

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA



Estrategias de planificación para motores de búsqueda verticales

Danilo Fernando Bustos Pérez

Profesor Guía: Dra. Carolina Bonacic Castro
Profesor Co-guía: Dr. Mauricio Marín Caihuán

Trabajo de Titulación presentado en conformidad
a los requisitos para obtener el Título de
Ingeniero Civil Informático

SANTIAGO DE CHILE
2013

© Danilo Fernando Bustos Pérez

Se autoriza la reproducción parcial o total de esta obra, con fines académicos, por cualquier forma, medio o procedimiento, siempre y cuando se incluya la cita bibliográfica del documento.

AGRADECIMIENTOS

Dedicado a

RESUMEN

Resumen en Castellano

Palabras Claves: keyword1, keyword2 .

ABSTRACT

Resumen en Inglés

Keywords: keyword1, keyword2 .

ÍNDICE DE CONTENIDOS

Índice de Figuras	iii
-------------------	-----

Índice de Tablas	iv
------------------	----

1. Introducción	1
------------------------	----------

1.1. Antecedentes y motivación	2
1.2. Descripción del problema	2
1.3. Objetivos y solución propuesta	2
1.3.1. Objetivo General	2
1.3.2. Objetivos Específicos	2
1.3.3. Alcances	2
1.3.4. Solución propuesta	2
1.3.5. Características de la solución	2
1.3.6. Propósito de la solución	2
1.4. Metodología y herramientas de desarrollo	2
1.4.1. Metodología	2
1.4.2. Herramientas de desarrollo	2
1.5. Resultados obtenidos	2
1.6. Organización del documento	2

2. Marco teórico	3
-------------------------	----------

2.1. Motores de búsqueda verticales	3
2.2. Índice invertido	5
2.3. Estrategias de evaluación de <i>queries</i>	6

<i>ÍNDICE DE CONTENIDOS</i>	ii
2.3.1. TAAT	6
2.3.2. DAAT	7
2.3.3. Consideraciones	7
2.4. Operaciones sobre listas invertidas	8
2.4.1. <i>OR</i>	8
2.4.2. AND	9
2.4.3. WAND	10
2.5. <i>Ranking</i>	11
2.5.1. TF-IDF	12
2.5.2. BM25	13
3. <i>Scheduling</i> en motores de búsqueda	14
4. Conclusiones	18
Referencias	19

ÍNDICE DE FIGURAS

2.1. Arquitectura típica de un motor de búsqueda	4
2.2. Índice invertido	6
2.3. Operación OR	9
2.4. Operación AND	9
2.5. Operación AND	11
3.1. Arquitectura de un sistema de recuperación de la información con réplicas . . .	16

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

1.2 DESCRIPCIÓN DEL PROBLEMA

1.3 OBJETIVOS Y SOLUCIÓN PROPUESTA

1.3.1 Objetivo General

1.3.2 Objetivos Específicos

1.3.3 Alcances

CAPÍTULO 2. MARCO TEÓRICO

En este capítulo se exponen los conceptos teóricos del presente trabajo de tesis. Primero se explica qué es un motor de búsqueda vertical. Luego se definen las estrategias de evaluación de *queries*. Posteriormente se describen las diferentes operaciones sobre listas invertidas. Finalmente, se explica el concepto de *ranking*.

2.1 MOTORES DE BÚSQUEDA VERTICALES

A medida que pasa el tiempo y la Web sigue creciendo, los motores de búsqueda se convierten en una herramienta cada vez más importante para los usuarios. Estas máquinas ayudan a los usuarios a buscar contenido dentro de la Web, puesto que conocen en cuales documentos de la Web aparecen qué palabras. Si estas máquinas no existieran, los usuarios estarían obligados a conocer los localizadores de recursos uniformes (URL) de cada uno de los sitios a visitar. Además, los motores de búsquedas en cierto modo conectan la Web, ya que existe un gran número de páginas Web que no tienen referencia desde otras páginas, siendo el único modo de acceder a ellas a través de un motor de búsqueda.

Un motor de búsqueda está construido por diversos componentes. Su arquitectura típica se puede ver en la Figura 2.1. Existe un proceso denominado *crawling*, éste posee una tabla con los documentos Web iniciales en los que se extrae el contenido de cada uno de ellos. A medida que el *crawler* comienza a encontrar enlaces a otros documentos Web, la tabla de documentos a visitar crece. El contenido que se extrae en el procedimiento de *crawling* es enviado al proceso de

