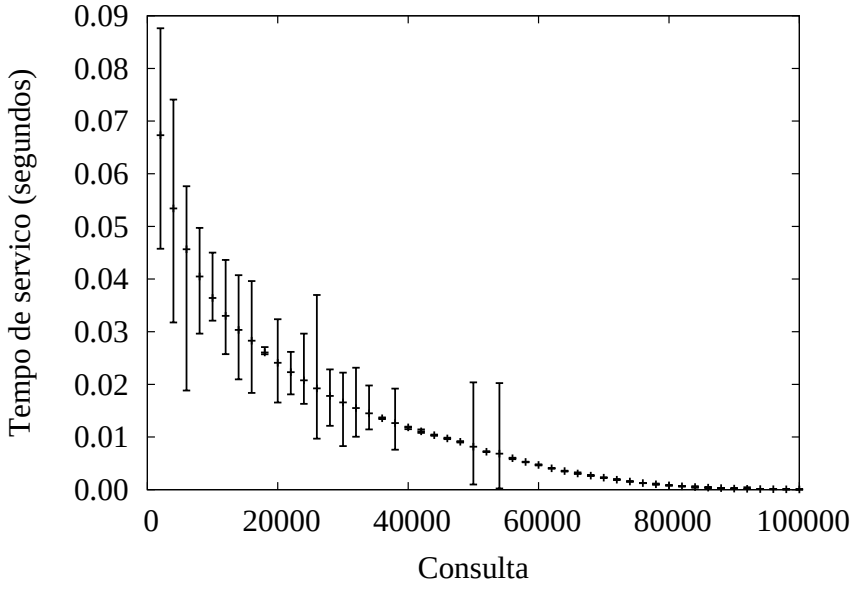


Figura 3: Distribuição dos tempos locais de serviço por consulta.

Influência do Uso do *Cache* do Disco

Identificamos o uso do *cache* do disco nos diferentes servidores de índice como a principal fonte de desbalanceamento. A Figura 4 mostra a função densidade de probabilidade dos tempos locais de acesso ao disco no nosso *cluster* com 7 servidores de índice. Observamos que os tempos de acesso ao disco em todos os servidores de índice estão agrupados basicamente em duas regiões principais: a primeira região é relacionada aos tempos de acesso ao disco menores que $4,5$ milisegundos e a segunda região aos tempos de acesso ao disco maiores que $4,5$ milisegundos. Atribuímos a primeira região de tempos locais de acesso ao disco menores às consultas cujas listas invertidas são encontradas no *cache* do disco (referida como "região do *cache*"), e a segunda região de tempos locais de acesso ao disco maiores às consultas cujas listas invertidas tiveram que ser realmente recuperadas do disco (referida como "região do disco").

Verificamos que o desbalanceamento nos tempos de serviço entre os servidores de índice aumenta com o número de servidores operando na região do *cache*, como mostrado na Figura 5. Os pontos na Figura 5 mostram o desbalanceamento para cada consulta e a linha mostra o desbalanceamento médio sobre as consultas em função do número de servidores de índice operando na região do *cache*. Este valor é complementar ao número de servidores de índice operando na região do disco. Por exemplo, para uma consulta particular sendo

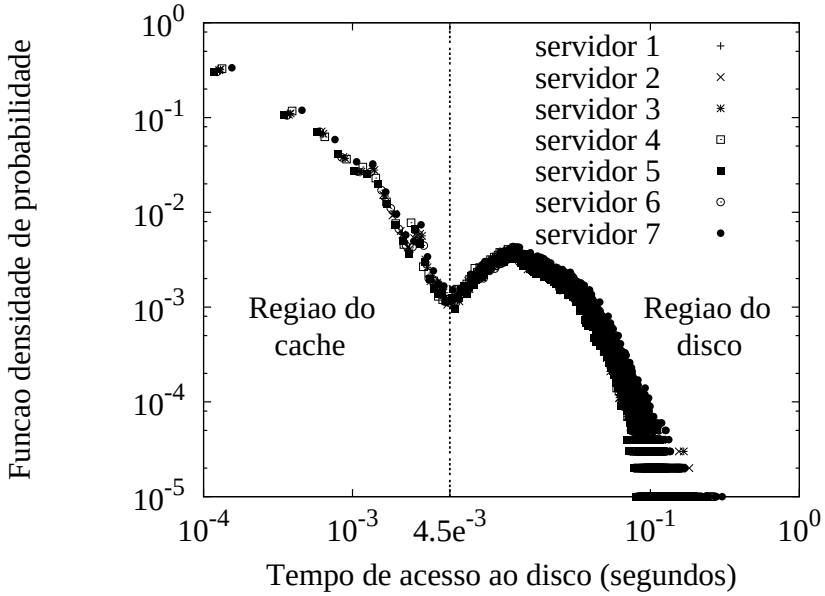


Figura 4: Função densidade de probabilidade dos tempos locais de acesso ao disco.

processada em nosso *cluster* de 7 servidores de índice, se a Figura 5 mostra que 2 deles operam na região do *cache*, então necessariamente os outros 5 estão na região do disco, diretamente influenciando a magnitude do desbalanceamento.

Para melhor entender como o *cache* do disco diretamente impacta o desbalanceamento, é importante olhar atentamente para o desbalanceamento médio na Figura 5 para alguns cenários representativos: sem *cache*, o pior caso, e o melhor caso. No cenário sem *cache* (i.e., 0 no eixo-x da Figura 5), todos os servidores de índice realmente acessam o disco para recuperar os dados necessários, obtendo o menor desbalanceamento (1.38) entre os casos onde existe pelo menos um servidor operando na região do disco. O pior caso para o desbalanceamento (i.e., 6 no eixo-x da Figura 5) apresenta um desbalanceamento muito maior (3.45) porque um único servidor de índice tem um tempo de serviço muito maior que os tempos de serviço correspondentes em todos os servidores restantes, levando assim a um alto valor de desbalanceamento. Isto acontece porque existe um único servidor de índice que tem um tempo de serviço grande e um conjunto de outros servidores que têm tempo de serviço muito menor porque recuperam os dados necessários do *cache* do disco. Pelo contrário, o melhor caso a evitar desbalanceamento (i.e., 7 no eixo-x da Figura 5) resulta num desbalanceamento médio de 1.08, sendo alcançado quando todos os servidores de índice operam na região do *cache*, resultando assim num valor de desbalanceamento

menor devido aos tempos de acesso ao disco relativamente menores e similares através do *cluster* de servidores.

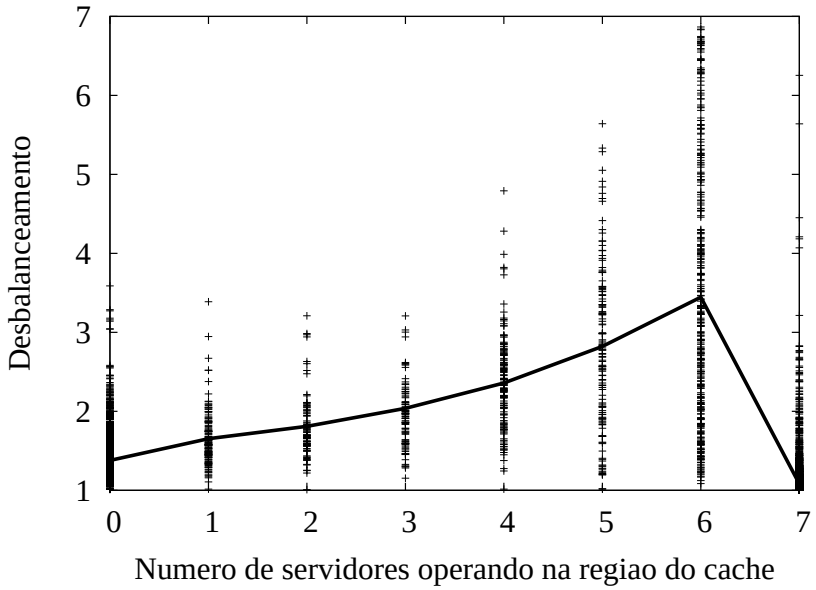


Figura 5: Desbalanceamento causado pelo número de servidores de índice operando na região do *cache*.

O melhor caso e o cenário sem *cache* apresentam os dois menores resultados desbalanceados, como conseqüência de terem todos os servidores de índice operando na mesma região (do *cache* ou do disco), assim provendo uma diferença não abrupta entre os tempos de serviço dos servidores participantes. Entretanto, o cenário sem *cache* ainda produz um desbalanceamento significativamente mais alto com respeito ao melhor caso, o que pode ser explicado pela variância mais alta encontrada no acesso direto ao disco quando comparada com a variância encontrada no acesso à memória. Além de ter o menor desbalanceamento, o melhor caso também provê o tempo de resposta mais rápido desde que todos os dados necessários para processar uma consulta são encontrados nos *caches* dos discos dos servidores de índice.

Além disso, analisamos o relacionamento entre o tamanho das consultas e a freqüência das consultas na coleção, investigando se existem ligações com o uso do *cache* do disco. O tamanho de uma consulta é dado pela soma dos tamanhos das listas invertidas relativas a seus termos. Portanto, consideramos separadamente as consultas que encontram um determinado nível de relação entre o tamanho das listas invertidas que demandam e sua

freqüência na coleção, e aquelas que não. Para isto, calculamos a relação como a razão entre o tamanho da consulta e a freqüência da consulta. Se esta razão é maior ou igual a 0.25 e menor ou igual a 4, então o tamanho e a freqüência da consulta são relacionados por um fator de 4, o que consideramos como representando um razoável nível de relação entre eles. Portanto, consultas que caem neste critério são consideradas relacionadas, caso contrário elas são consideradas não-relacionadas.

A Figura 6 mostra o tamanho normalizado das consultas em função da freqüência normalizada das consultas na coleção, mas fazemos a distinção entre consultas relacionadas e consultas não-relacionadas. Quando fazemos esta distinção, é interessante analisar separadamente três diferentes regiões representativas que aparecem na Figura 6: (i) a Região 1 é caracterizada por dados não-relacionados onde o tamanho das consultas está prevalecendo sobre a freqüência das consultas; (ii) a Região 2 contém consultas relacionadas; e (iii) a Região 3 é caracterizada por uma região não-relacionada onde a freqüência das consultas está prevalecendo sobre o tamanho das consultas.

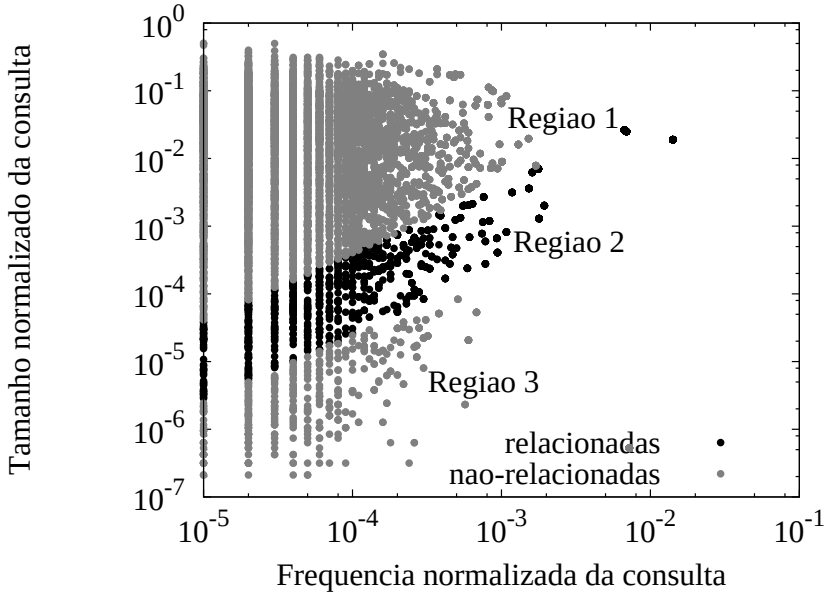


Figura 6: Comparando a freqüência das consultas e o tamanho das consultas.

A Figura 7 compara o tempo de serviço para consultas relacionadas e não-relacionadas com respeito ao seu tamanho e sua freqüência na coleção. Esta comparação mostra claramente que as consultas bem relacionadas (Região 2) tiram um proveito maior do *cache* do disco. Isto acontece porque elas têm o melhor compromisso entre o tamanho de suas

listas invertidas e sua freqüência na coleção. Por um lado, as listas invertidas maiores são demandadas pelas consultas mais freqüentes, favorecendo o uso do *cache* do disco por estas listas invertidas grandes. Por outro lado, consultas raras, cujas listas invertidas são improváveis de serem encontradas no *cache* do disco, requerem as listas invertidas menores que não demandam tempos grandes de transferência do disco. Para os dados não-relacionados na Região 1, a freqüência das consultas é proporcionalmente menor que o tamanho das consultas. Isto implica que consultas raras demandam listas invertidas maiores, assim resultando em não uso do *cache* do disco e grandes atrasos de transferências. Os dados não-relacionados na Região 3 enfrentam o oposto: os termos das consultas impõem volumes de dados relativamente pequenos a serem recuperados no sistema, assim obtendo tempos de serviço pequenos através de atrasos pequenos de transferência ou através do uso do *cache* do disco. Embora estas consultas tenham tempos de serviço pequenos elas não são tão numerosas quanto as consultas relacionadas ou as consultas não-relacionadas na Região 1. Portanto, consultas relacionadas prevalecem como um grupo que obtém os menores tempos de serviço.

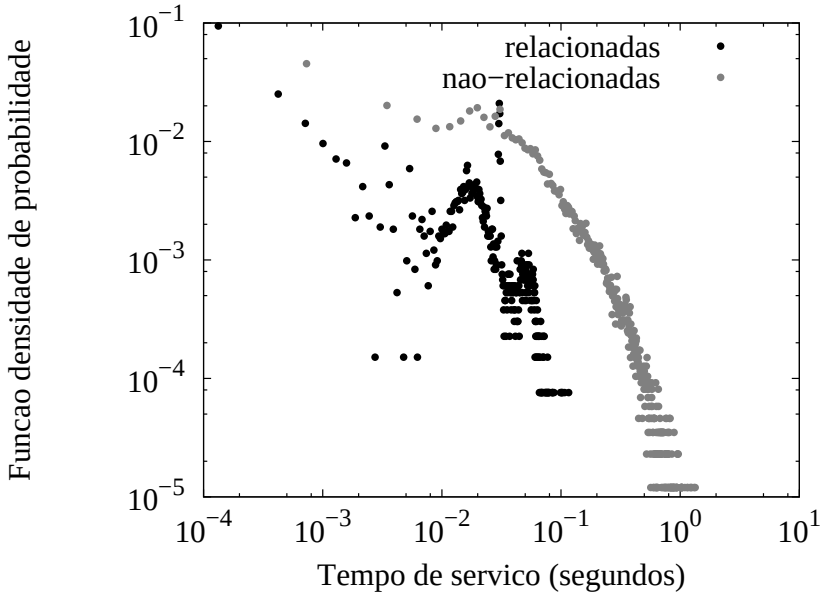


Figura 7: Função densidade de probabilidade dos tempos de serviço das consultas relacionadas e não-relacionadas.

Estes resultados sobre a influência do *cache* do disco no desbalanceamento também sugerem que quanto maior a memória disponível para o *cache* do disco, menor o desbalan-

ceamento, e quanto maior o número de servidores de índice no *cluster*, maior o desbalanceamento, como discutiremos a seguir.

Influência do Tamanho da Memória Principal e do Número de Servidores de Índice

Outra fonte de desbalanceamento é o tamanho da memória principal dos servidores de índice porque isto afeta a disponibilidade dos dados na região do *cache* nos servidores e, conseqüentemente, o desbalanceamento. A Figura 8 mostra o desbalanceamento médio em função do número de servidores de índice em nosso *cluster* com 7 servidores de índice, ao variar o tamanho da memória principal em cada servidor. Observamos que o desbalanceamento médio nos tempos de serviço entre os servidores de índice aumenta enquanto o tamanho da memória principal diminui, como era de se esperar. Por um lado, quando o tamanho da memória principal é relativamente grande comparado ao tamanho do índice local armazenado nos servidores de índice, existe maior capacidade de memória disponível para o sistema operacional executar operações do *cache* do disco. Isto implica que os tempos locais de acesso ao disco em todos os servidores de índice caem na região do *cache* para um alto percentual de consultas na nossa coleção e este é exatamente o melhor cenário que produz o menor desbalanceamento. Por outro lado, considerando uma memória principal relativamente menor disponível para o *cache* do disco, os servidores de índice precisam realmente recuperar as listas invertidas do disco. Neste cenário, as consultas estão mais susceptíveis ao desbalanceamento porque alguns blocos do disco podem ser encontrados no *cache* do disco de uns poucos servidores de índice e não serem encontrados no *cache* do disco dos servidores restantes.

Uma outra fonte de desbalanceamento é o número de servidores de índice no *cluster*, porque a probabilidade de ocorrer variação entre os tempos locais de serviço aumenta com o número de servidores de índice que participam no processamento paralelo da consulta. Observamos na Figura 8 que, para um tamanho fixo da memória principal, o desbalanceamento médio nos tempos de serviço entre os servidores de índice aumenta com o número de servidores de índice que participam no processamento paralelo da consulta. Já discutimos que o desbalanceamento médio aumenta com o número de servidores de índice que operam na região do *cache*, como mostrado na Figura 5. Entretanto, isto indica que quanto maior o número de servidores de índice que participam no processamento paralelo da consulta, maior a probabilidade de aumentar a razão entre o número de servidores que operam na

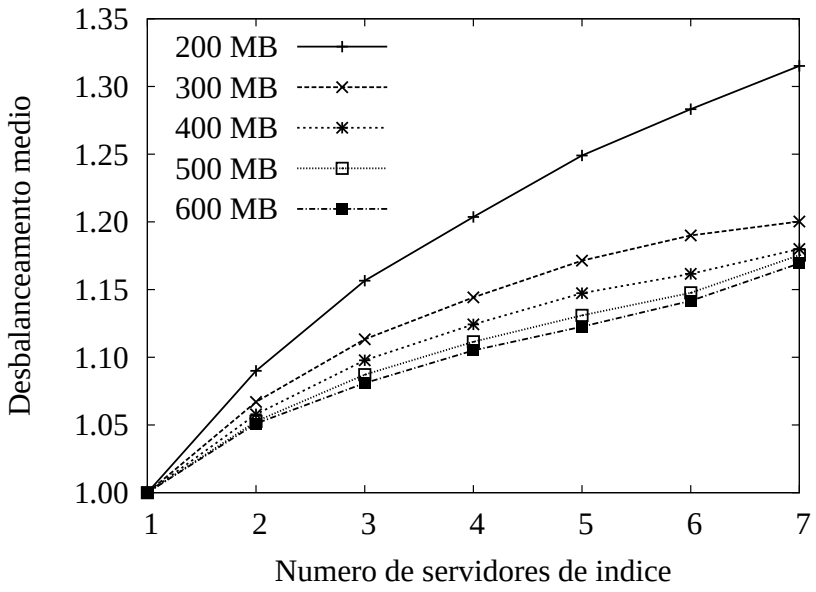


Figura 8: Desbalanceamento médio em função do tamanho da memória principal nos servidores de índice.

região do *cache* e aqueles que operam na região do disco. Como conseqüência, isto leva a desbalanceamentos maiores nos tempos de serviço por consulta no *cluster*.

Um Modelo de Planejamento de Capacidade para Mecanismos de Busca na Web

Nesta seção, propomos um modelo de planejamento de capacidade para mecanismos de busca na Web que considera o desbalanceamento nos tempos de serviço de uma consulta entre os servidores de índice homogêneos. Nossa estratégia de planejamento de capacidade conta com um modelo analítico baseado em filas para estimar o "tempo médio de resposta do sistema". O mecanismo de busca na Web é representado por uma rede de filas aberta composta pelo *broker* e pelo subsistema de servidores de índice. Assumimos que os servidores de índice têm recursos homogêneos (como seria o caso em vários cenários reais) e que a coleção de documentos é uniformemente distribuída através de todos os servidores. Assim, consideramos que a carga está balanceada através de todos os servidores de índice. Finalmente, a rede que conecta os servidores de índice e o *broker* é tipicamente uma rede de alta velocidade e, como observado em nossos experimentos, introduz atrasos desprezíveis

no tempo de resposta do sistema. Portanto, a rede não é explicitamente representada em nosso modelo.

Em nosso modelo de busca na Web, o subsistema composto por todos os servidores de índice é modelado como uma rede de filas *fork-join* [37]. Numa rede de filas *fork-join*, cada tarefa (i.e., consulta) que chega é dividida (i.e., *fork*) em p sub-tarefas idênticas. Cada sub-tarefa é enviada para um servidor de índice diferente. A disciplina de filas em cada servidor de índice é "Primeiro a Chegar, Primeiro a ser Servido" (*First-Come First-Served* (FCFS)). Quando uma sub-tarefa termina a execução, irá esperar até que todas as outras sub-tarefas terminem (i.e., *join*). Apenas neste momento a tarefa completa a execução e sai da rede. Este comportamento imita o paralelismo no processamento de consultas pelos servidores de índice e a sincronização introduzida no *broker* para combinar os resultados parciais. A Análise do Valor Médio (*Mean Value Analysis* (MVA)) [43] oferece uma solução eficiente para redes de filas com solução do tipo forma-produto. Em particular, MVA pode ser usada para produzir estimativas de desempenho para cada servidor de índice individual. Entretanto, a característica *fork-join* viola as suposições requeridas pela solução MVA exata. Assim, usamos técnicas MVA aproximadas [37] para resolver o modelo completo de busca na Web.

Consultas típicas podem ter demandas heterogêneas pela CPU e disco dos servidores de índice dependendo do número de termos que elas contém. Além disso, devido à localidade de referência nos termos das consultas que chegam no mecanismo de busca, um servidor de índice pode encontrar algumas ou todas as listas invertidas no *cache* do disco (em memória). Assim, algumas consultas podem não recuperar nenhum dado do disco. De fato, durante nossos experimentos de validação, encontramos um número não desprezível de tais consultas.

A fim de capturar o impacto das demandas heterogêneas por recursos, refinamos nosso modelo do servidor de índice como se segue. Primeiro, modelamos duas classes de consultas separadamente: a classe *small* (consultas com no máximo 2 termos) e a classe *large* (consultas com mais que 2 termos). Segundo, modelamos separadamente as demandas médias pela CPU e disco, bem como a probabilidade de acerto total no *cache* do disco (i.e., todas as listas invertidas são encontradas no *cache* do disco), num servidor de índice para cada classe de consultas. As consultas da classe *large* têm maiores demandas pela CPU e disco de um servidor de índice do que as consultas da classe *small*, enquanto as consultas da classe *small* têm maiores probabilidades de acerto total no *cache* do disco de um servidor de índice do que as consultas da classe *large*. Além disso, assumimos que as consultas

podem ter demandas diferentes pela CPU dependendo se recuperam algum dado do disco. Finalmente, dado que as consultas são processadas seqüencialmente por cada servidor de índice, não existe fila em nenhum recurso (nem na CPU nem no disco) de um servidor de índice.

Finalmente, o tempo gasto no *broker* consiste principalmente de processamento local para enviar a consulta para todos os servidores de índice, receber os resultados parciais de todos os servidores, e mesclar os resultados parciais recebidos, o que depende do número de servidores no *cluster*. Observamos que o tempo de processamento no *broker* é relativamente baixo comparado ao tempo de resposta do sistema. Existem duas razões fundamentais para isto. Primeiro, a operação do *broker* é realizada inteiramente usando a memória principal, assim demandando apenas o tempo da CPU ao contrário da operação de um servidor de índice que é composta por demandas pela CPU e pelo disco. Segundo, todas as tarefas que o *broker* executa são tarefas relativamente simples que não tomam muito tempo da CPU. Deve-se notar que o *broker* não tem que fazer computações para gerar métricas de relevância para os documentos e não tem que executar operações algébricas, à exceção de comparar identificadores de documentos. A Tabela 1 apresenta os parâmetros de entrada do sistema e da carga de trabalho bem como os parâmetros de saída do nosso modelo. O tempo médio de residência de uma consulta num recurso é definido como a soma entre o tempo médio de espera e o tempo médio de serviço sobre todas as visitas ao recurso.

Modelo do Servidor de Índice

Esta seção deriva o tempo médio de residência de uma consulta num servidor de índice. Considerando que as consultas são processadas uma de cada vez por cada servidor de índice, não existe fila em nenhum recurso (nem na CPU nem no disco) de um servidor de índice. Assim, introduzimos aqui uma abstração para o servidor de índice como um único centro de serviço, cujo tempo médio de serviço para a classe r de consultas é dado por:

$$S_r^{server} = hit_r D_{cpu_{hit},r}^{server} + (1 - hit_r)(D_{cpu_{miss},r}^{server} + D_{disk,r}^{server}) \quad (1)$$

O tempo médio de serviço de uma consulta num servidor de índice é estimado como a média ponderada do tempo de serviço para cada classe, com os pesos dados pelas vazões relativas:

$$S^{server} = \sum_{r=1}^{R} \frac{\lambda_r}{\lambda} S_r^{server} \quad (2)$$

Tabela 1: Parâmetros de entrada e saída do nosso modelo.

Entradas	Descrição
p	Número de servidores de índice
R	Número de classes de consultas
λ_r	Taxa de chegada de consultas da classe r $(r = 1 \ldots R)$
S_p^{broker}	Tempo médio de serviço de uma consulta no *broker* para um *cluster* com p servidores de índice
$D_{cpu_{hit},r}^{server}$	Demanda média pela CPU num servidor de índice para a classe r de consultas que encontram todas as listas invertidas no *cache* do disco
$D_{cpu_{miss},r}^{server}$	Demanda média pela CPU num servidor de índice para a classe r de consultas que recuperam dados do disco
$D_{disk,r}^{server}$	Demanda média pelo disco num servidor de índice para a classe r de consultas
hit_r	Probabilidade de uma classe r de consultas encontrar todas as listas invertidas no *cache* do disco

Saídas	Descrição
R^{system}	Tempo médio de resposta do sistema
$R^{cluster}$	Tempo médio de residência de uma consulta no subsistema de servidores de índice
R_p^{broker}	Tempo médio de residência de uma consulta no *broker* para um *cluster* com p servidores de índice
R^{server}	Tempo médio de residência de uma consulta num servidor de índice
R_r^{server}	Tempo médio de residência num servidor de índice para a classe r de consultas
S^{server}	Tempo médio de serviço de uma consulta num servidor de índice
S_r^{server}	Tempo médio de serviço num servidor de índice para a classe r de consultas
U^{server}	Utilização total de recursos de um servidor de índice

Usando a Lei de Little [37], a utilização do centro de serviço representando um servidor de índice pela classe r de consultas pode ser estimado ao multiplicar S_r^{server} pela taxa de

chegada de consultas λ_r correspondente. A utilização total do servidor pode ser calculada como:

$$U^{server} = \sum_{r=1}^{R} \lambda_r S_r^{server} \tag{3}$$

Usando uma equação MVA para redes abertas [37], podemos estimar o tempo médio de residência num servidor de índice para a classe r de consultas como:

$$R_r^{server} = \frac{S_r^{server}}{1 - U^{server}} \tag{4}$$

Finalmente, o tempo médio de residência num servidor de índice é estimado como a média ponderada dos tempos de residência para cada classe como:

$$R^{server} = \sum_{r=1}^{R} \frac{\lambda_r}{\lambda} R_r^{server} \tag{5}$$

Modelo do Sistema

O tempo médio de resposta do sistema é a soma dos tempos médios de residência no *broker* e no subsistema de servidores de índice. O tempo médio de residência de uma consulta no *broker* para um *cluster* com p servidores de índice pode ser facilmente estimado usando MVA como:

$$R_p^{broker} = \frac{S_p^{broker}}{1 - \lambda S_p^{broker}} \tag{6}$$

Um limite inferior simples para o tempo médio de residência no subsistema *fork-join* é obtido ao ignorar os atrasos de sincronização e considerando que o tempo médio de residência no subsistema é igual ao tempo médio de residência num servidor de índice (veja Equação 5). Entretanto, à medida que o número de servidores de índice cresce, esperamos um desvio significante deste limite inferior devido aos custos adicionais de sincronização.

Um número de aproximações para modelos de fila com sincronização *fork-join*, com vários graus de complexidade e precisão, estão disponíveis na literatura (veja [3, 22, 25, 34, 40, 54, 57–60, 64] e referências nelas citadas). Nelson e Tantawi [40] propõem um limite superior muito simples para o tempo médio de resposta em redes de filas *fork-join*, que depende apenas do número de servidores de índice p, e do tempo médio de serviço de uma consulta e utilização do servidor. Os dois últimos parâmetros são facilmente estimados usando as Equações 2 e 3. Dado o p-ésimo número harmônico $H_p = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{p}$, o limite superior é dado por:

$$R^{cluster} \quad \leq \quad H_p \frac{S^{server}}{1 - U^{server}} \tag{7}$$

Combinando as Equações 5, 6, e 7, obtemos os seguintes limites para o tempo médio de resposta do sistema para nosso mecanismo de busca na Web:

$$R^{server} + R_p^{broker} \leq R^{system} \leq H_p \frac{S^{server}}{1 - U^{server}} + R_p^{broker} \quad (8)$$

Validação do Sistema

Uma série de experimentos de validação foram executados num ambiente dedicado consistindo de um *cluster* de 8 servidores de índice e um único *broker*. Usamos uma coleção composta por 10 milhões de páginas coletadas pelo mecanismo de busca TodoBR da Web brasileira em 2003. A coleção foi uniformemente distribuída através de 8 servidores de índice, resultando numa subcoleção local de tamanho $b = 1.25$ milhões de páginas. O conjunto de consultas usado em nossos testes é composto por $85,604$ consultas numa hora de carga alta da série de consultas TodoBR dobrada, que tem características de tráfego comumente encontradas em outras cargas de mecanismos de busca no mundo.

Os valores dos parâmetros de entrada do modelo foram facilmente obtidos ao recuperar estatísticas coletadas pelo sistema operacional Linux e disponibilizadas no sistema de arquivos */proc*, durante o experimento. A Tabela 2 apresenta os valores dos parâmetros de entrada do modelo obtidos em nossos experimentos. As demandas pelos recursos dos servidores de índice e as probabilidades de acerto são médias para todos os servidores.

A Figura 9 mostra o tempo médio de residência de uma consulta num servidor de índice, calculada a média sobre todos os servidores de índice, em função da taxa de chegada de consultas. A curva "estimado" representa os resultados obtidos com a Equação 5, ao passo que a curva "medido" contém a média dos tempos de residência medidos em todos os 8 servidores de índice. Como mostrado na Figura 9, nosso modelo captura razoavelmente bem o desempenho médio de um servidor de índice típico. Para uma taxa de chegada de 28 consultas/segundo, o servidor de índice está se aproximando do seu ponto de saturação. Para esta carga, o erro introduzido pelo nosso modelo é de apenas 25%, razoavelmente pequeno para estimativas de tempo de resposta [37].

As Figuras 10 e 11 mostram resultados experimentais bem como o limite inferior e superior, estimados pela Equação 8, em função da taxa de chegada e do número de servidores de índice, respectivamente. Na Figura 11, a fim de fazer uma comparação justa, mantivemos o tamanho da subcoleção b fixa variando apenas o número total de servidores p e, indiretamente, o tamanho da coleção total $n = pb$.

Tabela 2: Valores dos parâmetros de entrada do modelo.

Parâmetro	Valor	
	Classe *small*	Classe *large*
p	($\leq$) 8 servidores	
b	1.25 milhões de páginas	
S_p^{broker}, $p = 2$	0.33 ms	
S_p^{broker}, $p = 4$	0.39 ms	
S_p^{broker}, $p = 8$	0.52 ms	
$D_{cpu_{hit},r}^{server}$	87.72 ms	12.92 ms
$D_{cpu_{miss},r}^{server}$	6.36 ms	18.71 ms
$D_{disk,r}^{server}$	19.47 ms	46.12 ms
hit_r	0.20	0.11
λ_r/λ	0.73	0.27

Como as Figuras 10 e 11 mostram, o limite inferior é uma boa aproximação para sistemas com um número pequeno de servidores de índice e/ou servidores pouco sobrecarregados. Entretanto, à medida que a carga ou o número de servidores de índice aumenta, o tempo medido de resposta do sistema desvia significativamente do limite inferior devido ao custo adicional de sincronização. Isto contrasta com trabalhos anteriores que desconsideram o desbalanceamento nos tempos de serviço das consultas entre os servidores de índice homogêneos [23]. De fato, vemos que o tempo medido de resposta do sistema aproxima-se do limite superior para números grandes de servidores de índice e cargas pesadas. Em particular, para $p = 8$ servidores de índice e uma taxa de chegada de $\lambda = 28$ consultas/segundo, o erro na aproximação é de apenas 7%. Portanto, o limite-superior provê um aproximação simples de computar e razoavelmente precisa do tempo de resposta do sistema de mecanismos de busca na Web realísticos e tipicamente sobre carregados.

Um Exemplo de Aplicabilidade do Modelo

Nesta seção, discutimos como usar nosso modelo para o planejamento de capacidade de mecanismos de busca na Web de larga escala. Assumimos uma coleção de 20 bilhões de páginas, que é o tamanho dos índices do Google e Yahoo levado a público [24]. Assumimos

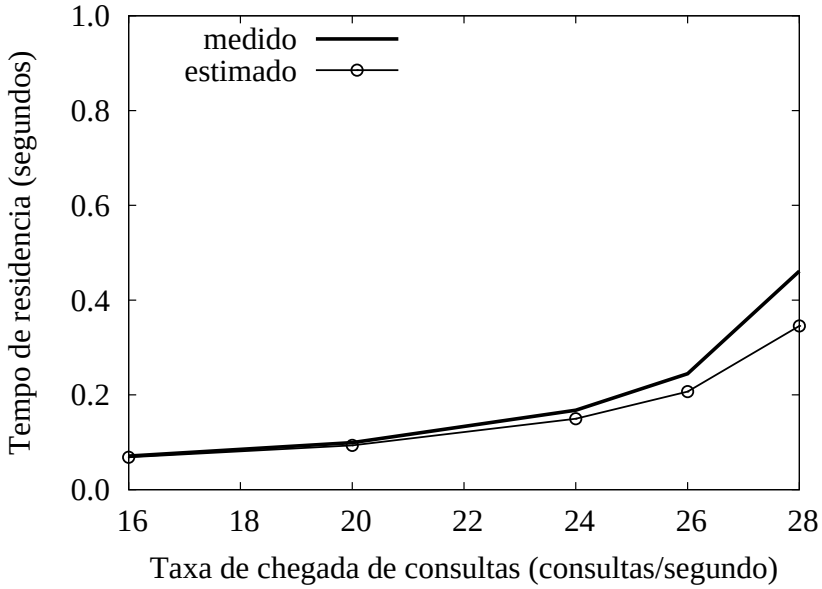


Figura 9: Tempo médio de residência de uma consulta num servidor de índice em função da taxa de chegada de consultas ($p = 8$).

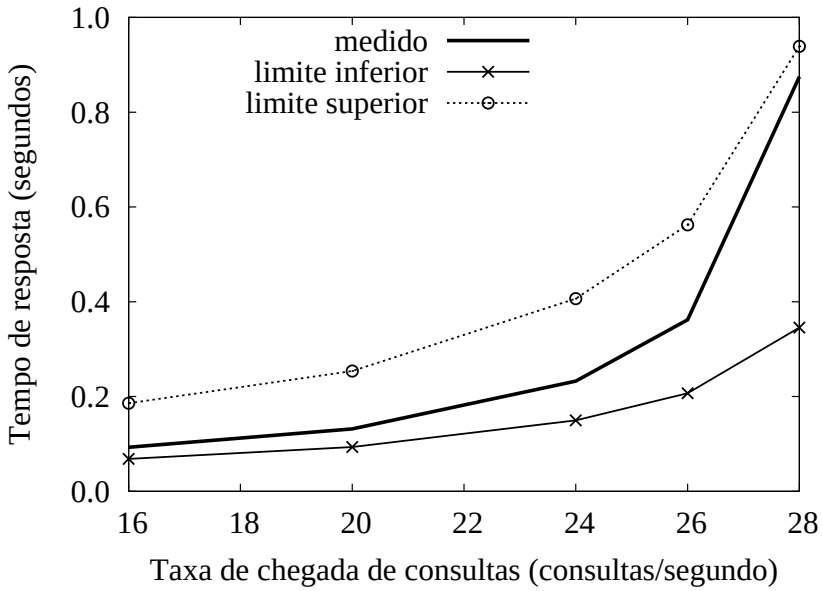


Figura 10: Tempo médio de resposta do sistema em função da taxa de chegada de consultas ($p = 8$).

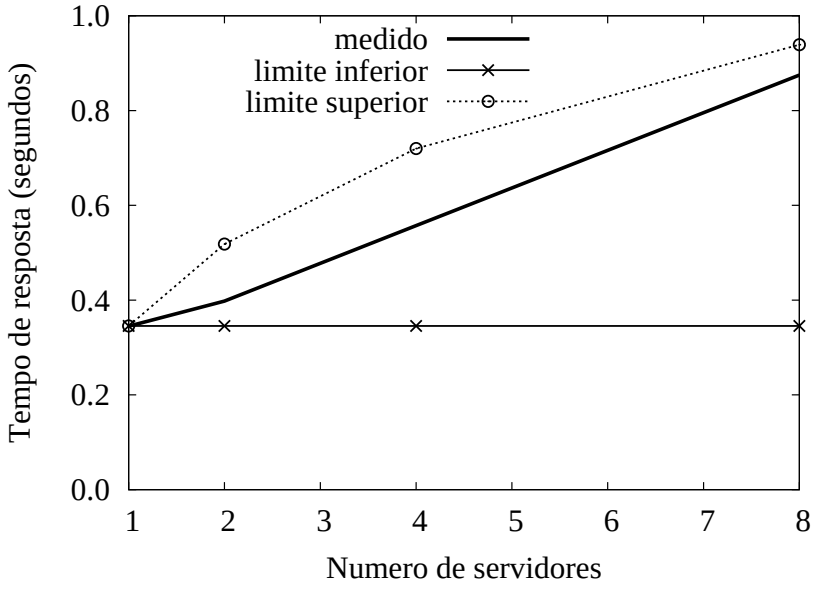


Figura 11: Tempo médio de resposta do sistema em função do número de servidores de índice p ($\lambda = 28$ consultas/segundo).

também que cada servidor de índice armazena uma subcoleção de tamanho $b = 10$ milhões de páginas — a maior coleção que temos disponível para experimentação, o que requer $2,000$ servidores de índice para hospedar a coleção inteira. Para obter os parâmetros do modelo do servidor de índice, executamos experimentos num único servidor de índice. Usamos uma coleção composta por aproximadamente 10 milhões de páginas coletadas pelo mecanismo de busca TodoBR da Web brasileira em 2003. Baseados nestes experimentos, determinamos os valores dos parâmetros do modelo do servidor de índice, tais como as demandas médias pela CPU e disco, e a probabilidade de acerto total no *cache* do disco de um servidor de índice para cada classe de consultas. Para obter o tempo médio de serviço de uma consulta no *broker* em função de $2,000$ servidores de índice (S_p^{broker}, $p = 2,000$), ajustamos uma linha reta para os valores de S_p^{broker} ($p = 2, 4, 8$) estimados durante nossos experimentos de validação (veja a Tabela 2). Encontramos uma aproximação bastante precisa dada por $S_p^{broker} = 3.18e^{-5}p + 0.000265$, a qual usamos para derivar $S_p^{broker} = 15.96$ milisegundos para $p = 2,000$. A Tabela 3 apresenta os novos valores dos parâmetros de entrada do modelo usados em nosso exemplo.

Uma vez que os parâmetros do modelo são computados, pode-se aplicar o modelo para derivar as métricas de desempenho de interesse. Considere que o gerente do mecanismo

Tabela 3: Novos valores dos parâmetros de entrada do modelo usados em nosso exemplo.

Parâmetro	Valor	
	Classe *small*	Classe *large*
p	2,000 servidores	
b	10 milhões de páginas	
S_p^{broker}	15.96 ms	
$D_{cpu_{hit},r}^{server}$	27.13 ms	72.33 ms
$D_{cpu_{miss},r}^{server}$	28.44 ms	92.24 ms
$D_{disk,r}^{server}$	41.78 ms	111.39 ms
hit_r	0.30	0.18
λ_r/λ	0.73	0.27

de busca na Web quer garantir que o tempo médio de resposta do sistema não excederá 300 milisegundos. Considere também que o gerente tem o objetivo de suportar uma taxa de chegada de consultas de 1,000 consultas/segundo. Usando o modelo com os parâmetros descritos na Tabela 3, calculamos o limite superior para o tempo médio de resposta do sistema em função da taxa de chegada de consultas. A Figura 12 mostra o limite superior para o tempo médio de resposta do sistema em função da taxa de chegada de consultas. A curva "sistema de base" representa os resultados derivados pelo modelo com os parâmetros estimados experimentalmente (veja a Tabela 3). Os resultados para a curva "sistema de base" indicam que, mesmo em baixas taxas, o limite superior para o tempo médio de resposta do sistema excede o limite definido pela gerência do mecanismo de busca na Web. Queremos avaliar que tipo de otimização nos recursos das máquinas pode reduzir o tempo médio de resposta do sistema para menos do que 300 milisegundos. Para isto, consideramos três cenários. Também na Figura 12, as curvas rotuladas de "discos 4x", "CPUs 4x", e "CPUs/discos 4x" representam o limite superior para o tempo médio de resposta do sistema para os cenários onde as CPUs são quatro vezes mais rápidas, os discos são quatro vezes mais rápidos, e as CPUs e discos são ambos quatro vezes mais rápidos, respectivamente. Estes são os três cenários analisados a seguir.

Cenário 1 - Discos são quatro vezes mais rápidos.

No primeiro cenário, queremos avaliar o impacto no tempo de resposta de uma consulta

de discos que são quatro vezes mais rápidos do que aqueles em uso. Isto é refletido no parâmetro do modelo ao dividir a demanda pelo disco por um fator de quatro. A solução do modelo com os novos parâmetros produz o novo limite superior para o tempo médio de resposta do sistema. Os resultados mostram que o limite superior para o tempo médio de resposta do sistema diminui significativamente, com ganhos de 7.49 vezes aproximadamente sobre o sistema de base quando o mesmo está se aproximando do seu ponto de saturação ($\lambda = 10$ consultas/segundo). Também observamos que a taxa de chegada suportada ($\lambda = 15$ consultas/segundo) aumenta 1.50 vezes aproximadamente quando comparada à taxa de chegada suportada pelo sistema de base ($\lambda = 10$ consultas/segundo). Entretanto, o limite superior ainda excede o limite definido mesmo em cargas leves.

Cenário 2 - CPUs são quatro vezes mais rápidas.

No segundo cenário, queremos avaliar o impacto no tempo de resposta de uma consulta de CPUs que são quatro vezes mais rápidas. A maneira de modelar as novas CPUs é dividir a demanda pela CPU por um fator de quatro. Usando o modelo, calculamos o novo limite superior para o tempo médio de resposta do sistema. Os resultados indicam que a configuração com CPUs mais rápidas supera ligeiramente o desempenho da configuração com discos mais rápidos, com ganhos de 8.26 vezes aproximadamente quando o sistema de base está se aproximando da saturação. Também observamos que a taxa de chegada suportada ($\lambda = 16$ consultas/segundo) aumenta 1.60 vezes aproximadamente quando comparada à taxa de chegada suportada pelo sistema de base ($\lambda = 10$ consultas/segundo). Entretanto, o limite superior ainda excede o limite definido mesmo em cargas leves (como no Cenário 1).

Cenário 3 - CPUs e discos são ambos quatro vezes mais rápidos.

No terceiro cenário, queremos verificar o impacto no tempo de resposta de uma consulta de CPUs e discos que são ambos quatro vezes mais rápidos. Para modelar os novos recursos, dividimos as demandas pela CPU e pelo disco por quatro. A solução do modelo com os novos parâmetros produz o novo limite superior para o tempo médio de resposta do sistema. Os resultados indicam que o tempo médio de resposta do sistema é menor do que 297 milisegundos numa taxa de chegada de 14 consultas/segundo, o que satisfaz o objetivo de nível de serviço para o mecanismo de busca na Web de 300 milisegundos por consulta. Os resultados também mostram uma redução notável no limite superior para o tempo médio de resposta do sistema, com ganhos de 35.90 vezes aproximadamente sobre o sistema de base quando o mesmo está se aproximando do seu ponto de saturação

($\lambda = 10$ consultas/segundo). Além disso, observamos um aumento expressivo na taxa de chegada suportada ($\lambda = 42$ consultas/segundo), com ganhos de 4.20 vezes aproximadamente sobre o sistema de base ($\lambda = 10$ consultas/segundo).

Ainda temos que satisfazer o objetivo de suportar uma taxa de chegada de consultas de $1,000$ consultas/segundo. Para suportar uma taxa de chegada de consultas mais elevada, o *cluster* de servidores de índice é usualmente replicado [14,18,45]. Replicação envolve custos adicionais relativamente pequenos, podendo-se esperar ganhos aproximadamente lineares na taxa de chegada de consultas suportada em função do número de sistemas replicados. O objetivo de suportar uma taxa de chegada de consultas de $1,000$ consultas/segundo pode ser atingido ao criar 72 réplicas do *cluster* de $2,000$ servidores de índice, cada réplica suportando uma taxa de chegada de 14 consultas/segundo e garantindo um tempo de resposta de 297 milisegundos. Portanto, nosso modelo indica que um *cluster* composto por $144,000$ servidores de índice (72 réplicas do *cluster* $\times$ $2,000$ servidores de índice num *cluster*) atingiria o desempenho desejado. Algumas especulações sugerem que mecanismos de busca na Web de larga escala podem de fato adotar *clusters* com vários milhares de máquinas, mas, até onde sabemos, não existem dados disponíveis publicamente para suportar esta informação. Se cada servidor de índice manipular uma subcoleção maior, o número total de servidores num *cluster* pode ser menor. Assim, deve-se ser cauteloso antes de extrapolar nossos resultados ilustrativos para um *cluster* arbitrário de qualquer mecanismo de busca na Web. Neste caso, para outras coleções de documentos e para outras máquinas, os parâmetros do nosso modelo podem ser estimados experimentalmente.

Este exemplo mostra como um modelo de planejamento de capacidade razoavelmente preciso provê uma ferramenta muito útil para a gerência apropriada de serviços modernos de busca na Web baseados em *cluster*.

Conclusões

Nesta tese, provemos um arcabouço para o projeto e análise da infra-estrutura de mecanismos de busca na Web. Neste arcabouço (i) investigamos e analisamos o desbalanceamento entre os servidores de índice homogêneos num *cluster* para processamento paralelo de consultas e (ii) propusemos um modelo de planejamento de capacidade para mecanismos de busca na Web.

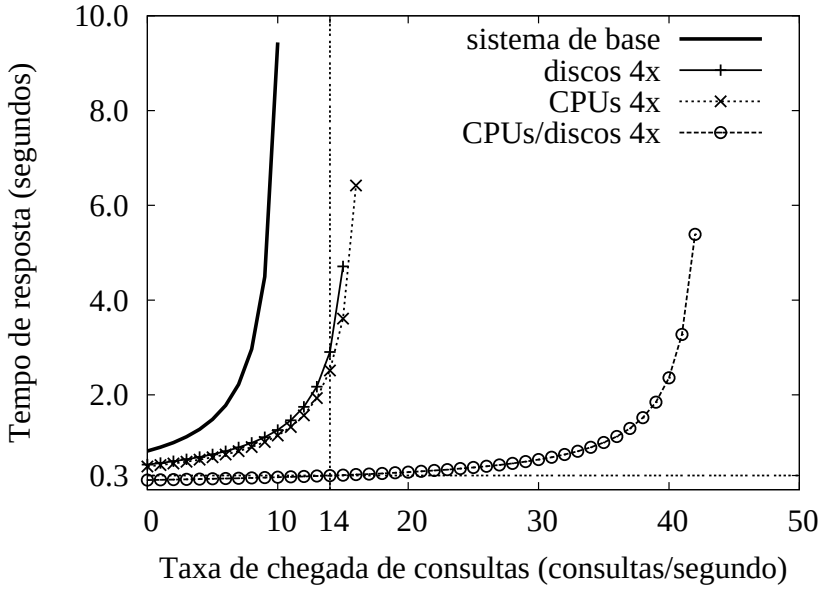


Figura 12: Limite superior para o tempo médio de resposta do sistema em função da taxa de chegada de consultas derivado em nosso exemplo.

Analisando o Desbalanceamento entre os Servidores de Índice Homogêneos

Como resultado da nossa análise da questão do desbalanceamento, verificamos que o cenário idealizado, que supõe tempos de serviço balanceados como uma conseqüência da distribuição uniforme de dados entre os servidores de índice homogêneos, é improvável de acontecer na prática. Esta é uma contribuição importante porque contradiz uma suposição usual de tempos de serviço balanceados feita por muitos modelos teóricos na literatura para simplificar sua tarefa de modelagem [20, 23, 44]. Nossas descobertas são derivadas de uma análise experimental detalhada usando um sistema de recuperação de informação e dados reais obtidos de um mecanismo de busca real. Além de verificar a presença de um certo nível de desbalanceamento entre os servidores de índice homogêneos, identificamos e caracterizamos as principais fontes deste desbalanceamento inesperado.

O principal fator para o desbalanceamento é o uso do *cache* do disco nos diferentes servidores de índice. Verificamos que o desbalanceamento para cada consulta aumenta com o número de servidores de índice que recuperam os documentos necessários do *cache* do disco. Por um lado, o pior caso para o desbalanceamento é alcançado quando um único servidor

de índice realmente acessa o disco para recuperar documentos enquanto todos os servidores restantes usam o *cache* do disco para a mesma consulta. Por outro lado, o melhor caso para evitar desbalanceamento é alcançado quando todos os servidores de índice operam na região do *cache*, levando assim a tempos de acesso a disco relativamente menores e similares através do *cluster* de servidores. Outra fonte de desbalanceamento identificada é o tamanho da memória principal dos servidores de índice homogêneos, que afeta a disponibilidade de recursos para o uso do *cache* do disco nos servidores. Verificamos que o desbalanceamento médio diminui enquanto o tamanho da memória principal aumenta. Outra fonte de desbalanceamento identificada é o número de servidores de índice no *cluster*. Verificamos que, para um tamanho fixo de memória principal, o desbalanceamento médio nos tempos de serviço entre os servidores de índice aumenta com o número de servidores no *cluster*.

Um Modelo de Planejamento de Capacidade para Mecanismos de Busca na Web

Também, propusemos um modelo de planejamento de capacidade para mecanismos de busca na Web que considera o desbalanceamento nos tempos de serviço de uma consulta entre os servidores de índice homogêneos. Nosso modelo, baseado na teoria de filas, é simples e razoavelmente preciso. Para ajustar o modelo, executamos experimentos num *cluster* pequeno de servidores de índice. Uma vez ajustado, comparamos as predições do nosso modelo com os resultados medidos empiricamente e encontramos grande concordância. Até mesmo no ponto de saturação, as predições do nosso modelo foram razoavelmente precisas. Também, ilustramos como aplicar nosso modelo para predizer o tempo de resposta de uma consulta ao adotar CPUs e discos mais rápidos do que aqueles em uso. Consideramos um cenário realístico, onde uma coleção de 20 bilhões de documentos é distribuída através de $2,000$ servidores de índice. Em nosso exemplo, mostramos que o gerente de um mecanismo de busca pode rapidamente predizer limites superiores para o tempo de resposta de uma consulta sem ter que executar experimentos.

Dada a complexidade da manutenção de mecanismos de busca, e a simplicidade e razoável precisão do nosso modelo, acreditamos que nosso modelo pode ser muito útil na prática.

Trabalho Futuro

Uma direção para trabalho futuro é estender nosso modelo de planejamento de capacidade para suportar múltiplas *threads* de processamento nos servidores de índice. Outra direção para pesquisa é melhorar nosso modelo para estimar a função de distribuição do tempo de resposta de uma consulta. A partir desta distribuição, pode-se encontrar seus percentis. Esta solução é útil se o gerente de um mecanismo de busca na Web exigir que o q-percentil do tempo de resposta esperado de uma consulta seja menor ou igual a um limite definido.

Uma outra direção para pesquisa futura é modelar *caching* dos resultados das consultas — que permite ao mecanismo de busca responder consultas repetidas recentemente a um custo muito baixo desde que não é necessário processar estas consultas — e *caching* das listas invertidas dos termos das consultas — que melhora o tempo de processamento de novas consultas que incluem pelo menos um termo cuja lista está guardada em memória [48]. Uma outra direção para pesquisa é identificar e analisar as razões para o uso do *cache* do disco, e eventualmente modelar a probabilidade de acerto no *cache* do disco. Alguns elementos que afetam o uso do *cache* do disco são a localidade de referência temporal das consultas e termos das consultas no tráfego de entrada, a localidade de referência espacial das listas invertidas dos termos das consultas no disco, e o tamanho da memória principal.

Uma outra direção para trabalho futuro é verificar, através de simulação, a precisão das predições de nosso modelo para mecanismos de busca na Web de larga escala, que contam com *clusters* com milhares de servidores de índice para suportar coleções compostas por bilhões de documentos. Seria importante considerar um *cluster* composto por p servidores de índice, tal que p é grande o suficiente para armazenar o índice de uma coleção de 20 bilhões de páginas, que é o tamanho dos índices do Google e Yahoo levado a público [24]. É difícil obter grandes coleções de documentos usadas pelos mecanismos de busca na Web para gerar uma página de resposta pra cada consulta recebida. A razão é que o conjunto de documentos coletados é visto como informação estratégica e proprietária pelos principais operadores de busca na Web. De fato, durante o curso desta tese, tivemos acesso apenas a uma coleção de documentos composta por aproximadamente 10 milhões de páginas Web coletadas pelo mecanismo de busca TodoBR da Web brasileira em 2003. Uma solução seria gerar uma coleção sintética de documentos grande, a partir de distribuições que são baseadas em estatísticas provenientes de conjuntos de dados reais.

Uma outra direção para pesquisa futura é desenvolver uma abordagem para encontrar a arquitetura ótima para um mecanismo de busca na Web em termos de custo, que combine as estratégias de particionamento e replicação a fim de satisfazer requisitos operacionais

para o tempo de resposta de uma consulta, vazão de consultas, e utilização do servidor. O custo é dado pelo número de servidores de índice na arquitetura. Portanto, o objetivo seria satisfazer os três requisitos operacionais com o menor número de servidores de índice. Este objetivo pode ser formalizado ao minimizar a função de custo:

$$c = p \times r \tag{9}$$

ao mesmo tempo satisfazendo os seguintes requisitos operacionais:

$$\begin{aligned} f_t(p, r, X) &\leq T \\ f_u(p, r, X) &\leq U \end{aligned} \tag{10}$$

onde r é o número de replicações (de um *cluster* de servidores de índice); p é o número de partições do índice (através dos servidores de índice num *cluster*); T é o requisito para o tempo de resposta; U é o requisito para a utilização; X é o requisito para a vazão; e $f_t(p, r, X)$ e $f_u(p, r, X)$ calculam tempo de resposta e utilização, respectivamente, para uma dada vazão X e uma dada arquitetura onde o índice é particionado em p divisões e replicado r vezes.

Design and Analysis of Web Search Systems

Contents

List of Figures

List of Tables

h

Glossary

Acronyms

CCDF: Complementary Cumulative Distribution Function, a function complementary to the Cumulative Distribution Function.

CDF: Cumulative Distribution Function, a function that completely describes the probability distribution of a random variable.

FCFS: First-Come First-Served, a queueing discipline in which requests are served in the order of arrival at a queue.

PDF: Probability Density Function, a function that represents a probability distribution in terms of integrals.

PMF: Probability Mass Function, a function that gives the probability that a discrete random variable is exactly equal to some value. A probability mass function differs from a probability density function in that the values of the latter, defined only for continuous random variables, are not probabilities; rather, its integral over a set of possible values of the random variable is a probability.

MVA: Mean Value Analysis, an efficient algorithm to solve product-form queueing networks and obtain mean values for queue lengths and response times.

Parameters

α: parameter of a Zipf's distribution.

b: size of the subcollection stored by an index server.

$D^{server}_{cpu_{hit},r}$: average demand for CPU at an index server for class r queries that find all inverted lists in the disk cache.

$D^{server}_{cpu_{miss},r}$: average demand for CPU at an index server for class r queries that retrieve data from disk.

$D^{server}_{disk,r}$: average disk demand at an index server for class r queries.

hit_r: probability of a class r query finding all inverted lists in the disk cache.

idf_i: inverse document frequency of the term k_i.

λ: query arrival rate.

λ_r: query arrival rate for class r ($r = 1 \ldots R$).

n: size of the whole document collection.

p: number of index servers in a cluster.

R: number of query classes.

R^{broker}_p: average query residence time at the broker for a cluster with p index servers.

$R^{cluster}$: average query residence time at the index server subsystem.

R^{server}: average query residence time at an index server.

R^{server}_r: average residence time for class r queries at an index server.

R^{system}: average query system response time.

S^{broker}_p: average query service time at the broker for a cluster with p index servers.

S^{server}: average query service time at an index server.

S^{server}_r: average service time for class r queries at an index server.

$tf_{i,j}$: number of times the term k_i occurs in document d_j.

U^{server}: total resource utilization of an index server.

Chapter 1

Introduction

1.1 Motivation

The World Wide Web dates from the end of the 1980s [17] and no one could have imagined its current impact. The boom in the use of the Web and its exponential growth are now well known. Just the amount of textual data available is estimated to be in the order of hundreds of terabytes. In addition, other media, such as images, audio, and video, are also available. This triggers the need for efficient tools to manage, retrieve, and filter information from this huge and diversified database.

Search engines have become an essential and popular tool for dealing with the huge amount of information found on the Web. A survey by iProspect [30] reveals that 35.1% of the Internet users use search engines at least once a day, 21.2% use search engines four or more times a day, and 22.7% use search engines multiple times a week, which indicates that search engine usage is a popular type of online activity.

Further, search engines are currently a critical component of Web economy. The tremendous success of keyword targeted advertising has fuelled Web economy to new heights. In fact, in 2004 American companies spent between 9 and 10 billion dollars in online advertising to promote their products and services [29]. Around $40 - 50\%$ of this amount is estimated to have been spent on search advertising. According to JupiterResearch projections, by 2010, search advertising alone will represent a market of 18.9 billion [29]. Additionally, search engines are expected to play a major role in the context of corporate search, i.e., installing search engines in the Intranets of large companies to sift through the vast amounts of data produced internally. Fairly recent announcements of new products by major players, such as Google [26] and Yahoo [63], indicate a rising interest in corporate

search. In this case, the interest of the corporation is on compiling and organizing the information it generates regarding its own business, using information on the business as a de facto asset.

Web search engines require a huge amount of computational resources to handle the incoming query traffic, which is often characterized by high peak requirements. Further, the fact that the number of documents available on the Web keeps growing consistently—there are now at least 20 billion documents in the Web [24]—makes the problem even more challenging. To cope with these requirements, modern Web search engines rely on clusters of server machines for query processing [14, 18, 45].

The architecture of a typical Web search engine is composed of clusters of index servers, with the documents partitioned among them (each index server stores a part of the document collection and an index for it). This architecture is usually referred to as *document partitioning* and is preferred because it simplifies maintenance, simplifies the generation of the index (which can be done locally), and degrades gracefully (because the failure of an index server does not prevent any query from being answered, though the final answer set might not contain all the relevant documents in the collection). The cluster also includes a broker that communicates with the various index servers.

A new user query reaches the search engine through the broker, which sends a copy of it to each index server for local processing. Typically, each index server returns its top 10 ranked documents to the broker, which runs a merge in place to determine the final 10 answers to be sent to the user.

The processing of a query can be split into two consecutive major phases [14]. A first phase which consists of retrieving references to the documents that contain all query terms and ranking them according to some relevance metric (usually done by the index servers). A second phase which consists of taking the top ranked answers, typically 10, and generating snippets, title, and URL information for each of them (usually done by a cluster of document servers, each one holding a part of the document collection). While this second phase has roughly constant cost, independently of the size of the document collection, the first phase has a cost that increases with the size of the collection. Therefore, the performance of the first phase is crucial for maintaining the scalability of modern search engines that deal with an ever-increasing amount of Web documents.

Given the complexity involved in designing efficient Web search engines and the key role such systems play in Internet usage nowadays, it is of utmost importance to understand the behavior of Web search engines. This is essential for the performance analysis and capacity

planning of such systems in order to allow them to properly face the ever-increasing demand users are submitting them to.

1.2 Objectives and Contributions

The objective of this thesis is to provide a performance framework for the design and analysis of the infrastructure of Web search engines. Our goal is to have a simple and reasonably accurate tool that can answer capacity planning questions such as:

(i) Given a collection composed of n documents distributed over p machines, what kind of average query response time guarantees one can expect?

(ii) What kind of optimization in machine resources might yield a reduction in the average query response time to meet a service level objective defined by the management of the Web search engine?

(iii) What is the minimum number of replications of the cluster of index servers that will guarantee that, on the average, the query response time on a peak period will not exceed the threshold defined by the management of the Web search engine?

In this thesis, we analyze the performance of retrieving the most relevant documents for a given user query, i.e., the first phase of the query processing task. The major contributions of this thesis are:

- Investigation and analysis of the imbalance issue in a computational cluster for parallel query processing composed of homogeneous index servers, as presented in Chapter 4. We verify in practice a consistent imbalance per query in the service time at index servers, even though the distribution of sizes of inverted lists at the various servers are quite balanced. This is an important experimental result because our findings contradict the usual assumption of balanced service times adopted by previous theoretical models found in the literature. Moreover, we identify and fully analyze the main sources of imbalance: the use of disk cache, the size of main memory in the homogeneous index servers, and the number of index servers in the cluster.

- A capacity planning model for Web search engines that considers the imbalance in query service times among homogeneous index servers, as presented in Chapter 5. Our model, which relies on a queueing-based analytical model, is simple and yet

reasonably accurate. To set up the parameters of our model, we run experiments on a small cluster of index servers. Once the key parameters have been estimated, we verify the accuracy of the model by comparing its predictions with experimental results also produced using the small cluster of index servers. Finally, we illustrate how to use our model to predict query response time when adopting faster CPUs and disks than those in use. We consider a realistic scenario, where a collection of 20 billion documents is distributed over 2,000 index servers.

- Characterization of four query datasets with millions of queries representing the query workload of two Brazilian search engines, namely TodoBR[1] [55] and Radix [42], and two worldwide search engines, namely AllTheWeb [1] and Altavista [2], as presented in Section 5.2. A subset of 85,604 queries in a high-load hour of the TodoBR query dataset is used for a series of validation experiments of the capacity planning model for Web search engines. Our findings show that (i) the distribution of queries and of terms in queries through the four considered datasets follow a Zipf's distribution; (ii) queries containing two or fewer terms are the most frequent for all considered datasets; (iii) there is a periodic behavior on the query workload through the four query datasets; and (iv) interarrival times of queries in the TodoBR dataset follow an Exponential distribution.

1.3 Organization of this Thesis

This thesis is organized as follows. Chapter 2 introduces basic concepts related to our performance framework for the design and analysis of the infrastructure of Web search engines. Chapter 3 discusses the related work in four key areas to the capacity planning for Web search engines: index organization, system architecture, workload characterization, and performance modeling. Chapter 4 investigates and analyzes the imbalance issue in a computational cluster composed of homogeneous index servers. Chapter 5 proposes a capacity planning model for Web search engines. Our conclusions and future work follow in Chapter 6.

[1]TodoBR is a trademark of Akwan Information Technologies, which was acquired by Google in July 2005.

Chapter 2

Basic Concepts

In this chapter, we introduce basic concepts related to our performance framework for the design and analysis of the infrastructure of Web search engines. Section 2.1 presents the architecture of our system, describing the cluster of index servers in Section 2.1.1, the index organization in Section 2.1.2, the ranking strategy in Section 2.1.3, and the parallel query processing technique in Section 2.1.4. Section 2.2 presents queueing networks for system performance evaluation and prediction, including multiple-class product-form queueing networks in Section 2.2.1 and fork-join queueing networks in Section 2.2.2.

2.1 An Architecture for Web Search Engines

2.1.1 Cluster of Index Servers

Modern Web search engines typically rely on computational clusters for query processing [14, 18, 45]. Such cluster is composed of a single broker and p index servers. Figure 2.1 illustrates this architecture for a typical Web search engine. Let n be the size of the whole document collection. Assuming that the documents are uniformly distributed among the p index servers, the size b of any local subcollection is then given by $b = n/p$.

The broker receives user queries from client nodes and forwards them to the index servers, triggering the parallel query processing through the p local subcollections. Each index server searches its own local subcollection and produces a partial ranked answer. These partial ranked answers are then sent to the broker where they are combined through an in-memory merging operation. The final list of ranked documents is then sent back to the user.

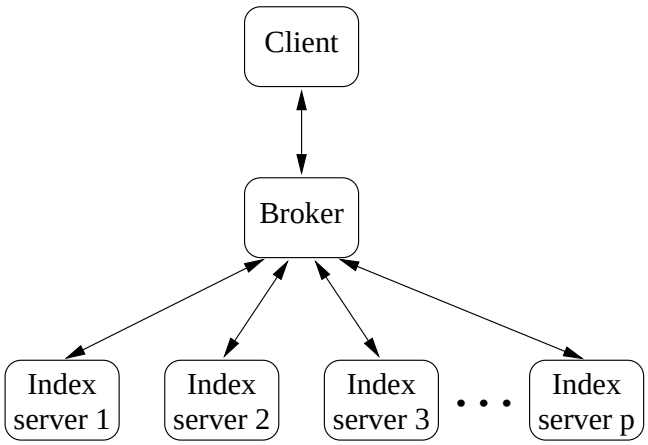


Figure 2.1: Architecture of a typical search engine.

For the moment we disregard application-level caching—neither of query results at the broker nor of lists of documents at the index servers. We model, however, the effects of disk cache at the index servers, which is natively implemented by the local operating system.

We note that the described architecture is for a single query processing cluster, which constitutes the basic unit of modern search engines. Large scale modern Web search engines basically replicates this cluster unit for supporting a higher query arrival rate [14, 18, 45].

2.1.2 Inverted Index

An inverted index [12, 62] is typically adopted as the indexing structure for each subcollection. Inverted files are useful because they can be searched based mostly on the set of distinct words in all documents of the collection. They are simple data structures that perform well when the pattern to be searched for is formed by conjunctions and disjunctions of words.

The structure of our inverted indexes is as follows. It is composed of a *vocabulary* and a *set of inverted lists*. The vocabulary is the set of all unique terms (words) in the document collection. Each term in the vocabulary is associated with an inverted list that contains an entry for each document in which the term occurs. Each entry is composed of a document identifier and the within-document frequency $tf_{i,j}$ representing the number of occurrences of term k_i within the document d_j.

The documents of the whole collection are randomly distributed among the p index servers, a policy that works fine in balancing the distributions of the sizes of the inverted

lists that compose the local inverted indexes [5, 7, 8], as further detailed in Section 4.3.1. The size of each local inverted index is $O(n/p)$, where n is the size of the whole document collection. This type of index organization, hereafter referred to as *document partitioning*, is currently the de facto standard in all major search engines [14, 45].

2.1.3 Vector Space Model

In this work, we use the standard vector space model [47] to rank the documents retrieved from the index servers. Modern search engines also adopt link-based ranking, such as PageRank [19], combined with a text-based ranking, such as the vector space model, to rank documents. However, since link information is pre-computed offline as a global measure, its usage has only modest impact on performance. Note that no matter the complexity of the adopted information retrieval algorithm, our model can capture the behavior of the algorithm because of its experimental basis (further details in Chapter 5). Thus, our approach remains realistic.

In the vector space model, documents and queries are represented as vectors in a space composed of the terms in the vocabulary of the collection. With every term k_i in a document d_j is associated a weight $w_{i,j}$. In this way, a document d_j is represented as a vector of term weights $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$, where t is the total number of distinct terms in the entire document collection. Each $w_{i,j}$ weight reflects the importance of term k_i in document d_j and is usually computed as:

$$w_{i,j} = tf_{i,j} \times idf_i \tag{2.1}$$

The term frequency $tf_{i,j}$ is simply the number of times the term k_i occurs in document d_j. The *tf* factor provides one measure of how well that term describes the document contents. The inverse document frequency idf_i is a measure of the general importance of the term k_i and is usually computed as:

$$idf_i = \log \frac{N}{n_i} \tag{2.2}$$

where n_i is the number of documents in which k_i occurs and N is the total number of documents in the collection. The motivation for usage of an *idf* factor is that terms which appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one. The expression for $w_{i,j}$ is usually referred to as *term frequency-inverse document frequency* or *tf-idf* weight. Its foundations lie in the observation that a term is

more important if it occurs many times in a document and less important if it occurs in many documents in the collection.

In the vector space model, a query q is also represented as a vector of term weights $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$. For comparing a query with a document, the most commonly used measure is the cosine of the angle between the query and document vectors. The similarity between a document d_j and a query q is defined as:

$$\begin{aligned} sim(d_j, q) &= \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} \\ &= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \end{aligned} \tag{2.3}$$

where $|\vec{d_j}|$ and $|\vec{q}|$ are the norms of the document and query vectors. The factor $|\vec{q}|$ does not affect the ranking (i.e., the ordering of the documents) because it is the same for all documents. The factor $|\vec{d_j}|$ provides a normalization in the space of the documents. By computing the similarities between all the documents in a collection $\mathcal{D}$ and a given query q, we obtain an ordered list of documents, where documents more likely to satisfy the query have higher similarities.

A standard algorithm for ranking documents with the vector space model uses a set of accumulators, one accumulator for each document in a collection, and a set of inverted lists. For each query term k_i, the contribution $sim(d_j, q, k_i)$—made by the term k_i to the degree of similarity between a query q and each document d_j in the inverted list—is added to the value of the accumulator of document d_j. This contribution, called partial similarity, is given by:

$$sim(d_j, q, k_i) = w_{i,j} \times w_{i,q} = tf_{i,j} \times \log \frac{N}{n_i} \times tf_{i,q} \times \log \frac{N}{n_i} \tag{2.4}$$

The final result is composed by the documents with the highest accumulator values. A simple version of this algorithm is shown in Figure 2.2.

Using the *idf* factor implies global knowledge about the whole collection to be available at the index servers. In an architecture characterized by a strategy of local document partitioning, this could be accomplished if index servers exchange their local *idf* factors after the local index generation phase. Each index server may then derive the global *idf* factor from the set of local *idf* factors [44].

1. For each document d_j in the collection, set accumulator $A_{d_j} \leftarrow 0$.
2. For each term k_i in the query q,
   (a) Retrieve the inverted list for k_i from disk.
   (b) For each term entry $< d_j, tf_{i,j} >$ in the inverted list,
   set $A_{d_j} \leftarrow A_{d_j} + sim(d_j, q, k_i)$.
3. Divide each non-zero accumulator A_{d_j} by the document norm $\mid \vec{d_j} \mid$.
4. Identify the f highest accumulator values (where f is the number of documents to be presented to the user) and retrieve the corresponding documents.

Figure 2.2: Basic algorithm for ranking using the vector space model.

2.1.4 Parallel Query Processing

In our experiments, a client machine submits queries to the broker according to a query arrival distribution. This broker then broadcasts each query to all index servers. Once each index server receives a query, it retrieves the full inverted lists relative to the query terms, intersects these lists to produce the set of documents that contains all query terms (i.e., the conjunction of the query terms[1]), computes a relevance score for each document, and sorts them by decreasing score—this results in a partial ranked answer to be sent by each index server to the broker. Each query term k_i is processed by decreasing idf_i, i.e., by increasing order of the number n_i of documents in the whole collection containing the term k_i, thus leading to a significantly more efficient conjunction of their inverted lists. As soon as the ranking is computed, the top ranked documents at each index server are transferred to the broker machine. The broker is then responsible for combining the partial ranked answers received from the index servers through an in-memory merging operation. The final list of top ranked documents is then sent back to the client machine.

For simplicity, we adopt a single processing thread at each index server. Lu et al. [33] verify that the query arrival rate supported by the system increases with the number of threads, until either CPUs or disks are overutilized. The use of multiple processing threads at index servers is left for future work.

Notice that in this architecture, characterized by a strategy of local document partitioning, the response time of a particular query basically depends on the execution time of the

[1]Taking the conjunction of the query terms is now standard practice on the Web

slowest index server to produce the corresponding partial answer set. Therefore, the higher the imbalance in execution times of index servers, the larger tends to be the response time of a query processed by the cluster of servers. Thus, it is critically important to avoid imbalance. If the document collection is partitioned among a certain number of homogeneous index servers in a balanced way, such that all of them manage a similar amount of data when processing a query, it would be expected that execution times were also balanced. This idealized scenario of supposing balanced execution times as a direct consequence of a uniform collection distribution among index servers is indeed a usual assumption taken by theoretical models in the literature to simplify the modeling task. Nevertheless, such an idealized balance is unlikely to be found in practice as we point out in this work. Such an observation is based on the experimental analysis described in Chapter 4.

2.1.5 The Broker is not the Main Bottleneck

Observing the architecture for parallel query processing in Figure 2.1, we notice that the broker constitutes a potential bottleneck. Every query that is submitted to our cluster of index servers is processed by the broker, which has to merge the partial results produced by the various index servers. However, this merging of partial results is done all in memory and can be done quickly. As a result, the broker works at relatively low loads at all times, as we now evaluate.

To stress the broker, we implemented a simulator for the cluster with p index servers. By varying p we can stress the broker and observe how it behaves. The broker takes queries at an arrival rate λ (which we varied) and passes them to a single machine that runs our simulator. For each query, the broker sends p query requests to our simulator, one for each index server. For each query request it receives, the simulator does not process it. Instead, it sorts an answer from an array of pre-computed answers stored in main memory, and sends this answer to the broker. Thus, from the viewpoint of the broker everything goes on as if there were indeed p index servers in the cluster, except for the negligible time spent by the simulator to generate partial answers to the broker. Figure 2.3 reports the average time at the broker per query. We observe that even at high loads and with a large number of index servers, time at the broker is not affected. In fact, with 256 index servers and query arrival rates around 100 queries/second, average time at the broker per query is less than 10 milliseconds.

This is because of two fundamental reasons. First, all the work the broker does is carried out fully in main memory. Second, all the tasks the broker executes are simple

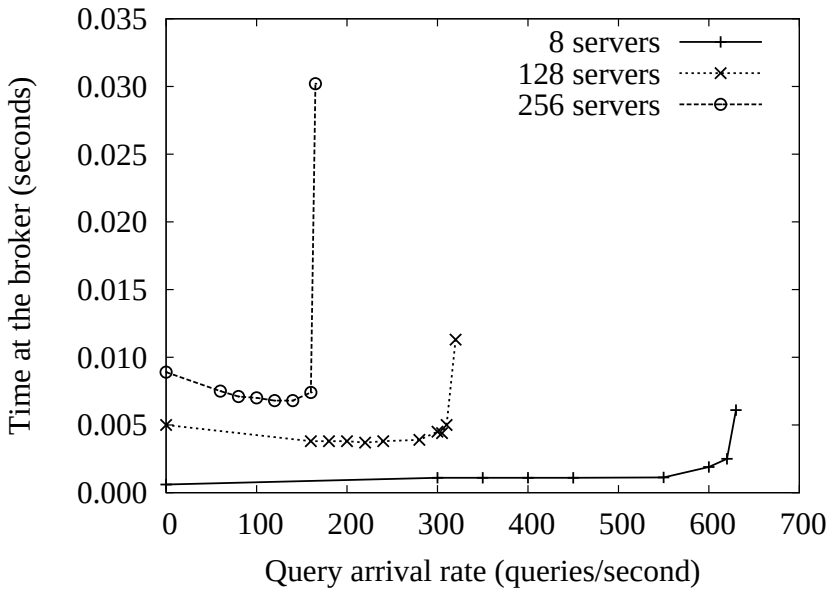


Figure 2.3: Average time (seconds) at the broker per query as a function of the query arrival rate (queries/second).

tasks that do not take much CPU time. The broker does not have to make ranking computations and does not have to execute algebraic operations (other than comparing document identifiers). Further, since it has to wait for the partial results for each query, it has plenty of free resources that can be shared with the various queries in the input stream.

Eventually the broker saturates as shown in Figure 2.3. In our case saturation is abrupt. The reason is that saturation is caused by contention at the network drive interface. With 256 index servers, for instance, each user query requires 256 *write* operations to send the queries out and another 256 *read* operations to retrieve the partial answers from the network. At a rate above 150 queries/second, the network drive interface fails to handle the load. This is not critical though because the index servers saturate at a rate one order of magnitude smaller (i.e., index servers approximate saturation at 28 queries/second as we later show in Section 5.3.2). That is, a broker whose main task is to merge partial answer sets is not the main bottleneck.

2.2 Queueing Networks

Queueing networks have been extensively applied to represent and analyze computer systems. They have proved to be a powerful and versatile tool for system performance evaluation and prediction. A *queueing network* is a network of interconnected queues that represents a computer system [36]. A *queue* in a queueing network stands for a resource (e.g., CPU, disk, network) and the queue of requests waiting to use the resource. A queue is characterized by a function $S(n)$ that represents the average service time per request when there are n requests at the queue. The number of requests n at the queue is called the *queue length*.

Not all requests that flow through the resources of a queueing network are similar in terms of the resources used and the time spent at each resource. The total workload submitted to a computer system may be broken down into several workload components, which are represented in a queueing model by a *class* of requests. Different classes may have different service demand parameters and different workload intensity parameters. The workload intensity parameters provide a measure of the load placed on a system, indicated by the number of units of work that contend for system resources. In the case of Web search engines, this measure corresponds to the number of incoming queries per second. The workload service demand parameters specify the total amount of service time required by each basic component of the workload on each resource of the system. In a Web search engine, the service time for a user query is equivalent to the amount of time needed to select the most relevant documents for this query.

Classes of requests may be classified as *open* or *closed* depending on whether the number of requests in the queueing network is unbounded or fixed, respectively. Open classes allow requests to arrive, go through the various resources, and leave the system. Closed classes are characterized by having a fixed number of requests in the queueing network. A queueing network in which all classes are open is called an open queueing network. A queueing network in which all classes are closed is called a closed queueing network. A queueing network in which some classes are open and others are closed is called a *mixed* queueing network.

A central concept in the solution and analysis of queueing networks is the state of the network [37]. The state represents a distribution of requests over classes of requests and resources. The solution of queueing networks consists of finding the long term (i.e., the *steady state*) probability of being in any particular state. This refers to the probability of taking a random snapshot (or checkpoint) of the system and finding the system in a

particular state. The analysis of queueing networks consists of evaluating a set of performance measures, such as system response time, system throughput, and server utilization. The popularity of queueing networks for system performance evaluation is due to a good balance between a relative high accuracy in the performance results and the efficiency in model analysis and evaluation. In this framework, the class of *product-form queueing networks* has played a fundamental role.

Product-form queueing networks have a simple closed-form expression of the steady-state probability distribution that allow to define efficient algorithms to evaluate average performance measures [13]. The precise characterization of the class of product-form network is not easy. The product-form solution is related to some properties of the queueing network model that are defined on the Markov process underlying the queueing model. The most famous result concerning product-form queueing networks is the BCMP theorem [15], developed by Baskett, Chandy, Muntz, and Palacios. The BCMP theorem defines the well-known class of BCMP queueing networks with product-form solution for open, closed, or mixed models with multiple-classes of customers and various service disciplines and service time distributions. The steady-state probability distribution is expressed as the product of the distributions of the single queues with appropriate parameters and, for closed networks, with a normalization constant. In the case of service centers with a *First-Come First-Served* (FCFS) service discipline, under which requests are serviced in the order in which they arrive, the service time distributions are required to be exponential with the same mean for all classes. Although all classes must have the same mean service time at any given resource, they may have different visit ratios, which allows the possibility of different service demands for each class at any given resource. In open networks, the time between successive arrivals is assumed to be exponentially distributed.

In our work, the Web search engine is represented as an open queueing network composed of the broker and the subsystem of index servers, as further discussed in Chapter 5. The broker is modeled as a product-form queueing network with a single class, an index server as a product-form queueing network with multiple-classes, and the subsystem of index servers as a fork-join queueing network.

2.2.1 Multiple-Class Open Queueing Networks

Multiple-class product-form queueing networks have efficient computational algorithms for their solution. Because of its simplicity and intuitive appeal, we adopt the MVA-based algorithm for approximate solution of open multiple-class models [37].

Consider the following notation for the multiple-class open model presented here.

- K: number of resources or service centers of the model
- R: number of classes of requests
- λ_r: arrival rate of class r
- $S_{i,r,}$: average service time of class r requests at resource i
- $V_{i,r}$: average visit ratio of class r requests at resource i
- $D_{i,r}$: average service demand of class r requests at resource i; $D_{i,r} = V_{i,r}S_{i,r}$
- $R_{i,r}$: average response time per visit of class r requests at resource i
- $R'_{i,r}$: average residence time of class r requests at resource i, i.e., the total time spent by class r requests at resource i over all visits to the resource; $R'_{i,r,} = V_{i,r}R_{i,r}$
- $\overline{n}_{i,r}$: average number of class r requests at resource i
- $\overline{n}_i$: average number of requests at resource i
- $X_{i,r}$: class r throughput at resource i
- $X_{0,r}$: class r system throughput
- R_r: class r response time

In steady-state, the throughput of class r equals its arrival rate. Thus,

$$X_{0,r} = \lambda_r \tag{2.5}$$

The application of Little's Law [37] to each resource gives

$$\overline{n}_{i,r} = X_{i,r}R_{i,r} \tag{2.6}$$

Using the Forced Flow Law [37] and Equation 2.5, the throughput of class r at resource i is

$$X_{i,r} = X_{0,r}V_{i,r} = \lambda_r V_{i,r} \tag{2.7}$$

The average residence time for the entire execution is $R'_{i,r} = V_{i,r}R_{i,r}$. Using Equation 2.7 in Equation 2.6 the average queue length per resource for each class becomes

$$\overline{n}_{i,r} = \lambda_r R'_{i,r} \tag{2.8}$$

Combining the Utilization Law [37] and the Forced Flow Law, the utilization of resource i by class r requests can be written as

$$U_{i,r} = X_{i,r}S_{i,r} = \lambda_r V_{i,r}S_{i,r} = \lambda_r D_{i,r} \tag{2.9}$$

The average time a class r request spends at a resource, from arrival until completion, has two components: the time for receiving service and the time spent in queue. The latter is equal to the time required to service requests that are currently in the resource when the request arrives. Thus,

$$\begin{aligned} R_{i,r} &= S_{i,r}(1 + \overline{n}^A_{i,r}) \\ V_{i,r}R_{i,r} &= V_{i,r}S_{i,r}(1 + \overline{n}^A_{i,r}) \\ R'_{i,r} &= D_{i,r}(1 + \overline{n}^A_{i,r}) \end{aligned} \tag{2.10}$$

where $\overline{n}^A_{i,r}$ is the average queue length at resource i seen by an arriving class r request.

Te arrival theorem [49] states that in an open product-form queueing network, a class r arriving request at service center i sees the steady-state probability distribution of the resource state, which is given by the queue length. Thus,

$$\overline{n}^A_{i,r} = \overline{n}_i \tag{2.11}$$

From Equations 2.10 and 2.11, we get

$$R'_{i,r} = D_{i,r}(1 + \overline{n}_i) \tag{2.12}$$

Substituting Equation 2.12 into Equation 2.8, yields

$$\overline{n}_{i,r} = \lambda_r D_{i,r}(1 + \overline{n}_i) = U_{i,r}(1 + \overline{n}_i) \tag{2.13}$$

For any two classes r and s, we have

$$\frac{\overline{n}_{i,r}}{\overline{n}_{i,s}} = \frac{U_{i,r}}{U_{i,s}} \tag{2.14}$$

Using Equation 2.14 and considering the fact that $\overline{n}_i = \sum_{s=1}^{R} \overline{n}_{i,s}$, Equation 2.13 can be rewritten as

$$\overline{n}_{i,r} = \frac{U_{i,r}}{1 - U_i} \tag{2.15}$$

Applying Little's Law to Equation 2.15, the average residence time for class r requests at resource i is

$$R'_{i,r} = \frac{D_{i,r}}{1 - U_i} \tag{2.16}$$

The interaction among the open classes of a multiple-class model is explicitly represented by the term U_i of Equation 2.16, which corresponds to the total utilization of resource i by all the classes in the model.

The analysis of a product-form model with multiple open classes begins with the constraint that $U_i \leq 1$ for all resources of the network. From Equation 2.9, the stability condition for an open model is

$$U_i \leq 1 \quad \forall \quad i \tag{2.17}$$

$$\sum_{r=1}^{R} \lambda_r D_{i,r} \leq 1 \quad \forall \quad i, r \tag{2.18}$$

2.2.2 Fork-Join Queueing Networks

Parallelism in computer systems can be modeled by fork-join queuing networks [37]. When entering a concurrent processing stage, an arriving request forks into subtasks that are executed independently either on different service centers or on the same service center. Upon completing execution, each subtask waits at the join point for its sibling tasks to complete execution. The fork operation starts new concurrent subtasks. The join operation forces one subtask to wait for sibling subtasks.

Due to the wide-spread use of parallelism in computer and storage systems, fork-join queueing networks have been studied extensively [3, 22, 25, 34, 40, 54, 57–60, 64]. An exact analysis of the fork-join queuing network is presented only for fork-join networks with 2 queues [25,40,58]. There is no known closed-form solution for fork-join queuing networks with more than 2 queues. Hence, the performance measures of such networks computed using approximation and bounding techniques.

In our work, we use the upper-bound on the average response time proposed in [40] for fork-join queueing networks with exponential interarrival and service times. The fork-join queueing network consists of p ($p > 1$) service centers connected in parallel, as illustrated in Figure 2.4. A request to the fork-join network is split into p independent sub-requests, one for each of the p service centers. The original request completes when all subrequests complete, i.e., when all of them arrive at the join point. Thus, the time spent by a request in the fork-join network is the maximum of the times spent in each of the p service centers.

The average service time of a request is assumed to be the same at each of the p service centers of a fork-join network.

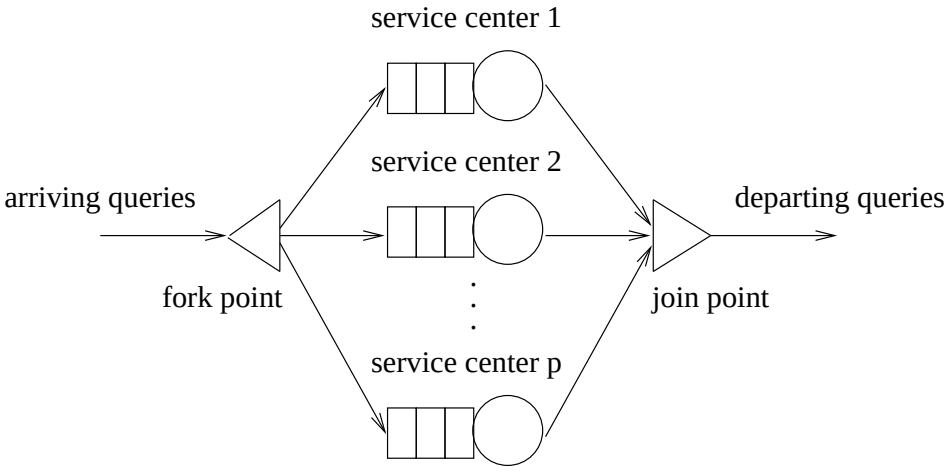


Figure 2.4: Fork-join queueing network.

Consider the following parameters for the fork-join queueing network:

- S: average service time at a service center;
- p: number of parallel service centers at the fork-join network;
- R: average response time at the fork-join network;
- U: utilization of a service center.

The upper-bound on the average response time for the fork-join queueing network is bounded by:

$$R \leq H_p \frac{S}{1-U} \tag{2.19}$$

where H_p is the p-th harmonic number defined as $H_p = \sum_{i=1}^{p}(1/i)$. The harmonic number accounts for the synchronization time at the joining point. The rationale for this approximation comes from the fact that the expectation of a random variable defined as the maximum of p exponentially distributed random variables with average S is equal to SH_p.

Chapter 3

Related Work

In this chapter, we discuss related work in four key areas to the capacity planning for Web search engines: index organization in Section 3.1, system architecture in Section 3.2, workload characterization in Section 3.3, and performance modeling in Section 3.4.

3.1 Index Organization

Different strategies for distributing the index of a document collection among machines have been discussed in the literature. Tomasic and Garcia-Molina [56] compare the performance impact on "boolean and" query processing of two basic and distinct options for storing the inverted lists, namely document partitioning and term partitioning, as illustrated in Figures 3.1 and 3.2 respectively.

Document partitioning: The documents are evenly distributed among index servers and each server generates a local inverted file for its documents. For processing a query, a central broker machine (to which all the queries are first directed) broadcasts the query to all index servers. A query is processed by an index server by reading into memory all the inverted lists related to query terms, intersecting them, and producing a list of matching documents. This partial answer set is sent to the broker, which concatenates all the partial answer sets to produce the answer to the query.

Term partitioning: A global inverted file is generated for all documents, and the full inverted lists are evenly distributed across index servers. For processing a query, the broker determines which index servers hold inverted lists relative to the query terms, breaks the query into subqueries, and sends them to the respective servers. Each

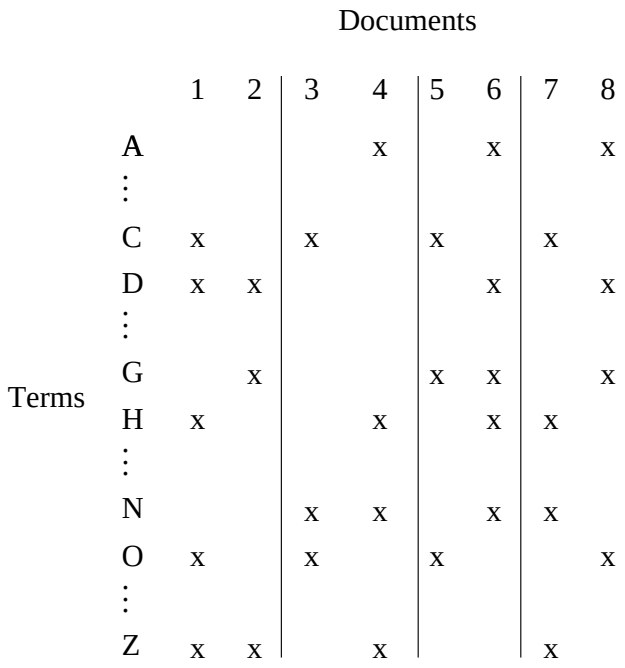


Figure 3.1: Document partitioning.

subquery is composed by the terms which are stored in the index servers it is sent to. If an index server receives a query with a single term, it fetches the corresponding inverted list and returns it to the broker. If the subquery contains multiple terms, the index server intersects the corresponding inverted lists and sends the result as the partial answer set. The broker intersects (instead of concatenating) the partial answer sets to obtain the final answer.

In the work by Tomasic and Garcia-Molina [56], simulation experiments attempt to determine under what conditions each index organization is better, how each index organization scales up to large systems (more documents, more processors) and what is the impact of key parameters, such as seeking time of the storage device, load level, and number of keywords in a query. They generate synthetic databases and queries, from probability distributions that are based on actual statistics. Their results indicate that the document partitioning has the most balanced use of resources, which leads to better performance under more stressful scenarios. The term partitioning performs poorly because it saturates the LAN by transmitting many long inverted lists, and takes longer to read inverted lists which are not split across disks.

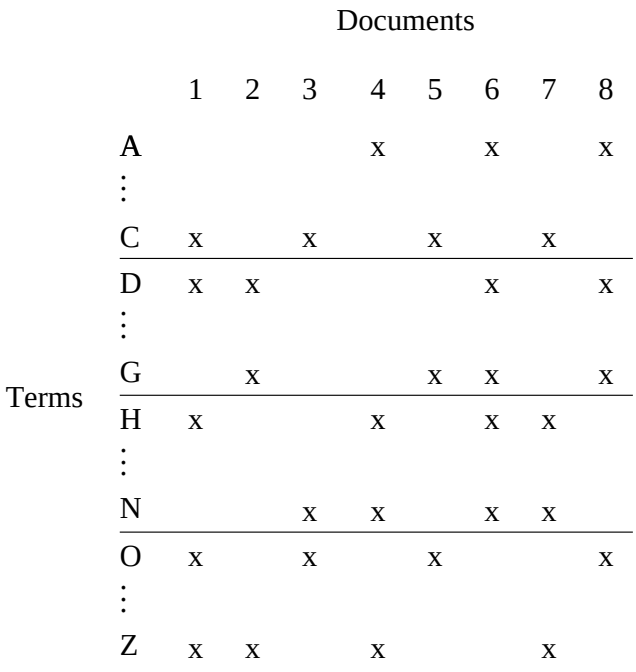


Figure 3.2: Term partitioning.

Jeong and Omiecinski [31] consider the document partitioning and the term partitioning approaches to physically divide inverted indexes in a shared-everything multiprocessor machine with multiple disks. By simulation, they study the performance impact of these schemes on boolean query processing under a number of workloads where the term frequencies in the documents, the term frequencies in the queries, the number of disks, and the multiprogramming level are varied. In general, they found that when the term distribution is less skewed or when the term distribution in the user query is uniformly distributed that the term partitioning performed the best. However, when the term distribution is high skewed, the document partitioning performed the best.

Ribeiro-Neto and Barbosa [44] consider the document partitioning and the term partitioning for a digital library distributed in a tightly coupled environment. The retrieval system uses the vector space model as ranking strategy. The operational environment is that of a network of workstations connected by fast switching technology, where each machine has its own local memory and disk. Experiments were based on an analytical model coupled with a small simulator and investigate how query performance is affected by the index organization, the network speed, and the disks transfer rate. All estimates are based on the documents and queries in the TREC3 collection [27]. The results indicate that term

partitioning consistently outperforms document partitioning for disjunctive queries in the presence of fast communication channels, mainly because term partitioning allows trading seek operations in disk to network traffic and greater concurrency among the various queries.

MacFarlane et al. [35] investigate the document partitioning and term partitioning schemes for distributing inverted lists. The retrieval model implemented is the probabilistic and the system offers only a sequential query service. The search topology is that of a master/slave with a top node and a number of leaf nodes, each with its own disk. The data used in experiments are part of the documents and queries in the TREC7 collection [28]. Experimental results indicate that the document partitioning is the preferable method for sequentially submitted queries. The problem with the term partitioning is that too much data has to be communicated from the leaf to the top process and the sort cannot be parallelized without further communication between leaves and top node.

A previous work of ours [6] compares the document partitioning and term partitioning strategies for distributing inverted lists. The retrieval system uses the vector space model and addresses a concurrent query service. The architecture adopts a network of workstations model and the client-server paradigm. The data used in experiments comprise documents and queries in TREC3 collection [27], besides an artificial query set that mimics Web-like queries. The impact of the two index partitioning strategies on query processing performance was evaluated on a real case framework. Experimental results on retrieval efficiency show that term partitioning outperforms document partitioning for disjunctive queries specially when the number of processors exceeds the average number of terms in query. The main reason is that term partitioning provides a high concurrent query service, which is particularly evidenced when the number of processors exceeds the average number of terms in query.

Sornil and Fox [51] combine the document partitioning and term partitioning approaches to organize inverted indexes in a hybrid partitioned scheme. In the hybrid scheme they divide an inverted list into a number of equal sized chunks, which are randomly distributed to nodes in the system. Their system architecture consists of a number of nodes and an information retrieval server connected through an interconnection network, each node containing one disk and one CPU. They return the entire inverted lists to the information retrieval server, that is, their network of processing nodes can be regarded as one big disk supplying inverted lists to the information retrieval server. They model collections, query term selection, disk nodes, and the network. Simulation results show that the hybrid

partitioning outperforms the document partitioning and the term partitioning over a range of conditions, because it allows load balancing across nodes with a larger partitioning unit than the document partitioning. The term partitioning performs better than the document partitioning in most conditions, because of the saturated nodes due to the amount of I/O incurred by the document partitioning.

In a previous work of ours [4], we compare the performance of the hybrid partitioning, introduced by Sornil and Fox [51], against the document partitioning and the term partitioning. Our work differs from that presented in [51] in the following aspects. While they deal only with returning entire inverted lists to an information retrieval server, we address the complete query processing. Also, while they model documents and queries, we use a real Web collection as experimental data. Our results show that the hybrid partitioning is competitive and sometimes outperforms both document and term partitioning for disjunctive queries. Regarding load balance, the hybrid partitioning outperforms the term partitioning with gains reaching two times. Regarding response time, the hybrid partitioning and the term partitioning have similar performance on average, and both outperform significantly the document partitioning, with gains that reached five times. A similar result is observed regarding the system throughput. A difficulty with the hybrid approach is that it has disadvantages compared to both document partitioning—where each node completely indexes a subcollection, so processing can be node-oriented—, and term partitioning—where the number of disk accesses is minimized. Whether there are advantages is unclear.

Much of the literature comparing document partitioning, term partitioning, and hybrid partitioning is inconsistent. Using simulation and artificial data, Tomasic and Garcia-Molina [56] and Jeong and Omiecinski [31] find in favor of document partitioning. On experimentation and real data, but only 50 queries—insufficient to show disk cache effects—MacFarlane et al. [35] find the same result. Contradicting all of these results, Ribeiro-Neto and Barbosa [44] and Badue et al. [6] find that term partitioning is superior. Using artificial data, Sornil and Fox [51] show that the hybrid partitioning outperforms both the document partitioning and the term partitioning. On real data, Badue et al. [4] confirm that the hybrid partitioning is competitive and sometimes outperforms both the document partitioning and term partitioning. Sornil and Fox [51] and Badue et al. [4] also show that both hybrid partitioning and term partitioning outperform significantly the document partitioning.

However, much of these previous work is open to question. Artificial or small sets of documents or queries are not likely to be predictive of real-world behavior, and simulations designed to estimate time must deal with a great many complex variables—including disk cache operations performed by operating system, relativities of CPU speed, network bandwidth, network delay, disk properties, term skew, and query skew—if they are to be realistic.

Moffat et al. [39] introduces a pipelined query evaluation methodology, based on term partitioning, in which partially evaluated queries are passed amongst the set of index servers that host the query terms. The broker then becomes a traffic director rather than a ranking engine, and the computational load is more effectively shared among the index servers. They compare the pipelined approach to the document partitioning and term partitioning approaches experimentally. The various document collections used in their experiments are all derived from the TREC Terabyte collection built in 2004 by crawling a large number of sites in the .gov domain. In total, the collection contains 426 gigabytes. A stream of $20,000$ queries was derived from the Excite query logs [53]. Their results show that the term partitioning is highly inefficient, the document partitioning has fast response at low query loads, and the pipelining approach provides the best response times and throughput rates at high query loads.

Moffat et al. [38] examine methods for load balancing in the pipelined query evaluation methodology based on term partitioning [39] and propose a suite of techniques for reducing net querying costs. In particular, they explore the load distribution behavior that pipelining displays, and show that the imbalances can be addressed by techniques that include predictive index list assignments to nodes and selective index list replication. Their results are derived from live experimentation in a search system testbed. The data used in experiments comprise 426 gigabytes GOV2 crawl of the .gov domain used in the TREC Terabyte Track since 2004, and a set SYNQ of queries that have been artificially adapted to the GOV2 crawl to give term-frequency, repetition, and answer-frequency properties close to those of real queries (the Excite97 query log) on general Web data (the TREC wt10g collection) [61]. In combination, the techniques they propose increase the throughput of term-distributed indexing by 30%. Nevertheless, local fluctuations in workload mean that each node in the network is less than 100% busy, and while the final throughput rates attained in their experiments remain tantalizingly close to the rates achieved by an equivalent document-distributed computation, they did not succeed in beating document distribution, despite the heavier CPU consumption of the latter. An important conclusion

of their investigation is thus that document partitioning retains its leading position as the method against which others must be judged.

Given its superior performance and popularity in large-scale search engines [14, 45], we adopt the document partitioning for distributing the index of our document collection among index servers (see Section 2.1).

3.2 System Architecture

Barroso et al. [14] describe the cluster architecture of the Google [26] search engine, as illustrated in Figure 3.3. To provide sufficient capacity to handle query traffic, their service consists of multiple clusters distributed worldwide, each cluster with around a few thousand machines. A DNS-based load-balancing system selects a cluster by accounting for the user's geographic proximity to each physical cluster, while also considering the available capacity at the various clusters. A hardware-based load balancer in each cluster monitors the available set of Web servers (that we refer to as brokers) and performs local load balancing of requests across a set of them. After receiving a query, a broker coordinates the query execution and formats the results into a Hypertext Markup Language (HTML) response to the user's browser.

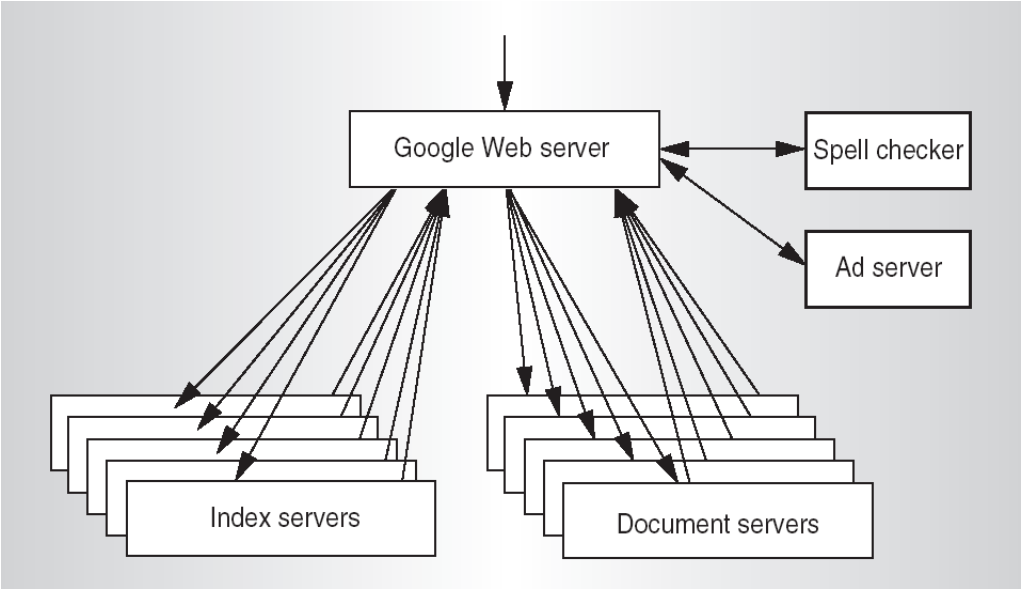


Figure 3.3: Google query-serving architecture.

Query execution consists of two major phases. In the first phase the index servers consult an inverted index, determine a set of relevant documents by intersecting the hit lists of the individual query words, and compute a relevance score for each document. This relevance score determines the order of results on the output page. The search process is parallelized by dividing the index into pieces, each having a randomly chosen subset of documents from the full index (which we refer to as document partitioning). A pool of index servers serves requests for each index piece, and the overall index cluster contains one pool for each index piece. Each request chooses an index server within a pool using an intermediate load balancer. The final result of this first phase of query execution is an ordered list of documents identifiers. We note that index partitioning is used to create scalability in collection size, and replication of each index piece across a pool of index servers is used to create scalability in query throughput. Replication also provides fault tolerance.

The second phase involves taking this list of document identifiers and computing the actual title and Uniform Resource Locator (URL) of these documents, along with query-specific document summary. Document servers handle this job, fetching each document from disk to extract the title and the keyword-in-context snippet. As with the index lookup phase, the strategy is to partition the processing of all documents by randomly distributing documents into smaller index pieces, having multiple replicas of document servers responsible for handling each index piece, and routing requests through a load balancer. When all phases of query processing are complete, a broker generates the appropriate HTML for the output page and returns it to the user's browser.

Risvik et al. [45] describe the cluster architecture of the Fast [1] search engine composed of a dispatcher (that we refer to as broker) and a set of index servers, as illustrated in Figure 3.4. Similarly to the Google architecture described in [14], Fast search engine architecture also adopts document partitioning for scaling on data volume and replication of index partitions for scaling on performance and providing fault tolerance. Let I denote the full index and S_i^j an index server in a $n \times m$ cluster. By data partitioning, the full index I is partitioned across index servers S_i^j in the row j, each server having a disjoint subset I_i of documents from the full index. By data replication, index servers S_i^j in the column i replicate the index partition I_i.

Similarly to the works presented in [14, 45], we consider a Web architecture composed of a single broker and a cluster of index servers (see Section 2.1). The whole collection of documents is partitioned across the index servers, such that each server stores its own

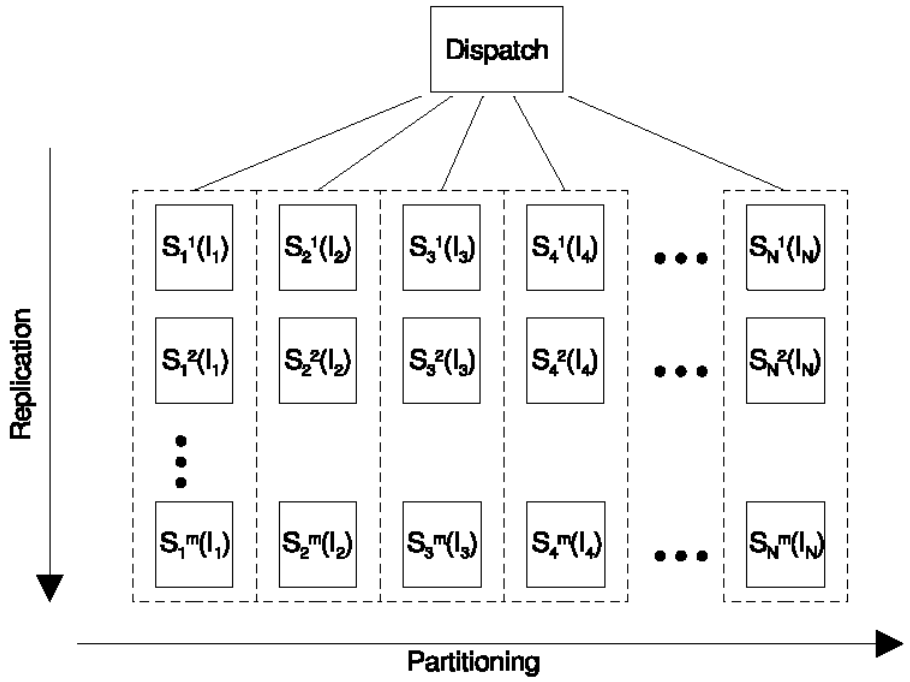


Figure 3.4: Fast search cluster overview.

local subcollection. Upon the arrival of a given user query, the task of retrieving the most relevant documents for this query is then shared among the index servers, so that each server performs the retrieving task for the query only on its partition of the document collection. A broker is responsible for merging the partial ranked answers from the servers to produce the final ranked answer. Note that we focus on the performance of the first phase of the query processing task, i.e., the retrieval of the most relevant documents for a given user query.

Orlando et al. [41] present the architecture of a parallel and distributed engine for searching the Web on which they explore three main parallelization strategies: (i) a task parallel strategy, by which queries are executed independently by a set of homogeneous index servers; (ii) a data parallel strategy (that we refer to as document partitioning), by which each query is processed in parallel by index servers accessing disjoint subsets of documents of the database; and (iii) a hybrid strategy, which is a combination of the task parallel and data parallel strategies. They have conducted real experiments that highlighted the better performance of the hybrid strategy due to a good exploitation of memory hierarchies, in particular of the buffer cache which virtualizes the access to the

disk-resident inverted lists. This work presents a similar analysis to ours in terms of identifying the disk cache operations as a factor for accelerating disk access times. In fact, it inherently considers any imbalance on service times among index servers in its results because they are derived from real experimentation. Nevertheless, this experimentation-based approach fails to be aware of the imbalance and to characterize its impact on the performance of a Web search engine, as we do in our work (see Chapter 4).

3.3 Workload Characterization

Previous work on query characterization for Web search engines mainly focuses on the characterization of user search behavior and user search goals to enhance the relevance to users of the provided answers, i.e., to improve the search *efficacy*. Silverstein et al. [50] present an analysis of individual queries, query duplication, and query sections in a query log from the AltaVista Search Engine. Spink et al. [52] examine the query reformulation by users, and particularly the use of relevance feedback by users of the Excite Web search engine. Rose and Levinson [46] describe a framework for understanding the underlying goals of user searches. Baeza-Yates et al. [11] analyze query log data and show several models about how users search and how users use search engine results. Chau et al. [21] study the information needs and search behavior of the users for a Web site search engine and compare them with those of general-purpose search engine users. Kammenhuber et al. [32] use client-side logs to evaluate user behavior in what they call Web search clickstreams, i.e., search-induced clicks on the answer page provided by the search engine and the subsequently visited hyperlinked pages. Nevertheless, characterizing interarrival times of queries in typical Web search engines is crucial for a performance evaluation of search *efficiency* in terms of, for instance, query response times. This is our focus in our work. Beizel et al. [16] analyze hourly variations in query traffic and remark that the number of queries issued is substantially lower during non-peak hours than peak hours. In our work, besides confirming this result for four different real-world search engines, we provide as an outcome the characterization of queries issued in these search engines with performance evaluation and capacity planning in mind (see Section 5.2).

3.4 Performance Modeling

Although many performance models exist for capacity planning of different systems [37], the availability in the literature of performance models for Web search engines is rather limited.

Cacheda et al. [20] present a case study of different architectures for a distributed information retrieval system, in order to provide a guide to approximate the optimal architecture with a specific set of resources. Using a simulator based on an analytical model for query processing (similar to the one described in [44]), they analyze the effectiveness of a distributed, replicated, and clustered architecture simulating a variable number of workstations. A document model generates synthetic documents from probability distributions that are based on actual statistics from the SPIRIT collection, composed of approximately 94 million documents and 1 terabytes of text, and a query model selects uniformly the number of terms between 1 and 4 terms per query, based on the terms used in the TREC10 topic-relevance queries. Their results show that in a purely distributed information retrieval system, the brokers become the bottleneck due to the high number of local answer sets to be sorted. In a replicated system, the network is the bottleneck due to the high number of query servers and the continuous data interchange with the brokers. Finally, they demonstrate that a clustered system will outperform a replicated system if a high number of query servers is used, essentially due to the reduction of the network load. However, a change in the distribution of the users' queries could reduce the performance of a clustered system.

The analytical model presented in the work by Cacheda et al. [20] assumes that service times are balanced if index servers manage a similar amount of data when processing a query. On the contrary, we found that even with a balanced distribution of the document collection among index servers, relations between the frequency of a query in the collection and the size of its corresponding inverted lists lead to imbalances in query service times at these same servers, because these relations affect disk cache behavior (see Chapter 4). Also, their simulation results show that the brokers become the bottleneck in a distributed system, and the network is the bottleneck in a replicated system. In contrast, we observed in our experiments that the average query residence time at the broker is relatively low compared to the average query system response time, and the network introduces negligible delays to the average query system response time (see Section 5.3).

Chowdhury and Pass [23] introduce a framework based on queueing theory for analyzing and comparing architectures for search systems in terms of their operational requirements:

throughput, response time, and utilization. Using this framework, they also examine a scalability strategy that combines index partitioning and replication to meet operational requirements imposed on search systems. Lastly, they introduce a new cost-based analysis model that finds an optimal set of solutions to consider when designing a search system. Nevertheless, their queueing model obliviously assumes a perfect balance among the service times of index servers that process an equal number of documents per query. Also, they do not verify the accuracy of their model by comparing its predictions with experimental results. In contrast, based on evidence from practical experiments in our work (see Chapter 4), we propose a performance model for capacity planning purposes that considers the imbalance in query service times among homogeneous index servers, while providing a model validation with practical experiments in a real-world testbed (see Chapter 5).

In short, to the best of our knowledge, ours is the first work to propose a capacity planning model for Web search engines based on *experimental work* using actual data and system implementation.

Chapter 4

Analyzing Imbalance among Homogeneous Index Servers

In this chapter, we investigate and analyze the imbalance among homogeneous index servers in a cluster for parallel query processing. Section 4.1 motivates the experimental analysis of the imbalance among homogeneous index servers. Section 4.2 characterizes the workload used in the experimental analysis. Section 4.3 characterizes imbalance in the service times of homogeneous index servers, describing the experimental setup in Section 4.3.1, and the sources for the verified imbalance in Sections 4.3.2 and 4.3.3. Our concluding remarks follow in Section 4.4.

4.1 Introduction

In the architecture for parallel query processing, characterized by a local partitioning of the document collection, the response time of a query is determined by the service time of the slowest index server. As a consequence, imbalance in service times among index servers increases the response time of a query executed by the cluster of servers. Therefore, it is critically important to avoid imbalance among index servers if higher performance is to be achieved.

A common counter-measure against imbalance is to distribute the whole collection of documents among homogeneous index servers in a balanced way, such that each server handles a similar amount of data for processing any given query. Homogeneous index servers have identical configuration of hardware and software. At a first glance, as a consequence of having similar data volumes handled at each server for a given query, one would

expect that service times at the homogeneous index servers would also be approximately balanced. Indeed, this idealized scenario of balanced service times is a usual assumption taken by theoretical models for Web search engines [20, 23, 44]. However, in a real case scenario, relations between the frequencies of queries in the collection and the sizes of the corresponding inverted lists lead to imbalances in query service times.

In this chapter, we carefully investigate and analyze the imbalance issue in a computational cluster composed of homogeneous index servers. As a major contribution, we verify that the idealized scenario of balanced service times at homogeneous index servers with similar data volumes is unlikely to be found in practice. Our results are derived from experiments in an information retrieval testbed fed with real data obtained from a real-world search engine. This is an important experimental result because our findings shed light on a usual assumption that is obliviously taken as valid by previous theoretical models, whereas imbalance masks possibilities for performance improvements. Moreover, we identify and fully analyze the main sources of imbalance: the use of disk cache, the size of main memory in the homogeneous index servers, and the number of servers in the cluster.

4.2 Workload Characterization

The test collection is composed of 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. The inverted index for the whole collection occupies roughly 12 gigabytes. The query set used in our tests is composed of 100 thousand queries, extracted from a partial log of queries submitted to the TodoBR search engine in September 2003.

The distribution of text terms in both documents and queries follows a Zipf's distribution [9, 10]. Figure 4.1 shows the normalized frequency of text terms in documents and the normalized frequency of text terms in queries. The x-axis shows the resulting rank of each text term when these are sorted by decreasing order of occurrence in documents or in queries. Therefore, the frequency in documents (or in queries) that are expected for the x most frequent term is given by

$$f(x) = O(x^{-\alpha}), \qquad \alpha > 0. \tag{4.1}$$

Fitting a straight line to the log-plot of the data presented in Figure 4.1, we can estimate the value of the parameter α that is equal to the slope of the line. Our data shows that this value is 0.93 and 0.80 for the distribution of text terms in documents and in queries, respectively.

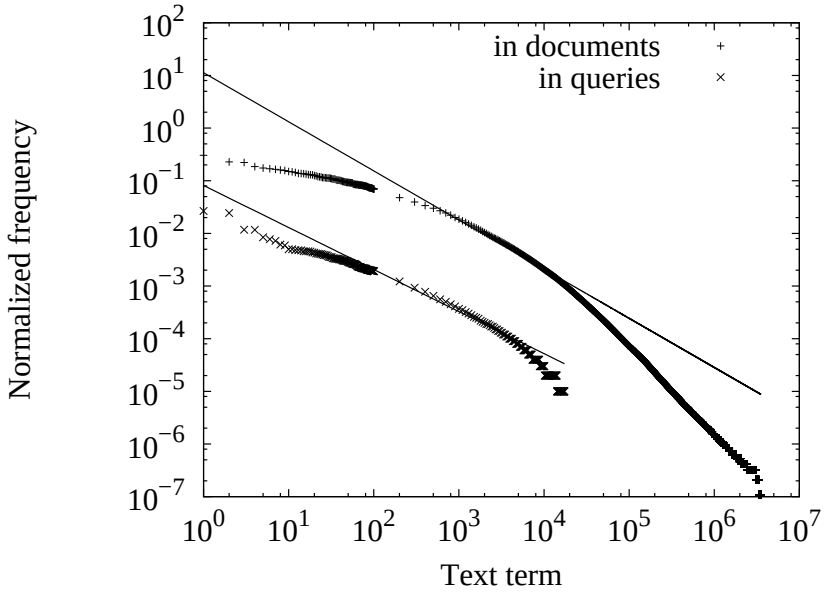


Figure 4.1: Frequency of text terms in documents and in queries.

The query vocabulary has 21,552 terms and the text vocabulary has 3,541,678 terms. Common terms between both collections are 17,468. Figure 4.2 shows the normalized frequency of text terms in the document collection as a function of the normalized frequency of text terms in the query collection, thus considering only the 17,468 common terms between both distributions. Comparing the normalized frequency of text terms in documents and in queries, we observe that—even if dealing with rare query terms—it is likely that query terms are mentioned in a large number of documents. This is important because this indicates that such a query set consistently generates a significant query processing load in our system.

In fact, there are some very rare terms in our collection, thus leading to small inverted lists. As a consequence, when we partition the collection among the index servers, some of them may not store any portion of the inverted lists related to rare terms. In the case of queries concerning such rare terms, the imbalance is calculated as the ratio between the maximum service time and average service time of index servers that have inverted lists for the query terms and effectively participate in the parallel query processing. The number of unparticipating index servers tends to increase with the total number of servers. In our test collection, this case occurs in only 2% of our queries and does not significantly impact the overall performance.

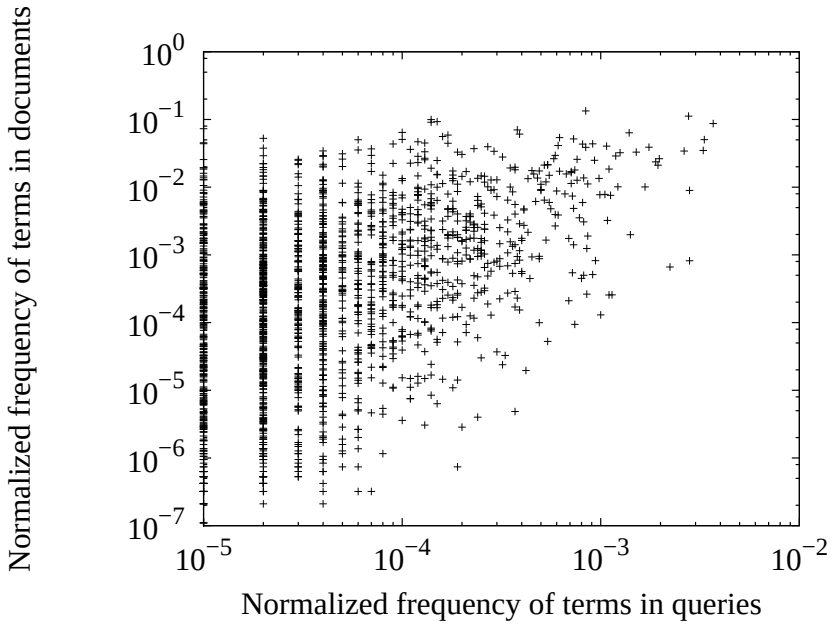


Figure 4.2: Relationship between the frequency of terms in documents and in queries.

It is important to investigate if there is a uniform distribution of document partitions among index servers, because otherwise this would be an expected source of imbalance. Consider our cluster with 7 index servers (detailed in Section 4.3.1), such that documents are randomly distributed in 7 subcollections. Table 4.1 shows the coefficient of correlation between the normalized frequency of text terms in the subcollections of documents and the normalized frequency of text terms in the query collection. We observe that the correlation pattern between the document collection and query remains virtually unchanged after the partition of the whole collection among the index servers. This indicates that data distribution in our experiments seems unlikely to be a significant source of imbalance in the service time of parallel query processing.

Figure 4.3 shows the PMF of the sizes of queries in our query log. The size of a query is given by the sum of the sizes of the inverted lists related to its terms. It is interesting to point out that the distribution of service times of unrelated queries with respect to their size and their frequency in the collection (Figure 4.9) follows the same kind of distribution of sizes of queries.

Table 4.1: Correlation between the frequency of text terms in queries and in subcollections.

Subcollection	Coefficient of correlation
1	0.309722
2	0.309536
3	0.309643
4	0.309901
5	0.309465
6	0.309528
7	0.309692
Whole collection	0.309645

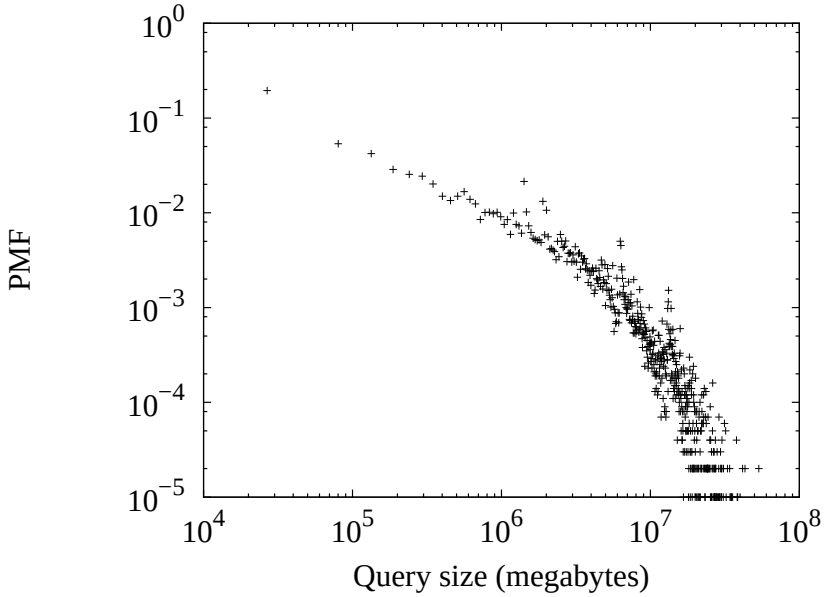


Figure 4.3: PMF of the sizes of queries.

4.3 Characterizing Imbalance

We define the *imbalance of a given query* as the ratio between the maximum service time and average service time of index servers participating in the parallel processing for this particular query. The service time is defined as the time for receiving service at an index server, which does not include the waiting time in queue. This imbalance metric equals 1

in a perfectly balanced scenario that yields the maximum service time exactly matching the average service time. As the imbalance metric progressively gets higher than 1, there is a stronger indication that the query response time is dominated by a much larger service time of a single index server.

In Section 4.3.1, we describe our experimental setup, including the homogeneous cluster of index servers and the uniform distributions of sizes of inverted lists across index servers. Based on this experimental study, we verify the presence of a significant level of imbalance in service time among the servers in despite of the collection being uniformly distributed among these same servers. Moreover, we identify and analyze the main sources for this imbalance: the use of disk cache in Section 4.3.2, and the size of main memory and the number of index servers in the cluster in Section 4.3.3.

4.3.1 Experimental Setup

For the experiments reported in this chapter, we use a cluster of 7 homogeneous index servers. In our setup, each index server is a Pentium IV with a 2.4 gigahertz processor, 1 gigabytes of main memory and a ATA IDE disk of 120 gigabytes. The broker is an ATHLON XP with a 2.2 gigahertz processor and 1 gigabytes of main memory. The client machine, responsible for managing the stream of user queries, is an AMD-K6-2 with a 500 megahertz processor and 256 megabytes of main memory. All of them run the Debian Linux operating system kernel version 2.6. Index servers and broker are connected by a 100 megabits/second high-speed network.

The document collection used in our experiments is relatively small compared to the enormous collections handled by modern search engines. In order to overcome this limitation and establish a scenario to conduct our experiments where the absence of enough capacity for disk cache may happen, we maintain a bounded ratio between the size of the subcollections and the size of the main memory at each index server by limiting the latter to 200 megabytes, unless otherwise stated.

In our experiments, we adopt the standard vector space model to rank the documents retrieved from the index servers (see Section 2.1.3). Combined to the text-based ranking performed by the vector space model, a link-based ranking might be used to improve the relevance evaluation of retrieved documents, i.e., the documents resulting from the intersection of inverted lists related to the query terms. Since link information for each document is pre-computed offline, its usage can be fully carried out using main memory, thus not generating imbalance in query service times among homogeneous index servers,

because the main sources of imbalance are related to disk operations, as further detailed in Sections 4.3.2 and 4.3.3. Also, we evaluate the full inverted lists (see Section 2.1.4). Modern search engines that deal with huge document collections perform a partial evaluation of inverted lists instead of a full one. If we adopt partial evaluation of inverted lists—meaning shorter inverted lists—imbalance in query service times among homogeneous index servers would be expected to be smaller. Nevertheless, partial evaluation of a huge document collection may cause a similar load as the one in our full evaluation case.

To avoid imbalance among index servers, we opt for balancing the distributions of the sizes of the inverted lists that compose the local inverted indexes. To achieve this we simply assign each document to an index server randomly. A random distribution of documents among index servers works well because it naturally spreads documents of various sizes across the cluster. As a result, the distributions of document sizes in the index servers become similar in shape, thus leading to inverted lists whose size distributions are also similar. Our motivation is to balance the storage space utilization at the different index servers and, as a consequence, reduce imbalance in service time at the servers [5, 7, 8], thus minimizing the effects of this possible source of imbalance.

Figure 4.4 illustrates the probability mass function (PMF)[1] of the size of the inverted lists that compose the 7 local inverted indexes in our cluster with 7 index servers (detailed in Section 4.3.1). We observe that the distribution of storage use is very similar in shape throughout the different index servers (actually, they overlap each other in Figure 4.4), indicating that the random assignment of documents to servers works fine to balance storage use among servers.

Although the utilization of disk space at index servers is balanced, as shown in Figure 4.4, we investigate if this balanced storage use among the subcollections reflects on balanced local service times among index servers, or not. Figure 4.5 illustrates the distributions of average, maximum, and minimum local service times per query. These statistics on service time for a query are computed from local service times of index servers that effectively participate in the parallel query processing in our cluster. Interval bars represent the minimum and maximum service times for each query. To allow visual inspection, we display results for selected queries at intervals of 2000 queries.

As an outcome of these experimental results, we verify in practice a consistent imbalance per query in the service time at index servers, even though the distribution of sizes

[1]For discrete random variables, such as the size of inverted lists, we use a probability mass function (PMF). For continuous random variables described later, such as the service time of queries, we use a probability density function (PDF).

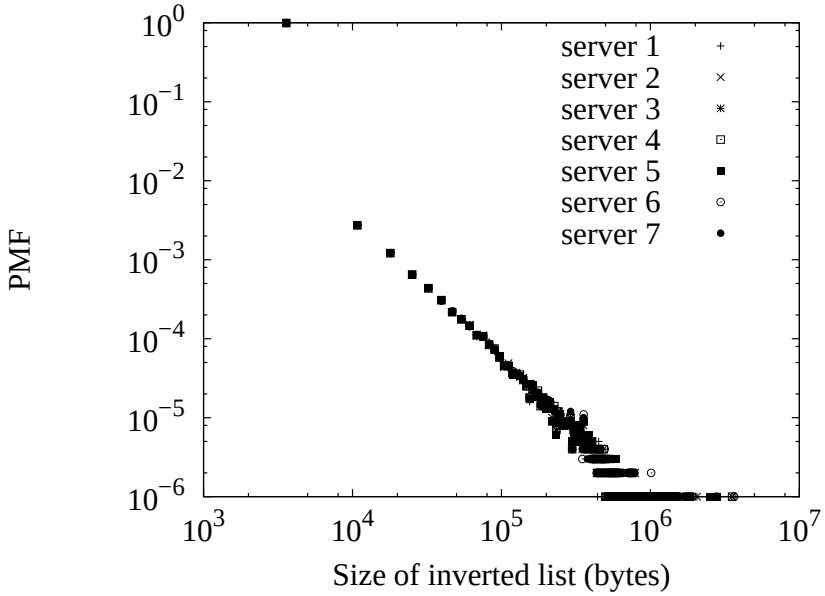


Figure 4.4: PMF of the sizes of inverted lists.

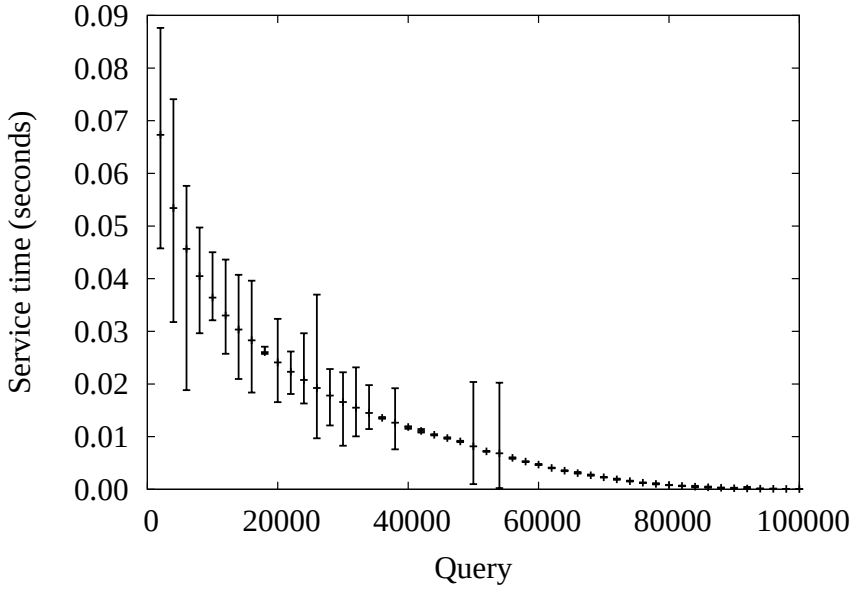


Figure 4.5: Distribution of local service times per query.

of inverted lists at the various index servers are quite balanced. Motivated by this unexpected result, which contradicts the usual assumption of balanced service times adopted

by theoretical models found in the literature, we conduct a comprehensive experimental analysis to investigate the sources for the observed imbalance. As a consequence, we identify the main sources for imbalance: the use of disk cache, the size of main memory in the homogeneous index servers, and the number of servers in the cluster. We analyze the first source of imbalance in Section 4.3.2 and the other ones in Section 4.3.3.

4.3.2 Influence of Disk Cache

We identify the use of disk cache at the different index servers as the major source of imbalance. To illustrate the consequences of this effect on query performance, we refer to a sample query processing observed in our real experiments, where we consider our cluster with 7 index servers and a user query q with the following service times (in milliseconds) at servers: 31.83, 26.41, 30.12, 24.43, 5.27, 35.09, 28.18. For the same sample, the disk access times (in milliseconds) at index servers is: 27.62, 22.18, 25.67, 20.25, 1.01, 30.87, 23.94, and the number of bytes retrieved from disk by the servers is: 374,128, 375,920, 378,328, 375,712, 374,376, 373,864, 373,352. Even though index servers read from the disk a similar amount of data, the service time of server 5 is much smaller than the others (1.01 milliseconds). A possible explanation for this relatively small disk access time is that inverted lists were found in the disk cache of the operating system, thus accelerating disk I/O at this particular index server in comparison with the disk access time observed at the other servers.

Figure 4.6 shows the PDF of local disk access times in our cluster. Note that the distributions of disk access time in the distinct index servers overlap each other, indicating that the behavior of disk access throughout the servers is very similar. We observe that the disk access times at all index servers are basically grouped in two main regions: the first region is related to disk access times less than 4.5 milliseconds and the second region to disk access times greater than 4.5 milliseconds. We attribute the first region of smaller local disk access times to queries whose inverted lists are found in the disk cache (referred to as *cache region*), and the second region of larger disk access times to queries whose lists had to be actually retrieved from disk (referred to as *disk region*). It is interesting to observe that the boundary between these two regions (cache region and disk region) and the peak in the disk region actually correspond to two technical specifications of the adopted storage devices: the average rotational latency (4.5 milliseconds) and the later plus the average seek latency (13.5 milliseconds). Seek latency is the time taken to move disk heads to the right track and rotational latency refers to the waiting time until the right sector is under the read/write head.

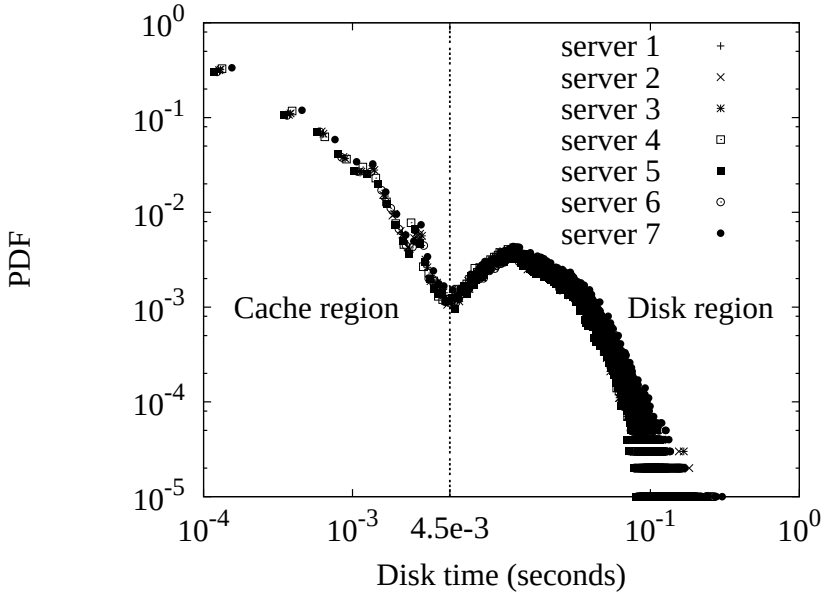


Figure 4.6: PDF of local disk access times.

Note that the disk access can be some orders of magnitude faster if the index server finds the needed data in the cache region, thus avoiding the much slower actual access to the disk. For a given query q, if the local disk access time at a single index server is in the disk region and the local disk access times at the other servers are in the cache region, then the imbalance of query q might be severe.

Indeed, we verify that imbalance in service times among index servers increases with the number of servers operating in the cache region, as shown in Figure 4.7. The points in Figure 4.7 show the imbalance for each query and the line shows the average imbalance over queries as a function of the number of index servers operating in the cache region. This value is of course complementary to the number of index servers operating in the disk region. For example, for a particular query being processed in our cluster of 7 index servers, if Figure 4.7 shows that 2 of them operate in the cache region, then necessarily the other 5 are in disk region, directly influencing the imbalance magnitude.

To better understand how disk cache directly impacts imbalance, it is important to take a careful look at the average imbalance in Figure 4.7 for some representative scenarios: no cache, the worst case, and the best case. In the no cache scenario (i.e., 0 in the x-axis of Figure 4.7), all index servers actually access the disk to retrieve the needed data, obtaining the lowest imbalance (1.38) among the cases where there is at least one server operating in

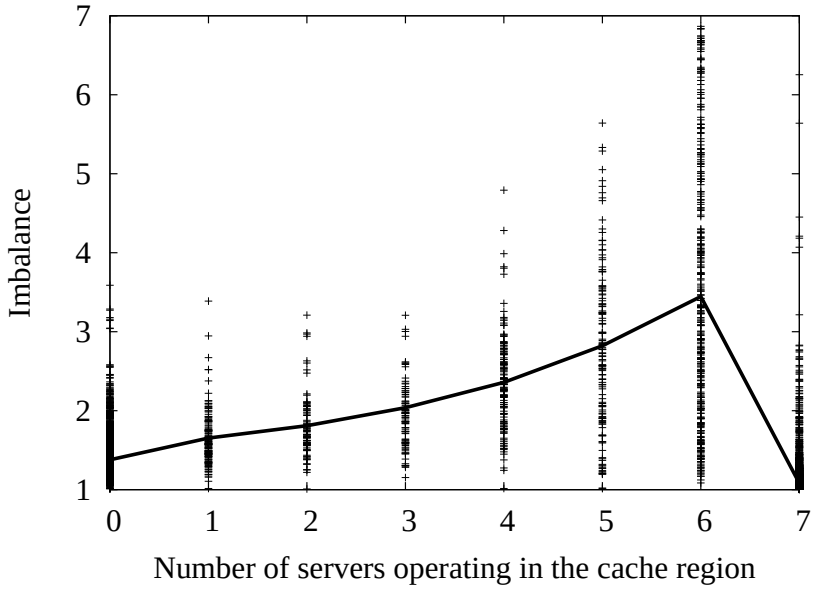


Figure 4.7: Imbalance caused by the number of index servers operating in the cache region.

the disk region. The worst case for imbalance (i.e., 6 in the x-axis of Figure 4.7) presents a much higher imbalance (3.45) because a single index server has a much larger service time than the corresponding service times at all remaining servers, thus leading to a high imbalance value. This happens because there is one single index server that has a large service time and a set of other servers that have much smaller service times because they retrieve the needed information from the disk cache. In contrast, the best case to avoid imbalance (i.e., 7 in the x-axis of Figure 4.7) results in an average imbalance of 1.08 and is achieved when all index servers operate in the cache region, thus resulting in a small imbalance value due to the relatively small and similar disk access times throughout the cluster of servers.

The best case and the no cache scenarios present the two smallest imbalanced results, a consequence of having all index servers operating in the same (cache or disk) region, thus providing no abrupt difference among the service times of the participating servers. Nevertheless, the no cache scenario still yields a significantly higher imbalance with respect to the best case, which can be explained by the higher variance found in direct disk access when compared to the variance found in memory access. Besides having the lowest imbalance, the best case also provides the fastest response time since all needed data to process a query are found in the disk caches at the index servers.

We further analyze the relationship between the size of queries and the frequency of queries in the collection, investigating if there are any links to the use of disk cache. As previously explained, the size of a query is given by the sum of the sizes of the inverted lists related to its terms. Therefore, we consider separately the queries that find a certain level of relation between the size of the inverted lists they demand and their frequency in the collection, and those that do not. To achieve this, we calculate the relation as the ratio between the query size and the query frequency. If this ratio is greater than or equal to 0.25 and less than or equal to 4, then the size and the frequency of the query are related by a factor of 4, which we consider as representing a reasonable level of relation between them. Therefore, queries that fall into this criterion are considered related, otherwise they are considered unrelated.

In Figure 4.8, we plot the normalized size of queries as a function of the normalized frequency of queries in the collection, but we make a distinction between the related and the unrelated queries. When we make this distinction, it is interesting to analyze separately three different representative regions that show up in Figure 4.8: (i) Region 1 is characterized by unrelated data where the size of queries is prevailing over the frequency of queries; (ii) Region 2 contains the related queries; and (iii) Region 3 is characterized by an unrelated region where the frequency of queries is prevailing over the size of queries.

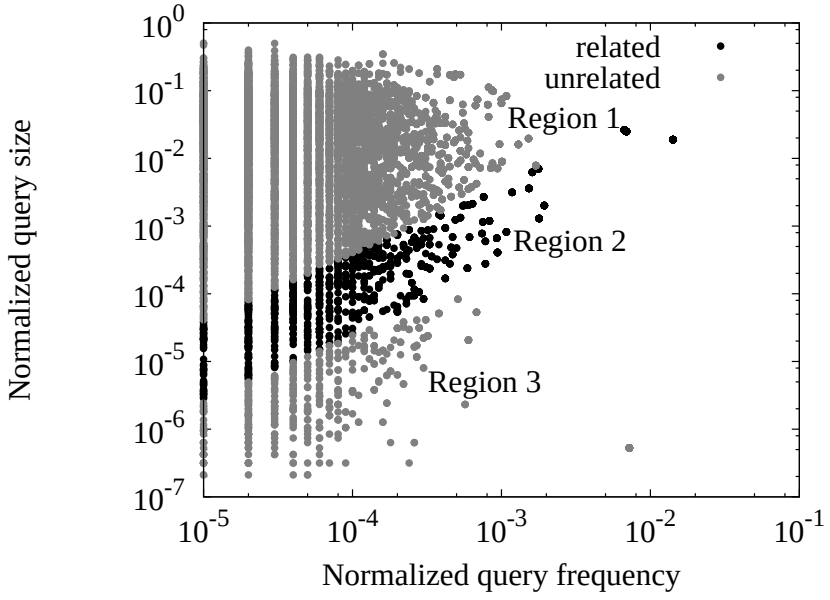


Figure 4.8: Comparing query frequency and query size.

Figure 4.9 compares the service time for related and unrelated queries with respect to their size and their frequency in the collection. This comparison clearly shows that the well related queries (Region 2) have taken a better benefit of disk cache. This happens because they have the best tradeoff between the size of their inverted lists and their frequency in the collection. On the one hand, the largest inverted lists are demanded by the most frequent queries, favoring disk caching of these large inverted lists. On the other hand, rare queries, unlikely to find the inverted lists they need in the disk cache, require the smallest inverted lists that do not demand large transfer times from the disk. For the unrelated data from Region 1, the frequency of queries is proportionally smaller than the size of queries. This implies that rare queries demand for large inverted lists, thus resulting in no use of disk cache and large transfer delays. The unrelated data from Region 3 face the opposite: query terms impose relatively small data volumes to be retrieved in the system, thus getting small service times either through small transfer delay or through the use of disk cache. Although these queries have small service times they are not as numerous as the related ones or the unrelated ones in Region 1. Therefore, related queries prevail as a group in getting the smallest service time. Furthermore, to corroborate this analysis it is important to notice that in Figure 4.9 the unrelated distribution mimics the size distribution (Figure 4.3) while the related one mimics the effect of disk cache (Figure 4.6).

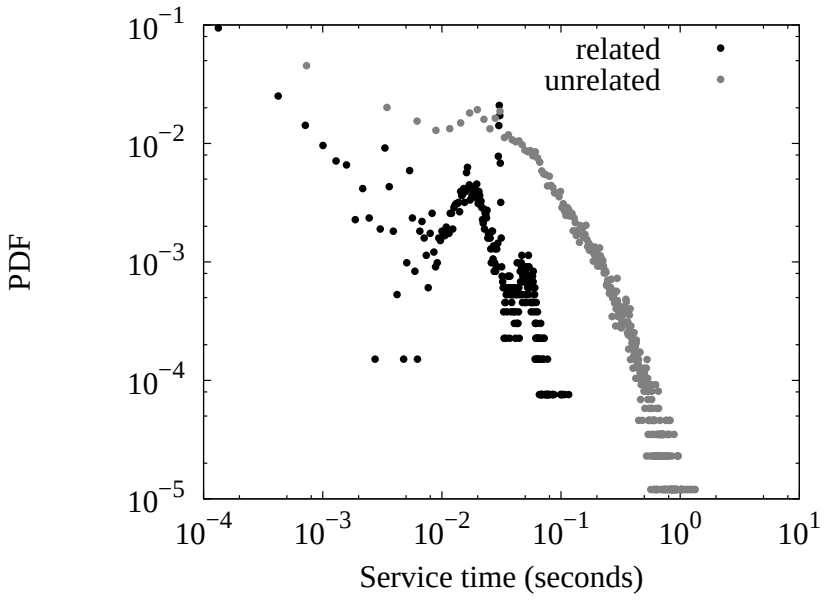


Figure 4.9: PDF of execution times for related and unrelated queries.

These results on the influence of disk cache on imbalance also suggest that the more memory available for disk cache at the index servers, the lower the imbalance, and the larger the number of servers in the cluster, the higher the imbalance, as we will discuss in the following.

4.3.3 Influence of Main Memory Size and Number of Index Servers

The results from Section 4.3.2 indicate that other source of imbalance is the size of the main memory of index servers because this affects the availability of data in the cache region at servers and, as a consequence, the imbalance. Therefore, we investigate in this section how the main memory size at the index servers actually influences on the imbalance in parallel query processing.

Figure 4.10 shows the average imbalance as a function of the number of index servers in our cluster, while varying the size of the main memory at each server. We observe that the average imbalance in service time of index servers increases as the size of main memory decreases, as would be expected. For the average imbalance shown in Figure 4.10, the best fitting we found was a logarithm growth of the number of index servers given by $O(\log^{1.24}(x))$, $O(\log^{1.04}(x))$, $O(\log^{1.08}(x))$, $O(\log^{1.21}(x))$, $O(\log^{1.27}(x))$, for 200, 300, 400, 500, and 600 megabytes of main memory, respectively.

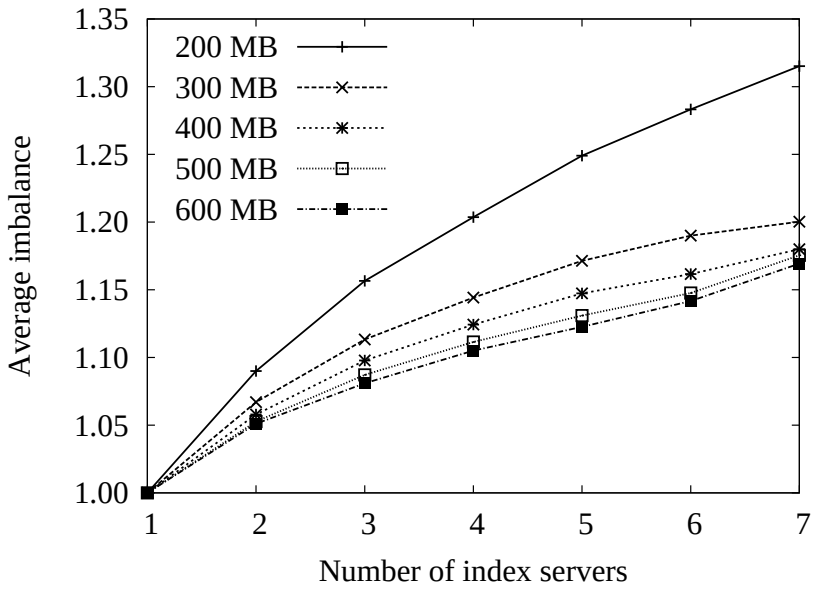


Figure 4.10: Average imbalance as a function of the main memory size at index servers.

On the one hand, when the main memory size is relatively large as compared to the size of the local index stored at the index servers, there is more memory capacity available for the operating system to perform disk cache operations. This implies that local disk access times at all index servers fall into the cache region for a high percentage of queries in our collection and this is exactly the best case scenario that produces the smallest imbalance (see Section 4.3.2). On the other hand, considering a relatively small main memory available for disk cache, index servers need to actually retrieve the inverted lists from the disk. In this scenario, the queries are more susceptible to imbalance as some disk blocks might be found in the disk cache of a few index servers and not be found in the disk cache of the remaining servers. We also point out that there is a diminishing return in terms of imbalance while the RAM memory capacity increases.

The results presented in Section 4.3.2 also indicates that another source of imbalance is the number of index servers in the cluster, because the probability to occur variance among local service times increases with the number of index servers participating in the parallel query processing. We observe in Figure 4.10 that, for a fixed size of main memory, the average imbalance in service times of index servers increases with the number of index servers participating in the parallel query processing. We have already discussed in Section 4.3.2 that the average imbalance increases with the number of index servers operating in the cache region, as shown in Figure 4.7. Therefore, this indicates that the larger the number of index servers participating in the parallel query processing, the higher the probability of increasing the ratio between the number of servers operating in the cache region and those in the disk region. As a consequence, this leads to larger imbalance in service times per query in the cluster.

4.4 Concluding Remarks

In this chapter, we investigated and analyzed the imbalance among homogeneous index servers in a cluster for parallel query processing. We verified a consistent imbalance per query in the service time at index servers, even though the distribution of sizes of inverted lists at the servers are quite balanced. Our results are derived from a comprehensive experimental analysis using an information retrieval testbed and real data obtained from a real-world search engine. This is an important experimental result because it sheds light on the usual assumption of balanced service times adopted by many theoretical models in the literature to simplify their modeling task [20, 23, 44]. Further, we have also identified and

fully analyzed the main sources for this unexpected imbalance: the use of disk cache, the size of main memory in the homogeneous index servers, and the number of index servers in the cluster.

Chapter 5

A Capacity Planning Model for Web Search Engines

In this chapter, we propose a capacity planning model for Web search engines. Section 5.1 introduces our model. Section 5.2 characterizes the query datasets used in our experimental analysis. Section 5.3 describes our capacity planning strategy for modern Web search engine architectures. Our concluding remarks follow in Section 5.4.

5.1 Introduction

Although many performance models exist for capacity planning of different systems [37], the availability in the literature of performance models for Web search engines is rather limited. Cacheda et al. [20] present a case study of different architectures for a distributed information retrieval system, in order to provide a guide to approximate the optimal architecture with a specific set of resources. Using a simulator based on an analytical model for query processing, they analyze the effectiveness of a distributed, replicated, and clustered architecture simulating a variable number of workstations. Chowdhury and Pass [23] introduce an approach based on queueing theory for modeling and analyzing architectures for search systems in terms of their operational requirements: throughput, response time, and utilization. Both the analytical model in [20] and the queueing model in [23] simply assume balanced service times among homogeneous index servers that process an equal number of documents per query. However, even with a balanced distribution of the document collection among index servers, relations between the frequency of a query in the

collection and the size of its corresponding inverted lists lead to imbalances in query service times at these same servers, because these relations affect disk cache behavior.

In this chapter, we propose a capacity planning model for Web search engines that considers the imbalance in query service times among homogeneous index servers. Our model, which is based on a queueing network, is simple and reasonably accurate. To set up the parameters of our model, we run experiments on a small cluster of index servers that we have available. Once the key parameters have been estimated, we can use our model to very quickly gain insight into the behavior of the Web search engine. To illustrate the applicability of our model in realistic scenarios, we consider a collection of 20 billion documents partitioned among 2,000 index servers and analyze the impact of adopting faster CPUs and disks on the query system response time.

Despite the use of stochastic modeling techniques, our work has a strong experimental nature. We rely on experimental measurements taken using an actual cluster of index servers to fine tune the parameters in our model. Further, we verify the accuracy of the model by comparing its predictions with experimental results also produced using the cluster of index servers.

Given the simplicity of our model, and its yet reasonable accuracy, we believe that it can be used successfully to study the behavior of large and complex modern search engines in a variety of scenarios.

5.2 Workload Characterization

In this section, we characterize the query arrival process based on actual access logs from typical Web search engines. We analyze query datasets representing the query workload of four different real-world Web search engines: TodoBR [55], Radix [42], AllTheWeb [1], and Altavista [2]. TodoBR and Radix focus on the Brazilian Web, whereas AllTheWeb and Altavista are worldwide search engines. We remark that the current availability of query datasets from modern operational search engines is rather restricted because such data are usually considered sensitive to search engine operators.

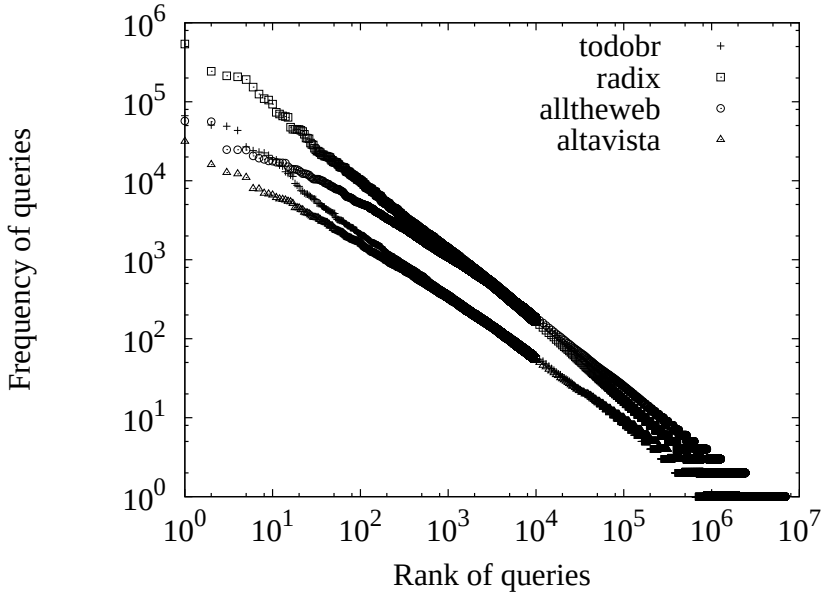
We first characterize the query collection provided by each considered dataset. Table 5.1 presents the length of each query dataset in different dimensions such as the number of observed days and the total number of registered queries. We also analyze how diverse queries are as well as how popular are the terms in queries found in each dataset. Figure 5.1 compares the frequency of unique queries and the frequency of unique terms in queries

throughout the four datasets. We verify that—although the query datasets cover different time periods, query loads, users, and languages—the distribution of queries and of terms in queries throughout the datasets are quite similar and follow a Zipf's distribution [9, 10]. Fitting a straight line to the log-plot of the data presented in Figure 5.1, we can estimate the value of the parameter α that is equal to the slope of the line. For the distribution of queries, the values of the parameter α of the Zipf's distribution are 0.82, 0.89, 0.75, and 0.74 for the TodoBr, Radix, AllTheWeb, and Altavista datasets, respectively. For the distribution of terms in queries, the values of the parameter α of the Zipf's distribution are 0.98, 1.09, 0.90, and 0.88 for the TodoBr, Radix, AllTheWeb, and Altavista datasets, respectively.

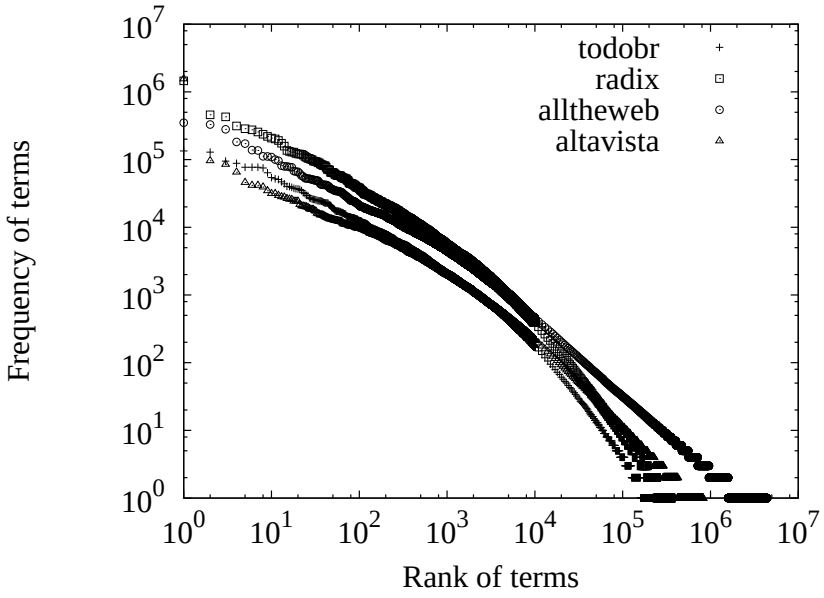
Table 5.1: Length of the considered query datasets.

	TodoBR	Radix	AllTheWeb	Altavista
Dataset begins at	Jan 01 2003	Jan 01 2003	Sep 01 2003	Sep 28 2001
Dataset ends at	Aug 31 2003	Aug 31 2003	Sep 21 2003	Oct 03 2001
Number of days	243	243	21	6
Number of queries	6,806,795	19,934,196	25,080,586	7,169,365
Number of unique queries	1,552,735	2,830,854	6,902,160	2,096,598
Number of unique terms	228,396	358,406	4,408,672	820,817
Average number of queries per day	28,012	82,034	1,194,314	1,194,893

Each query comprises a certain number of terms, thereby different queries impose a varying processing demand for document retrieval on the search engine collections. The performance effects of this heterogeneous demand on the Web search engine as a function of the number of terms in each query are further investigated in Chapter 5. For query characterization purposes, as in this section, we focus on the distribution of query lengths through the considered query datasets. On one hand, the maximum number of terms per query we observe in each dataset is 282 (TodoBR), 287 (Radix), 841 (AllTheWeb), 171 (Altavista), basically from a few users that copied a whole text snippet into the query box. On the other hand, however, the median query length is 1 for AllTheWeb, and 2 for TodoBR, Radix, and Altavista datasets, while the mean query length across the datasets is 2.02 (TodoBR), 1.91 (Radix), 1.70 (AllTheWeb), and 2.22 (Altavista). These results



(a) Frequency of queries



(b) Frequency of terms in queries

Figure 5.1: Frequency of unique queries and unique terms in queries.

suggest a significant trend towards queries composed of just one to a few terms. We thus define two classes to investigate the performance effects of queries with different lengths, namely class *small* (queries with at most 2 terms) and class *large* (queries with more than 2 terms). Table 5.2 shows the distribution of queries in the observed datasets onto each

defined class. The presented results confirm the prevalence of very short queries—i.e., queries containing two or fewer terms—for all datasets.

Table 5.2: Query class distribution.

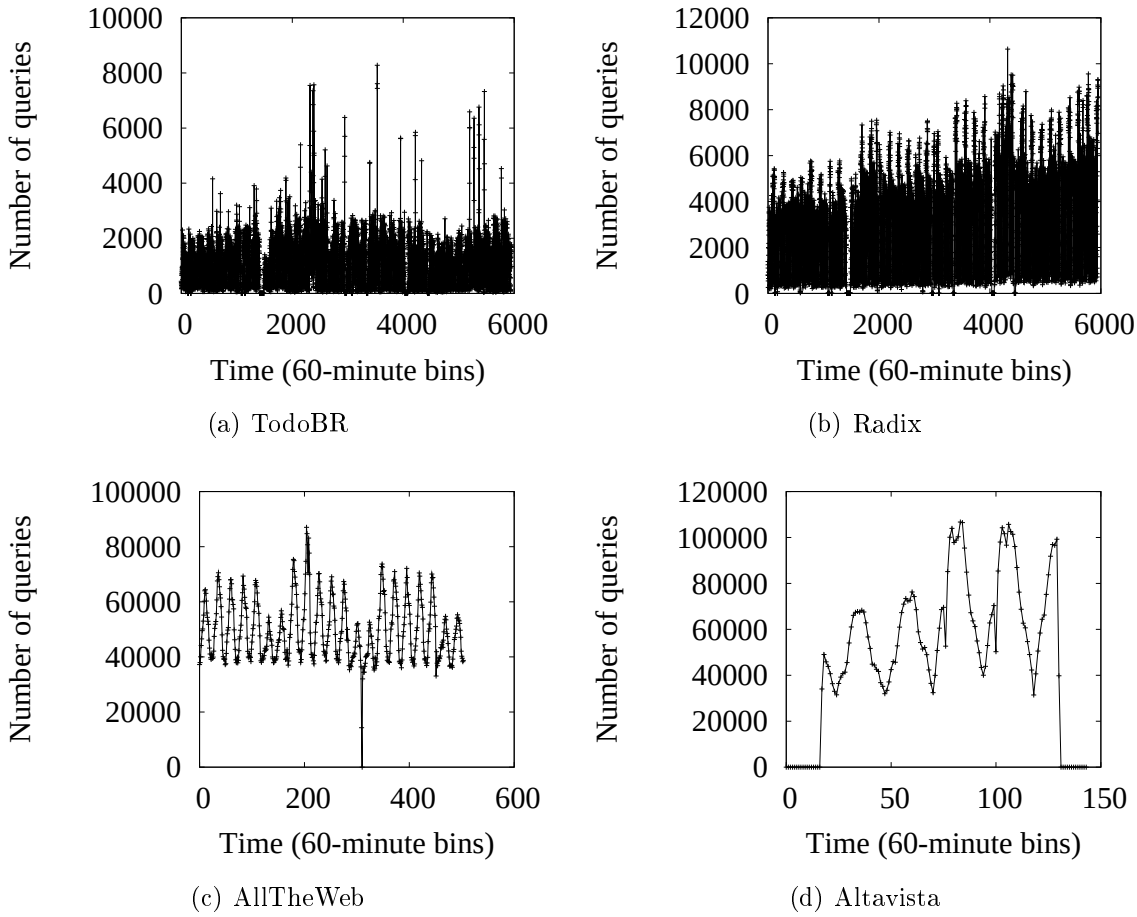
Class	TodoBR	Radix	AllTheWeb	Altavista
small	0.73	0.78	0.84	0.68
large	0.27	0.22	0.16	0.32

A periodic behavior on the query workload is consistently observed in the four considered query datasets. Figure 5.2 presents the query workloads measured in terms of the number of queries within 60-minute bins over the whole duration of each dataset. Figure 5.3 shows the mean query workload over all weeks. The query workload clearly presents daily load variations. In particular, working days present similar loads among them that are different from those observed on weekend days.

Table 5.3 shows the mean query arrival rate (in queries per second) over all weeks for all queries and queries in each class for all datasets. AllTheWeb and Altavista datasets present a heavier query load than TodoBR and Radix. This is an expected result since the former are worldwide search engines and the latter regional ones.

In spite of the number of available query datasets being rather limited, it is even harder to obtain datasets for the collection of documents used by Web search engines to generate an answer page for each received query. This is so because the set of collected documents is seen as strategic and proprietary information by the major Web search operators. In contrast, having access to the document collection is fundamental for analyzing the performance of a Web search engine in face of a given query workload using real experiments as we do in Chapter 5. Although we have access to four different query datasets, we only have access to the document collection of one of them, namely TodoBR. This would be somewhat constraining for the performance analysis because this Web search engine is relatively light-loaded in terms of query arrival rates as compared to other worldwide ones (e.g., AllTheWeb and Altavista).

The solution we adopt is to apply a folding procedure on the TodoBR dataset to boost its query arrival rate. Therefore, we fold the TodoBR dataset by merging all queries of each day of a week using data from the entire dataset, while still keeping the original characteristics of workload aspects with key impact on system performance, such as distribution of queries, distribution of terms in queries, distribution of the number of terms in queries,

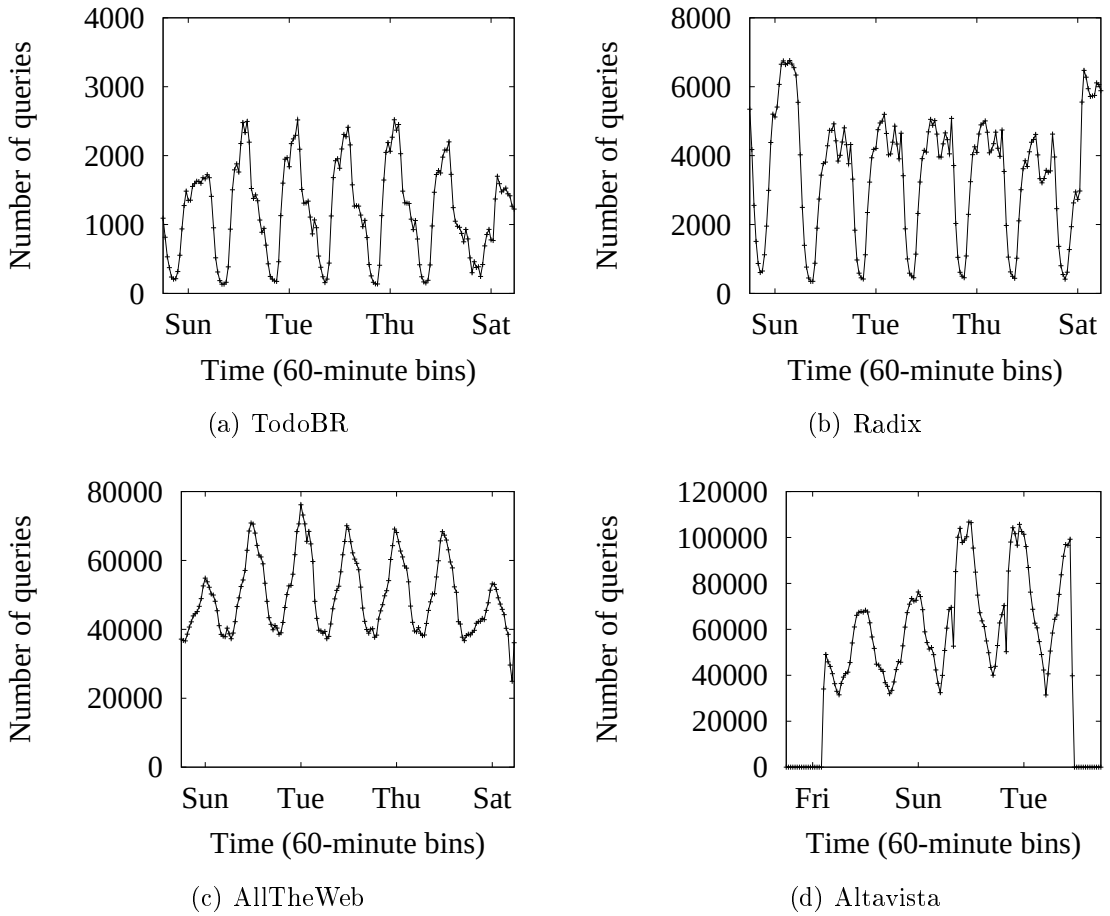


(a) TodoBR (b) Radix

(c) AllTheWeb (d) Altavista

Figure 5.2: Query load variation through the considered datasets.

and distribution of the interarrival times of queries characterized later. Figure 5.4 presents the daily load variation in the resulting folded TodoBR dataset. We highlight that the folded TodoBR dataset achieves query arrival rates similar to those observed in the more heavy-loaded Web search engines we study. This can be observed as we compare the results from Table 5.4 (for the query arrival rate of the folded TodoBR dataset) with the results presented in Table 5.3 for AllTheWeb and Altavista query datasets, and the results from Figure 5.4 (for the daily load variation in the folded TodoBR dataset) with the results presented in Figures 5.3(c) and 5.3(d) for the AllTheWeb and Altavista datasets, respectively. Through this folding procedure we are able to analyze and model the performance of a Web search engine in Chapter 5 based on the handling of this folded query dataset by a cluster of index servers with the corresponding document collection of the same search engine.



(a) TodoBR (b) Radix

(c) AllTheWeb (d) Altavista

Figure 5.3: Mean number of queries over time (modulo one week) for the considered datasets.

We also characterize the interarrival process of the folded version of the TodoBR dataset, which is used as input stream in the analysis and modeling further described in Chapter 5. We evaluate the fitting provided by a diverse set of well-known distributions to the interarrival time distribution observed per query class in several high-load hours with stable arrival rate of the folded TodoBR dataset. For all time periods analyzed, we verify that the Exponential distribution—although not providing the best-fit of all—presents a fairly reasonable fitting (for both small and large classes of queries) compared to the Gamma and Weibull distributions, whereas the Lognormal and Pareto distributions fail to model the observed data by far. Table 5.5 presents the sum of the squares of the differences between the observed interarrival time distribution per query class in a high-load hour and the best-fit provided by each of these well-known distributions.

Table 5.3: Query arrival rate (queries/second).

TodoBR			
Day	All	Class small	Class large
Sun	0.48	0.36	0.12
Mon	0.69	0.51	0.18
Tue	0.70	0.51	0.19
Wed	0.67	0.49	0.18
Thu	0.70	0.53	0.17
Fri	0.61	0.42	0.19
Sat	0.47	0.33	0.14

Radix			
Day	All	Class small	Class large
Sun	1.88	1.46	0.42
Mon	1.36	1.06	0.30
Tue	1.45	1.13	0.32
Wed	1.40	1.09	0.31
Thu	1.40	1.11	0.29
Fri	1.33	1.05	0.28
Sat	1.80	1.40	0.40

AllTheWeb			
Day	All	Class small	Class large
Sun	15.24	13.10	2.14
Mon	19.68	16.53	3.15
Tue	21.16	17.72	3.44
Wed	19.46	16.36	3.10
Thu	19.19	16.17	3.02
Fri	18.99	16.07	2.92
Sat	14.78	12.66	2.12

Altavista			
Day	All	Class small	Class large
Sun	21.16	14.55	6.61
Mon	29.64	20.77	8.87
Tue	29.36	20.15	9.21
Wed	27.54	19.05	8.49
Thu	—	—	—
Fri	13.61	9.32	4.29
Sat	18.97	13.03	5.94

For the sake of simplicity in modeling, we choose to proceed based on the assumption of an exponential interarrival process for queries. We point out that this decision is based on the fact that the Exponential distribution is much simpler to model than the other ones and that our results demonstrate that the Exponential distribution approximates the observed interarrival process of queries within quite acceptable bounds. In order to illustrate how close the fitting provided by the Exponential distribution is to the observed interarrival time distribution for the two classes of queries, we show the exponential fitting of query interarrival times in a high-load hour of the folded TodoBR dataset in Figure 5.5.

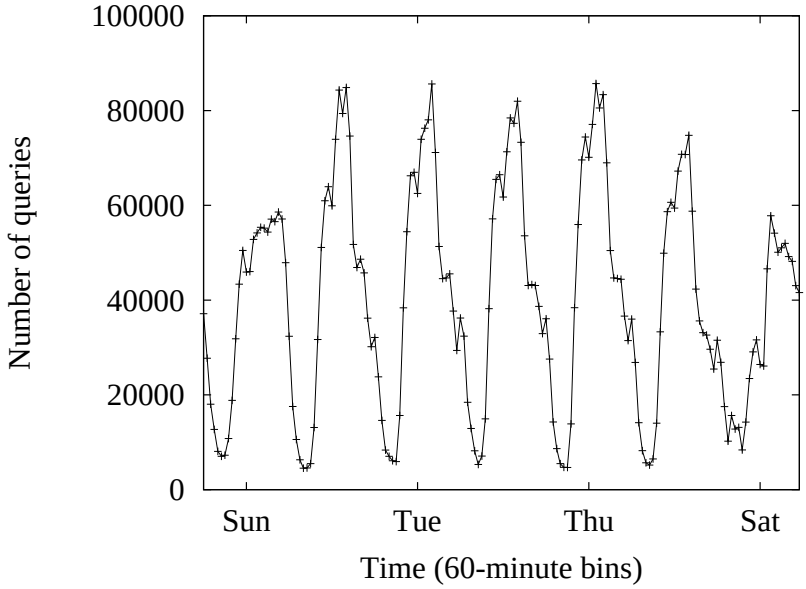


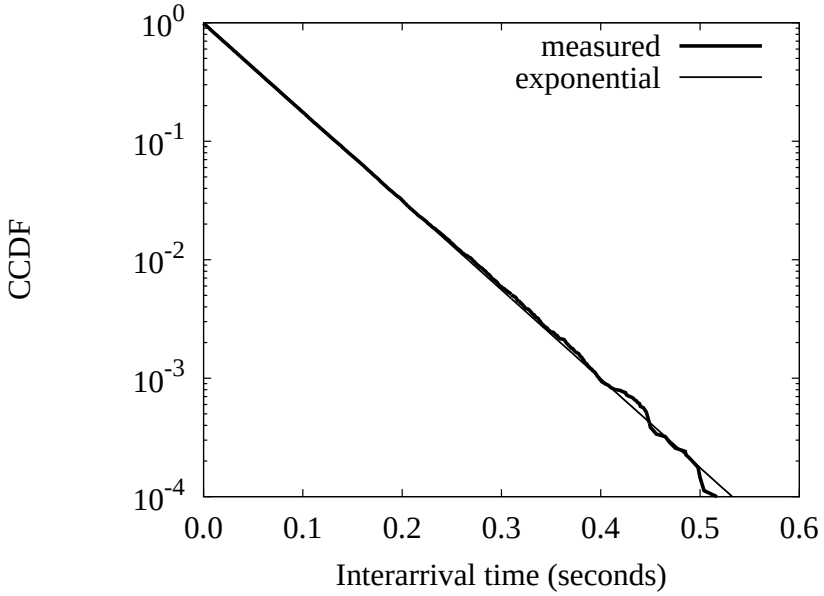
Figure 5.4: Daily load variation in the folded TodoBR log.

Table 5.4: Query arrival rate in the folded TodoBR log (queries/second).

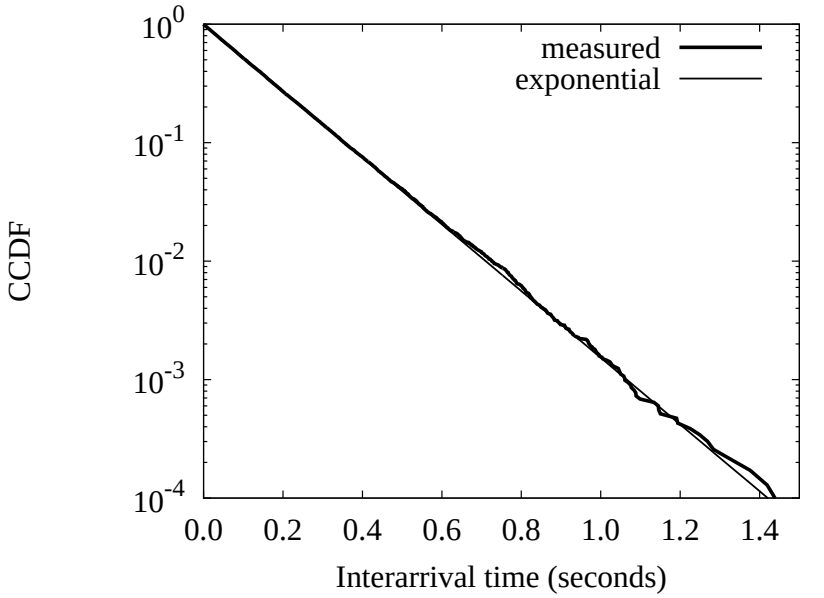
Day	All	Class small	Class large
Sun	16.27	12.08	4.19
Mon	23.58	17.42	6.16
Tue	23.79	17.30	6.49
Wed	22.77	16.67	6.10
Thu	23.80	18.03	5.77
Fri	20.77	14.21	6.56
Sat	16.05	11.20	4.85

5.3 Capacity Planning for Search Engines

This section describes our capacity planning strategy for modern Web search engine architectures, described in Section 2.1. Our capacity planning strategy relies on a queueing-based analytical model to estimate the *average query system response time*. Its design was driven by the empirical observation that a tool to be useful to Web engineers should be easy to configure and to apply in practical scenarios. Thus, model simplicity is of utmost



(a) Class small



(b) Class large

Figure 5.5: Distribution of interarrival times per query class.

importance, even if it comes at the cost of a *reasonable* compromise in model accuracy. As it will be shown, our model is simple, relies on easy-to-collect data, and still has reasonable accuracy.

Table 5.5: Sum of the squares of the differences between the measured and fitted distribution of interarrival times per query class.

Distribution	Class small	Class large
Exponential	0.003703	0.003777
Gamma	0.004901	0.001215
Weibull	0.002582	0.000874
Lognormal	0.382970	0.958472
Pareto	5.062961	9.861788

5.3.1 Performance Model Overview

The Web search engine is represented by the queueing network described in Figure 5.6. We model the system as an open queueing network composed of the broker and the subsystem of index servers. We assume the index servers have homogeneous resources (as would be the case in several real scenarios) and that the collection of documents is uniformly distributed over all servers. Thus, the load is assumed to be balanced across all index servers. Finally, the network connecting index servers and broker is typically a high-speed network and, as observed in our experiments, introduces negligible delays to the query system response time. Therefore, it is not explicitly represented in our model.

In our Web search model, the subsystem composed of all index servers is modeled as a fork-join queueing network [37]. In a fork-join queueing network, each arriving task (i.e., query) is split (i.e., *fork*) into p identical sub-tasks. Each sub-task is sent to a different index server. The queueing discipline at each index server is FCFS (First-Come First-Served). When a sub-task finishes execution, it will wait until all the other sub-tasks finish (i.e., *join*). Only at this moment the task completes execution and leaves the network. This behavior mimics the parallelism in processing queries by the index servers and the synchronization introduced at the broker for combining partial results. Mean Value Analysis (MVA) [43] offers an efficient solution for product-form queueing networks. In particular, MVA can be used to produce performance estimates for each individual index server. However, the fork-join feature violates the assumptions required by the exact MVA solution. Thus, we use approximate MVA and bounding techniques [37] (see Sections 2.2.1 and 2.2.2) to solve the complete Web search model.

As indicated in Section 5.2, typical queries may have heterogeneous demands for different resources of the index servers depending on the number of terms they contain.

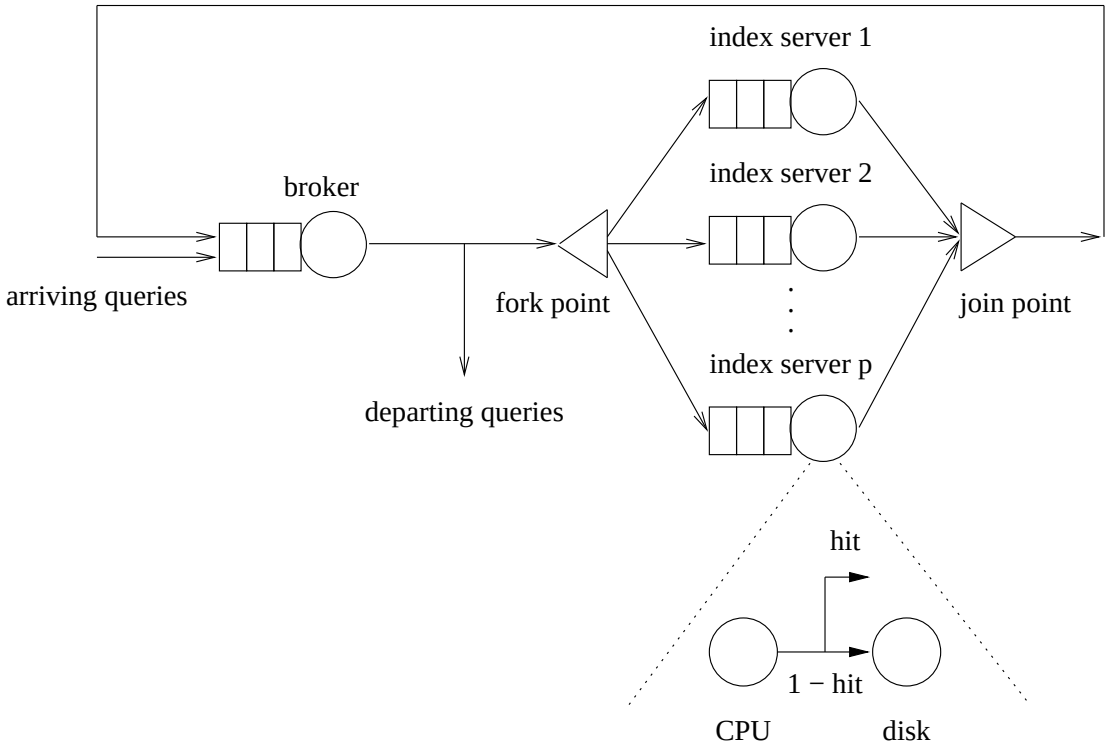


Figure 5.6: Queueing network for a Web search engine.

Moreover, recall that in order to process a query, an index server needs to retrieve the inverted lists related to the query terms from disk. Thus, query service time at the index server is dominated by disk time and, possibly, CPU time. However, due to the locality of reference in the terms of queries that reach the search engine, an index server might find some or all the inverted lists in the disk cache (in memory). Thus, some queries may not retrieve any data from disk. In fact, during our validation experiments (see Section 5.3.2), we found a non-negligible number of such queries in all classes of our workloads.

In order to capture the impact of query heterogeneous resource demands, we refine our index server model as follows. First, we separately model two query classes (as defined in Section 5.2), namely class *small* (queries with at most 2 terms) and class *large* (queries with more than 2 terms). Second, we separately model the average demands for CPU and disk, as well as the probability of full disk cache hit (i.e., *all* inverted lists are found in the disk cache) at an index server for each query class. Queries of class *large* have larger demands for CPU and disk of an index server than queries of class *small*, while queries of class *small* have higher probabilities of full disk cache hit at an index server than queries in class class *large*. Note that the impact of partial disk cache hits is indirectly captured

by the CPU and disk demands of each class. We further assume that queries may have different CPU demands depending on whether they retrieve any data from the disk. Given that queries are processed sequentially by each index server (see Section 2.1), there is no queueing at any resource (CPU nor disk) of an index server.

Finally, the query residence time at the broker consists of local processing for broadcasting the query to all index servers, receiving partial results from all servers, and merging the received partial results, which depends on the number of servers in the cluster. We remark that the average query residence time at the broker is relatively low compared to the average query system response time. Indeed, for our experiments with varied parameters, the average query residence time at the broker is negligible, reaching at most 0.01% of the average query system response time. There are two fundamental reasons for this. First, broker's operation is fully carried out using main memory, thus demanding only CPU time as opposed to an index server's operation that is composed of CPU and disk demands. Second, all the tasks the broker executes are relatively simple tasks that do not take much CPU time. It should be noted that the broker does not have to make ranking computations and does not have to execute algebraic operations, other than comparing document identifiers. Table 5.6 presents the system and workload input parameters as well as the output parameters of our model. The average residence time of a query at a resource is defined as the sum of the average waiting time in queue and the average service time (i.e., the average time for receiving service at the resource).

5.3.2 Model Solution

This section describes our solution to estimate the average query system response time of a Web search engine, modeled as shown in Figure 5.6. Recall that our main design goals are simplicity and reasonable accuracy. Moreover, we are particularly interested in solutions that deliver a good tradeoff for heavy load scenarios, when the search engine is approaching saturation (i.e., server utilization close to 100%).

Index Server Model

This section derives the average query residence time at one index server. Considering that queries are processed one at a time by each index server (see Section 2.1), there is no queueing at any resource (CPU nor disk) of an index server. Thus, we introduce here an

Table 5.6: Input and output parameters of our model.

Inputs	Description
p	Number of index servers
R	number of query classes
λ_r	Query arrival rate for class r ($r = 1 \dots R$)
S_p^{broker}	Average query service time at the broker for a cluster with p index servers
$D_{cpu_{hit},r}^{server}$	Average demand for CPU at an index server for class r queries that find all inverted lists in the disk cache
$D_{cpu_{miss},r}^{server}$	Average demand for CPU at an index server for class r queries that retrieve data from disk
$D_{disk,r}^{server}$	Average disk demand at an index server for class r queries
hit_r	Probability of a class r query finding all inverted lists in the disk cache

Outputs	Description
R^{system}	Average query system response time
$R^{cluster}$	Average query residence time at the index server subsystem
R_p^{broker}	Average query residence time at the broker for a cluster with p index servers
R^{server}	Average query residence time at an index server
R_r^{server}	Average residence time for class r queries at an index server
S^{server}	Average query service time at an index server
S_r^{server}	Average service time for class r queries at an index server
U^{server}	Total resource utilization of an index server

abstraction for the index server as a single service center, whose average service time of a class r query (i.e., the average time required to process it) is given by:

$$S_r^{server} = hit_r D_{cpu_{hit},r}^{server} + (1 - hit_r)(D_{cpu_{miss},r}^{server} + D_{disk,r}^{server}) \tag{5.1}$$

The average query service time at an index server is estimated as the weighted average of the service time for each class, with the weights given by their relative throughputs:

$$S^{server} = \sum_{r=1}^{R} \frac{\lambda_r}{\lambda} S_r^{server} \tag{5.2}$$

Using Little's Result [37], the utilization of the service center representing an index server by class r queries can be estimated by multiplying S_r^{server} by the corresponding query arrival rate λ_r. The total server utilization can be calculated as:

$$U^{server} = \sum_{r=1}^{R} \lambda_r S_r^{server} \tag{5.3}$$

Using an MVA equation for open networks [37] (see Sections 2.2.1), we estimate the average residence time of a class r query at an index server as:

$$R_r^{server} = \frac{S_r^{server}}{1 - U^{server}} \tag{5.4}$$

Finally, the average query residence time at an index server is estimated as the weighted average of the residence times for each class as:

$$R^{server} = \sum_{r=1}^{R} \frac{\lambda_r}{\lambda} R_r^{server} \tag{5.5}$$

System Model

The average query system response time is the sum of the average query residence times at the broker and at the index server subsystem. The average query residence time at the broker for a cluster with p index servers can be easily estimated using MVA as:

$$R_p^{broker} = \frac{S_p^{broker}}{1 - \lambda S_p^{broker}} \tag{5.6}$$

Recall that the index server subsystem (i.e., cluster of index servers) is modeled as a fork-join network. There is no known closed-form solution for fork-join networks with more than 2 queues. Hence, the performance metrics of such networks must be computed using approximation and bounding techniques. An easy lower-bound on the average query residence time in the fork-join subsystem is obtained by ignoring the synchronization delays and considering the average query residence time in the fork-join subsystem equals to the average query residence time at an index server (see Equation 5.5). However, as the number of index servers increases, we expect a significant deviation from this lower-bound due to the synchronization overhead.

A number of approximations for queueing models with fork-join synchronization, with various degrees of complexity and accuracy, are available in the literature (see [3, 22, 25, 34, 40, 54, 57–60, 64] and references within). Nelson and Tantawi [40] propose a very simple

upper-bound on the average response time for fork-join queueing networks, which depends only on the number of index servers p, and on the average query service time and server utilization (see Section 2.2.2). The last two parameters are easily estimated using Equations 5.2 and 5.3. Given the p^{th} harmonic number $H_p = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{p}$, the upper-bound on the average query residence time at the index server subsystem is given by:

$$R^{cluster} \leq H_p \frac{S^{server}}{1 - U^{server}} \tag{5.7}$$

As will be shown in the next section, we found this upper-bound—although quite simple—to yield reasonably accurate estimates of the average query residence time at the index server subsystem. Combining equations 5.5, 5.6 and 5.7, we obtain the following bounds on the average query system response time for our Web search engine:

$$R^{server} + R_p^{broker} \leq R^{system} \leq H_p \frac{S^{server}}{1 - U^{server}} + R_p^{broker} \tag{5.8}$$

Model Validation

A series of validation experiments were executed in dedicated environment consisting of a cluster of 8 homogeneous index servers and a single broker. Each index server runs on a 2.4 gigahertz Pentium IV processor with 256 megabytes of main memory and a 120 gigabytes ATA IDE hard disk. The broker is an ATHLON XP with a 2.2 gigahertz processor and 1 gigabytes of main memory. All of them run the Debian Linux operating system kernel version 2.6. Index servers and broker are connected by a 100 megabits/second high-speed network.

We used a test collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. The collection was uniformly distributed over the 8 index servers, resulting in a local subcollection of size $b = 1.25$ million pages. The query dataset used in our tests is composed of 85,604 queries in a high-load hour of the folded TodoBR dataset which, as discussed in Section 5.2, has traffic characteristics commonly found in other search engine workloads in the world.

The values of the model input parameters were easily obtained by retrieving statistics collected by the Linux operating system and made available at the */proc* pseudo-filesystem, during the experiment. We collected the value of each parameter for each query in our test dataset, averaging the results for each class at the end of the experiment. CPU and disk demands for a query are collected from the */proc/stat* pseudo-file. To estimate the fraction of queries that found all inverted lists in the disk cache (hit_r), we monitored the

total number of sectors successfully retrieved from disk by each query from each class, available in the I/O statistic field of the */proc/diskstats* pseudo-file. Table 5.7 presents the model input parameter values obtained in our experiments. The index server resource demands and hit probabilities are averages for all servers.

Table 5.7: Model input parameter values.

Parameter	Value	
	Class small	Class large
p	($\leq$) 8 servers	
b	1.25 million pages	
S_p^{broker}, $p = 2$	0.33 ms	
S_p^{broker}, $p = 4$	0.39 ms	
S_p^{broker}, $p = 8$	0.52 ms	
$D_{cpu_{hit},r}^{server}$	8.72 ms	12.92 ms
$D_{cpu_{miss},r}^{server}$	6.36 ms	18.71 ms
$D_{disk,r}^{server}$	19.47 ms	46.12 ms
hit_r	0.20	0.11
λ_r/λ	0.73	0.27

Figure 5.7 shows the average query residence time at an index server, averaged over all index servers, as a function of the total query arrival rate. The “estimated" curve represents the results obtained with Equation 5.5, whereas the “measured" curve contains the average query residence times measured in all 8 index servers. As shown, our model captures reasonably well the average performance of a typical index server. For an arrival rate of 28 queries/second, the average utilization of the disk, the bottleneck resource at the index servers, is already almost 75%, and the estimated aggregated utilization of the index server resources (U^{server}) approaches 91%. Thus, the index server is approaching saturation. For this load, the error introduced by our model is only 25%, reasonably small for response time estimates [37].

We now turn to the validation of the average query system response time. Figures 5.8 and 5.9 show experimental results as well as the lower and upper-bounds on the average query system response time, estimated via Equation 5.8, as functions of the arrival rate and of the number of index servers, respectively. In Figure 5.9, in order to make a fair

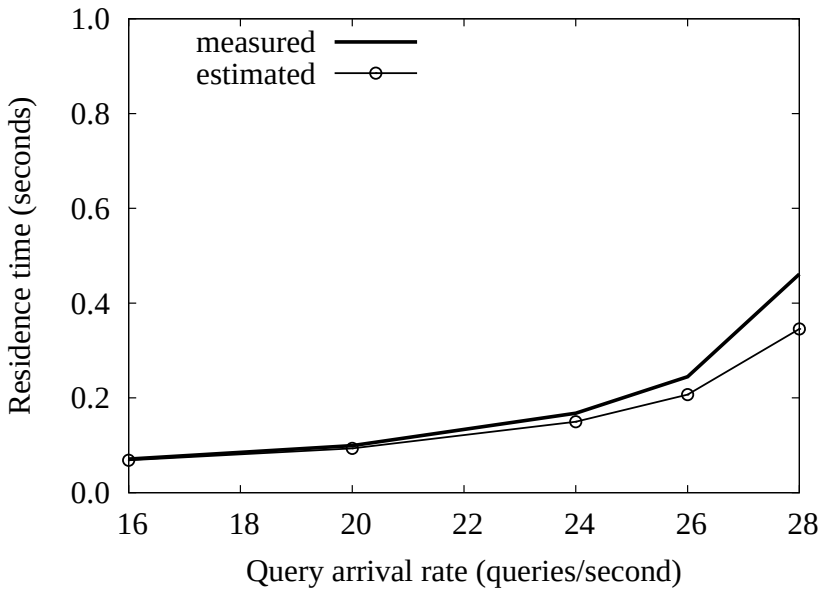


Figure 5.7: Average query residence time at an index server as a function of the query arrival rate ($p = 8$).

comparison, we kept the size of the subcollection b fixed (i.e., 1.25 million pages per index server), varying only the total number of servers p and, indirectly, the size of the total collection $n = pb$.

As the figures show, the lower-bound on the average query system response time is a good approximation for systems with a small number of index servers and/or light loaded servers. However, as either the load or the number of index servers increase, the measured average query system response time deviates significantly from the lower-bound due to the synchronization overhead. This contrast with previous work that disregards imbalance in query service times among homogeneous index servers. In fact, we see that the measured average query system response time approaches the upper-bound for large number of index servers and heavy loads. In particular, for $p = 8$ index servers and an arrival rate $\lambda = 28$ queries/second, the approximation error is only 7%. Therefore, the upper-bound provides a simple-to-compute and yet reasonably accurate approximation of the average query system response time of realistic, typically heavy-loaded, Web search engines.

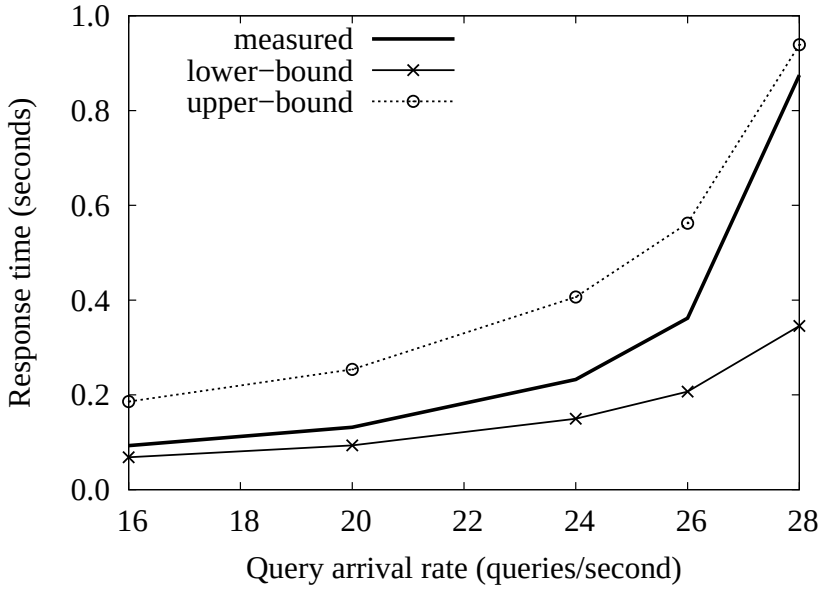


Figure 5.8: Average query system response time as a function of the query arrival rate ($p = 8$).

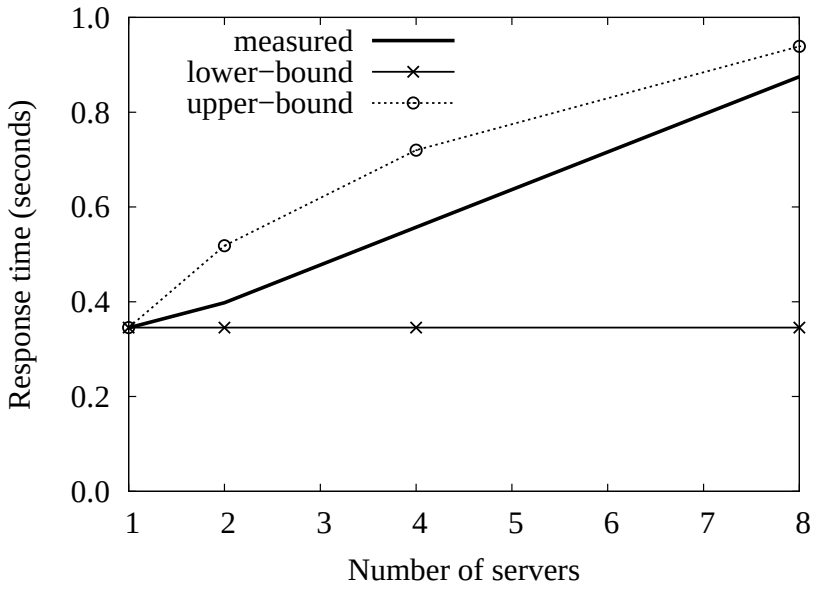


Figure 5.9: Average query system response time as a function of the number of index servers p ($\lambda = 28$ queries/second).

5.3.3 An Example of Model Applicability

In this section, we discuss how to use our model for the capacity planning of large scale Web search engines. This is important because the number of index servers in a real cluster is usually large and maintaining large clusters operational is quite expensive. Furthermore, our model is easy to use and depends only on information that is usually available in both Web search logs and standard operating systems.

We consider "what if" questions, concerning future scenarios for a Web search engine. We assume a collection of 20 billion pages, which is the size of the indexes for Google and Yahoo that has been made public [24]. We also assume that each index server stores a subcollection of size $b = 10$ million pages—the largest collection we have available for experimentation, which requires $2,000$ index servers to host the whole collection. To obtain the parameters of the index server model, we executed experiments in a single index server that runs on a 2.4 gigahertz Pentium IV processor with 1 gigabytes of main memory and a 120 gigabytes ATA IDE hard disk. We used a test collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. Based on these experiments, we determine parameter values for the index server model, such as the average demands for CPU and disk, and the probability of full disk cache hit at an index server for each query class. To obtain the average query service time at the broker as a function of $2,000$ index servers (S_p^{broker}, $p = 2,000$), we fit a straight line to the values of S_p^{broker} ($p = 2, 4, 8$) estimated during our validation experiments (see Table 5.7). We found a rather accurate fitting given by $S_p^{broker} = 3.18e^{-5}p + 0.000265$, which we used to derive $S_p^{broker} = 15.96$ milliseconds for $p = 2,000$. Table 5.8 presents the new parameter values used in our example.

Once the model parameters are computed, one can apply the model to derive the performance metrics of interest. Consider that the manager of the Web search engine wants to guarantee that the average query system response time will not exceed 300 milliseconds. Also consider that the manager has the goal of supporting a query arrival rate of $1,000$ queries/second. Using the model with the parameters described in Table 5.8, we calculate the upper-bound on the average query system response time as a function of the query arrival rate. Figure 5.10 shows the upper-bound on the average query system response time as a function of the query arrival rate. The "baseline" curve represents the results derived by the model with the experimentally estimated parameters (see Table 5.8). The results for the baseline curve indicate that, even at low rates, the response time upper-bound exceeds the threshold defined by the management of the Web search

Table 5.8: New model input parameter values used in our example.

Parameter	Value	
	Class small	Class large
p	2,000 servers	
b	10 million pages	
S_p^{broker}	15.96 ms	
$D_{cpu_{hit},r}^{server}$	27.13 ms	72.33 ms
$D_{cpu_{miss},r}^{server}$	28.44 ms	92.24 ms
$D_{disk,r}^{server}$	41.78 ms	111.39 ms
hit_r	0.30	0.18
λ_r/λ	0.73	0.27

engine. It is interesting to note that the average query residence time at the broker is 7.83% of the upper-bound on the average query system response time at an arrival rate of 1 query/second, and only 1.87% at an arrival rate of 10 queries/second when the system is approaching the point of saturation. Therefore, the average query residence time at the broker in the baseline system is negligible for high query loads.

We want to evaluate what kind of optimization in the resources of machines might yield a reduction in the average query system response time to less than 300 milliseconds. For this, we consider three scenarios. Also in Figure 5.10, the curves labeled "disks 4x", "CPUs 4x", and "CPUs/disks 4x" represent the upper-bound on the average query system response time for the scenarios where disks are four times faster, CPUs are four times faster, and CPUs and disks are both four times faster, respectively. These are the three scenarios analyzed in the following.

Scenario 1 - Disks are four times faster.

In the first scenario, we want to evaluate the impact on query system response time of disks that are four times faster than those in use. This is reflected in the model parameter by dividing the demands for disks by a factor of four. The solution of the model with the new parameters yields the new average response time upper-bound. The results show that the upper-bound on the average query system response time decreases significantly, with gains that reach 7.49 times approximately over the baseline system when it is approach-

ing saturation ($\lambda = 10$ queries/second). We also observe that the supported arrival rate ($\lambda = 15$ queries/second) increases 1.50 times approximately when compared to the arrival rate supported by the baseline system ($\lambda = 10$ queries/second). Nevertheless, the upper-bound still exceeds the defined threshold even at light loads. We note that the average query residence time at the broker is 12.05% of the upper-bound on the average query system response time at an arrival rate of 1 query/second, and 31.91% at an arrival rate of 15 queries/second when the system is approaching the saturation point. Although it is relatively low compared to the average query system response time, the average query residence time at the broker in this scenario is not negligible for high query loads.

Scenario 2 - CPUs are four times faster.

In the second scenario, we want to assess the impact on query system response time of CPUs that are four times faster. For modeling the new CPUs, we divide the demands for CPUs by a factor of four. Using the model, we calculate the new average response time upper-bound. The results indicate that the configuration with faster CPUs slightly outperforms the configuration with faster disks, with gains of 8.26 times approximately over the baseline system when it is approaching saturation ($\lambda = 10$ queries/second). We also observe that the supported arrival rate ($\lambda = 16$ queries/second) increases 1.60 times approximately when compared to the arrival rate supported by the baseline system ($\lambda = 10$ queries/second). Nevertheless, the upper-bound still exceeds the defined threshold even at light loads (as in Scenario 1). We point out that the average query residence time at the broker is 3.13% of the upper-bound on the average query system response time at an arrival rate of 1 query/second, and only 0.33% at an arrival rate of 16 queries/second when the system is approaching saturation. Therefore, the average query residence time at the broker in this scenario is negligible at any query load.

Scenario 3 - CPUs and disks are both four times faster.

In the third scenario, we want to verify the impact on query system response time of CPUs and disks that are both four times faster. For modeling the new resources, we divide the demands for CPUs and disks by four. The solution of the model with the new parameters yields the new average response time upper-bound. The results indicate that the average query system response time is less than 297 milliseconds at an arrival rate of 14 queries/second, which satisfies the service level objective for the Web search engine of 300 milliseconds per query. The results also show a remarkable reduction in the upper-bound on the average query system response time, with gains that reach

35.90 times approximately over the baseline system when it is approaching the saturation point ($\lambda = 10$ queries/second). Further, we observe an expressive increase in the supported arrival rate ($\lambda = 42$ queries/second), with gains that reach 4.20 times approximately over the baseline system ($\lambda = 10$ queries/second). We note that the average query residence time at the broker is 7.78% of the upper-bound on the average query system response time at an arrival rate of 1 query/second, 6.91% at an arrival rate of 14 queries/second, and only 0.90% at an arrival rate of 42 queries/second when the system is approaching the point of saturation. Therefore, the average query residence time at the broker in this scenario is negligible at high query loads (as in the baseline system).

We still have to meet the objective of supporting an arrival rate of $1,000$ queries/second. For supporting a higher query arrival rate, the cluster of index servers is usually replicated [14,18,45]. Replication involves relatively small overheads, and approximately linear gains in the query arrival rate supported can be expected as a function of the number of mirrored systems. The objective of supporting an arrival rate of $1,000$ queries/seconds can be achieved by creating 72 replicas of the cluster of $2,000$ index servers, each replica supporting an arrival rate of 14 queries/second and guaranteeing a query system response time of 297 milliseconds. Therefore, our model indicates that a cluster composed of $144,000$ index servers (72 cluster replicas $\times$ $2,000$ index servers in a cluster) would achieve the desired performance. Some speculations suggest that large scale Web search engines may indeed adopt clusters with several thousands of machines, but, to the best of our knowledge, there is no publicly available data to back up this information. If each index server handles a larger subcollection, the number of total servers in a cluster could be smaller. Thus, one must be cautious before extrapolating our illustrative results to an arbitrary cluster of any Web search engine. In this case, for other document collections and machines, the parameters of our model could be estimated experimentally.

This example shows how a reasonably accurate capacity planning model provides a very useful tool for the proper management of modern cluster-based Web search engines. Recall that our goal is to have a simple and reasonably accurate model that can answer three important capacity planning questions stated in Section 1.2. The first question is fully answered during our validation experiments (see Section 5.3.2), while the second and third questions are not completely answered in this example because the model predictions are not validated experimentally.

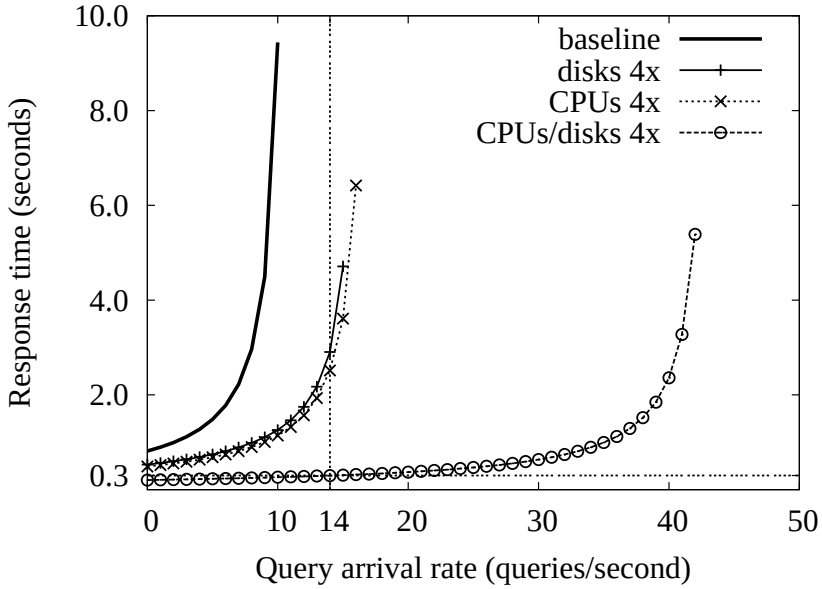


Figure 5.10: Upper-bound on the average query system response time as a function of the query arrival rate derived in our example.

5.4 Concluding Remarks

In this chapter, we proposed a capacity planning model for Web search engines that considers the imbalance in query service times among homogeneous index servers. Our model, which relies on a queueing-based analytical model, is simple and yet reasonably accurate. To estimate the parameters of our model, we ran experiments on a small cluster of index servers. Once the key parameters were estimated, we verified the accuracy of our model by comparing its predictions with the results measured empirically and found great concordance. Finally, we illustrated how to apply our model to predict query response time when adopting faster CPUs and disks than those in use. We considered a realistic scenario, where a collection of 20 billion documents is distributed over $2,000$ index servers. In our example, we showed that the manager of the search engine can quickly reach predictions for upper-bounds on the average query system response time without having to run any live experiments.

Given the reasonable accuracy of the model in estimating average query system response time, and its simplicity to configure and to apply in practical scenarios, we believe that it is useful to predict the behavior of modern cluster-based Web search engines.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize our main findings and present final conclusions in Section 6.1. We also outline a number of directions for future work in Section 6.2.

6.1 Conclusions

In this thesis, we provided a performance framework for the design and analysis of the infrastructure of Web search engines. In this framework we (i) investigated and analyzed the imbalance among homogeneous index servers in a cluster for parallel query processing; (ii) proposed a capacity planning model for Web search engines; and (iii) characterized the query workload of four different real-world Web search engines.

6.1.1 Analyzing Imbalance among Homogeneous Index Servers

As an outcome of our analysis of the imbalance issue, we verified that the idealized scenario, that supposes balanced service times as a consequence of an uniform data distribution among homogeneous index servers, is unlikely to be found in practice. This is an important contribution because it sheds light on the usual assumption of balanced service times taken by many theoretical models in the literature to simplify their modeling task [20,23,44]. Our findings derive from a comprehensive experimental analysis using an information retrieval testbed and real data obtained from a real-world search engine. Besides verifying the presence of a certain level of imbalance among homogeneous index servers, we have also identified and characterized the main sources for this unexpected imbalance.

The key factor for imbalance is the use of disk cache at the different index servers. We verified that imbalance for each query increases with the number of index servers that retrieve the needed documents from the disk cache. On the one hand, the worst case for imbalance is achieved when a single index server must actually access the disk for documents while all remaining servers are using the disk cache for the same query. The worst case presents a much higher average imbalance of 3.45—computed as a ratio between the maximum and the average service time among index servers—because in this case a single server has a service time much larger than the corresponding service times at all the remaining servers, thus resulting in a high imbalance. On the other hand, the best case to avoid imbalance results in an average imbalance of 1.08. It is achieved when all index servers operate in the cache region, thus leading to a relatively small and similar disk access times throughout the cluster of servers.

Other identified source of imbalance is the size of the main memory of the homogeneous index servers, which affects the availability of resources for disk cache at servers. We verified that the average imbalance decreases as the size of main memory increases, but this happens with a diminishing return with an increase in memory capacity. When the main memory size is relatively large as compared to the size of the local index stored at the index servers, all servers retrieve data from their disk caches for a high percentage of queries, which is the best case scenario that produces the lowest imbalance. In contrast, considering a relatively small main memory capacity available for disk cache, for a given query some index servers might access their disk caches while the others might need to actually access the disk, thus leading to a higher imbalance. In our cluster with 7 index servers, the average imbalance is 1.32 and 1.18 for 200 and 400 megabytes of main memory at each server, respectively.

Another identified source of imbalance is the number of index servers in the cluster. We verified that, for a fixed size of main memory, the average imbalance in service times of index servers increases with the number of servers in the cluster. The reason is that the larger the number of index servers participating in the parallel query processing, the higher the probability of some servers to operate in the cache region and, as a consequence, the higher the imbalance in service times of those servers. For 200 megabytes of main memory, the average imbalance is 1.09 and 1.32 in our cluster with 2 and 7 index servers, respectively.

Overall, the primary source of imbalance per query is the heterogeneous use of disk cache among the homogeneous index servers. Main memory size at index servers and the number of servers are actually indirect sources of imbalance because they cause heterogeneous use of disk cache.

6.1.2 A Capacity Planning Model for Web Search Engines

We also proposed a capacity planning model for Web search engines that considers the imbalance in query service times among homogeneous index servers. Our model, based on queueing theory, is simple and reasonably accurate. To fine tune the model, we ran experiments on a small cluster of index servers. Once tuned, we compared the predictions of our model with the results we measured empirically and found a high quality matching. Even at the saturation point, the predictions of our model were reasonably accurate.

Following, we illustrated how to apply our model to predict average query system response time when adopting faster CPUs and disks than those in use. We considered a realistic scenario, where a collection of 20 billion documents is distributed over $2,000$ index servers. In our example, we showed that the manager of the search engine can quickly reach predictions for upper-bounds on the average query system response time without having to run any live experiments.

Given the complexity of maintaining large scale Web search engines, and the simplicity and reasonable accuracy of our model, we believe our model can be quite useful in practice.

6.1.3 Query Workload Characterization of Four Web Search Engines

We further characterized four query datasets composed of millions of queries representing the query workload of four different real-world Web search engines, namely TodoBR, Radix, AllTheWeb, and Altavista. A subset of the TodoBR query dataset is used in the experiments to validate the capacity planning model for Web search engines. We verified that—although the query datasets cover different time periods, query loads, users, and languages—the distribution of queries and of terms in queries throughout the datasets are quite similar and follow a Zipf's distribution. Also, our results confirmed the prevalence of very short queries—i.e., queries containing two or fewer terms—for all datasets. Further, a periodic behavior on the query workload was consistently observed through the four considered query datasets. The query workload clearly presents daily load variations. In particular, working days present similar loads among them that are different from those observed on weekend days. Finally, we characterized the interarrival process of the folded version of the TodoBR dataset, which is used as input stream in the validation experiments for the capacity planning model for Web search engines. We verified that the Exponential distribution approximates the observed interarrival process of queries within quite acceptable bounds.

6.2 Future Work

A direction for future work is to extend our capacity planning model to support multiple processing threads at index servers. Other direction for research is to improve our model to estimate the distribution function of the query system response time of a cluster of index servers. From this distribution, one can find out its percentiles. This solution is useful if the management of the Web search engine requires the q-percentile of the expected query system response time to be less or equal than a threshold.

Another direction for further research is to model caching of query results—that allows the search engine to answer recently repeated queries at a very low cost since it is not necessary to process those queries—and caching of the inverted lists of query terms—that improves the query processing time for the new queries that include at least one term whose list is cached [48]. Another direction for research is to identify and analyze the reasons for the use of disk cache, and eventually model the probability of disk cache hit. Some of the elements that affect the use of disk cache are the temporal locality of queries and terms of queries in the input traffic, the spatial locality of the inverted lists of query terms in the disk, and the size of the main memory.

Another direction for future work is to verify by simulation the accuracy of our model predictions for large clusters with thousands of index servers for supporting a collection with billions of documents, as illustrated in Section 5.3.3. It would be important to consider a cluster with p index servers, such that p is large enough to store the index for a collection of 20 billion pages, which is the size of the indexes for Google and Yahoo that has been made public [24]. A difficulty is that it is hard to obtain large datasets for the collection of documents used by Web search engines to generate an answer page for each received query. This is so because the set of collected documents is seen as strategic and proprietary information by the major Web search operators. In fact, during the course of this thesis, we only had access to a document collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. A solution would be to generate a large synthetic document collection from probability distributions that are based on statistics from real datasets.

Another direction for further research is to develop an approach for finding the optimal architecture for a Web search engine in terms of cost, that combines partitioning and replication strategies to satisfy operational requirements for query response time, query throughput, and server utilization. It is important to establish constraints on the utilization of index servers to avoid wasting machine resources. By cost we mean the number of index

servers in the architecture. Therefore, the goal would be to satisfy the three operational requirements with the fewest number of index servers. On the one hand, for scaling on data volume, the index of a document collection can be partitioned among a set of index servers organized in a computational cluster for parallel query processing. On the other hand, for scaling on query throughput, the cluster of index servers can be replicated, and a load-balancing system can be used to direct queries to one of a set of mirrored clusters by accounting their available capacity. Replication involves relatively small overheads, and approximately linear gains in throughput capacity can be expected as a function of the number of mirrored systems. Partitioning and replication strategies are usually combined by Web search engines to handle huge document collections and high query loads while satisfying service level objectives.

Given operational requirements of query response time, query throughput, and server utilization, the goal would be to minimize the needed number of index servers. This goal can be formalized by minimizing a cost function:

$$c = p \times r \tag{6.1}$$

while meeting the following operational constraints

$$\begin{aligned} f_t(p, r, X) &\leq T \\ f_u(p, r, X) &\leq U \end{aligned} \tag{6.2}$$

where r is the number of replications (of a cluster of index servers); p is the number of index partitions (across the index servers in a cluster); T is the response time requirement; U is the utilization requirement; X is the throughput requirement; and $f_t(p, r, X)$ and $f_u(p, r, X)$ calculate the response time and utilization, respectively, for a given throughput X and a given architecture where the index is partitioned into p divisions and replicated r times.

Bibliography

[1] AllTheWeb. Available at `http://www.alltheweb.com`.

[2] Altavista. Available at `http://www.altavista.com`.

[3] F. Baccelli, W. A. Massey, and D. Towsley. Acyclic fork-join queuing networks. *Journal of the ACM (JACM)*, 36(3):615–642, 1989.

[4] C. S. Badue, R. Baeza-Yates, W. Meira Jr., B. Ribeiro-Neto, and N. Ziviani. Distributed architecture for information retrieval. In *Proceedings of the 1st International Seminar on Advanced REsearch in E-Business (EBR´02)*, pages 114–122, Rio de Janeiro, RJ, Brazil, November 2002.

[5] C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Analyzing imbalance among homogeneous index servers in a web search system. *Information Processing & Management (IP&M)*, 43(3):592–608, 2007.

[6] C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 10–20, Laguna de San Rafael, Chile, 2001. IEEE Computer Society.

[7] C. S. Badue, R. Barbosa, P. Golgher, B. Ribeiro-Neto, and N. Ziviani. Basic issues on the processing of web queries. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 577–578, Salvador, BA, Brazil, 2005.

[8] C. S. Badue, R. Barbosa, P. Golgher, B. Ribeiro-Neto, and N. Ziviani. Distributed processing of conjunctive queries. In *ACM SIGIR Workshop on Heterogeneous and Distributed Information Retrieval*, Salvador, BA, Brazil, 2005.

[9] R. Baeza-Yates. Query usage mining in search engines. In A. Scime, editor, *Web Mining: Applications and Techniques*, pages 307–321. Idea Group, 2004.

[10] R. Baeza-Yates. Applications of web query mining. In D. Losada and J.Fernández-Luna, editors, *Proceedings of the European Conference on Information Retrieval (ECIR'05)*, pages 7–22, Santiado de Compostela, Spain, 2005.

[11] R. Baeza-Yates, C. Hurtado, M. Mendoza, and G. Dupret. Modeling user search behavior. In *Proceedings of the 3rd Latin American Web Congress (LA-WEB'05)*, pages 242–251, Buenos Aires, Argentina, October 2005. IEEE CS Press.

[12] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press New York, Addison Wesley, 1999.

[13] S. Balsamo. Product form queueing networks. In *Performance Evaluation*, pages 377–401, 2000.

[14] L. A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[15] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975.

[16] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'04)*, pages 321–328, Sheffield, UK, 2004. ACM Press New York, NY, USA.

[17] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[18] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[19] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[20] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *Information Processing & Management (IP&M)*, 41(5):1141–1161, 2004.

[21] M. Chau, X. Fang, and O. R. L. Sheng. Analysis of the query logs of a web site search engine. *Journal of the American Society for Information Science and Technology (JASIST)*, 56(13):1363–1376, 2005.

[22] R. J. Chen. A hybrid solution of fork/join synchronization in parallel queues. *IEEE Transactions on Parallel and Distributed Systems*, 12(8):829–845, 2001.

[23] A. Chowdhury and G. Pass. Operational requirements for scalable search systems. In *Proceedings of the ACM 12th Conference on Information and Knowledge Management (CIKM'03)*, pages 435–442, New Orleans, LA, USA, 2003.

[24] K. J. Delaney. Yahoo contends it tops google in number of pages searched. Wall Street Journal, August 15, 2005, page B4.

[25] L. Flatto and S. Hahn. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 44(5):1042–1053, 1984.

[26] Google. Available at `http://www.google.com`.

[27] D. Harman. Overview of the third text retrieval conference. In *Proceedings of the 3rd Text REtrieval Conference (TREC-3)*, pages 1–19, Gaithersburg, MD, USA, 1994.

[28] D. Hawking, N. Craswell, and P. Thistlewaite. Overview of TREC-7 very large collection track. In *Proceedings of the 7th Text REtrieval Conference (TREC-7)*, pages 257–268, Gaithersburg, MD, U.S.A., 1998.

[29] JupiterMedia Corporate Information. JupiterResearch forecasts online advertising market to reach 18.9 billion by 2010; search advertising revenue to surpass display. Available at `http://www.jupitermedia.com/corporate/releases/05.08.15-newjupresearch2.html`, Press Release, August 15, 2005.

[30] iProspect. iProspect search engine user attitudes survey. Available at `http://www.iprospect.com/premi-umPDFs/iProspectSurveyComplete.pdf`, April-May 2004.

[31] B. S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, 1995.

[32] N. Kammenhuber, J. Luxenburger, A. Feldmann, and G. Weikum. Web search clickstreams. In *Proceedings of the 6th ACM SIGCOMM on Internet Measurement*, pages 245–250, Rio de Janeiro, RJ, Brazil, 2006. ACM Press New York, NY, USA.

[33] Z. Lu, K. S. McKinley, and B. Cahoon. The hardware/software balancing act for information retrieval on symmetric multiprocessors. In *Proceedings of the 4th International Euro-Par Conference on Parallel Processing (Euro-Par '98)*, pages 521–527, London, UK, 1998. Springer-Verlag.

[34] J. C. S. Lui, R. R. Muntz, and D. Towsley. Computing performance bounds of fork-join parallel programs under a multiprocessing environment. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):295–311, 1998.

[35] A. MacFarlane, J. A. McCann, and S. E. Robertson. Parallel search using partitioned inverted files. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, pages 209–220, La Coruña, Spain, 2000. IEEE Computer Society.

[36] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.

[37] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy. *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[38] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*, pages 348–355, Seattle, WA, USA, 2006.

[39] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*. To appear.

[40] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, 1988.

[41] S. Orlando, R. Perego, and F. Silvestri. Design of a parallel and distributed web search engine. In *Proceedings of the 2001 Parallel Computing Conference (ParCo'01)*, pages 197–204, Naples, Italy, 2001. Imperial College Press.

[42] Radix. Available at `http://www.radix.com.br`.

[43] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)*, 27(2):313–322, 1980.

[44] B. A. Ribeiro-Neto and R. A. Barbosa. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 182–190, Pittsburgh, PA, USA, 1998.

[45] K. M. Risvik, Y. Aasheim, and M. Lidal. Multi-tier architecture for web search engines. In *Proceedings of the 1st Latin American Web Congress (LA-WEB'03)*, pages 132–143, Santiago, Chile, November 2003.

[46] D. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of the 13th International World Wide Web Conference (WWW'04)*, pages 13–19, New York, NY, USA, 2004. ACM Press.

[47] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[48] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR´01)*, pages 51–58, New Orleans, Louisiana, USA, 2001.

[49] K. C. Sevcik and I. Mitrani. The distribution of queuing network states at input and output instants. *Journal of the ACM (JACM)*, 28(2):358–371, 1981.

[50] C. Silverstein, M. Henzinger, H. Marais, and M. Moriez. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[51] O. Sornil and E. A. Fox. Hybrid partitioned inverted indices for large-scale digital libraries. In *Proceedings of the 4th International Conference of Asian Digital Libraries*, Bangalore, India, 2001.

[52] A. Spink, B. J. Jansen, and H. C. Ozmultu. Use of query reformulation and relevance feedback by excite users. *Internet Research: Electronic Networking Applications and Policy*, 10(4):317–328, 2000.

[53] A. Spink, D. Wolfram, B. J. Jansen, and T. Saracevic. Searching the web: the public and their queries. *Journal of the American Society for Information Science (JASIS)*, 52(3):226–234, 2001.

[54] A. Thomasian and A. N. Tantawi. Approximate solutions for M/G/1 fork/join synchronization. In *Proceedings of the 26th Conference on Winter Simulation (WSC '94)*, pages 361–368, San Diego, CA, USA, 1994. Society for Computer Simulation International.

[55] TodoBR. Available at `http://www.todobr.com.br`.

[56] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS'93)*, pages 8–17, San Diego, CA, USA, 1993.

[57] E. Varki. Mean value technique for closed fork-join networks. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*, pages 103–112, New York, NY, USA, 1999. ACM Press.

[58] E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, 2001.

[59] E. Varki and L. W. Dowdy. Analysis of balanced fork-join queueing networks. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 232–241, Philadelphia, PA, USA, 1996.

[60] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. In *Performance Evaluation*, pages 245–265, 1994.

[61] W. Webber and A. Moffat. In search of reliable retrieval experiments. In J. Kay, A. Turpin, and R. Wilkinson, editors, *Proceedings of the 10th Australasian Document Computing Symposium*, pages 26–33, Sydney, Australia, 2005.

[62] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1999.

[63] Yahoo. Available at `http://www.yahoo.com`.

[64] K. H. Yeung and T. S. Yum. State reduction in the exact analysis of fork/join queueing systems with homogeneous exponential servers. In *Proceedings of the 1996 International Conference on Parallel and Distributed Systems (ICPADS'96)*, pages 57–, 1996.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo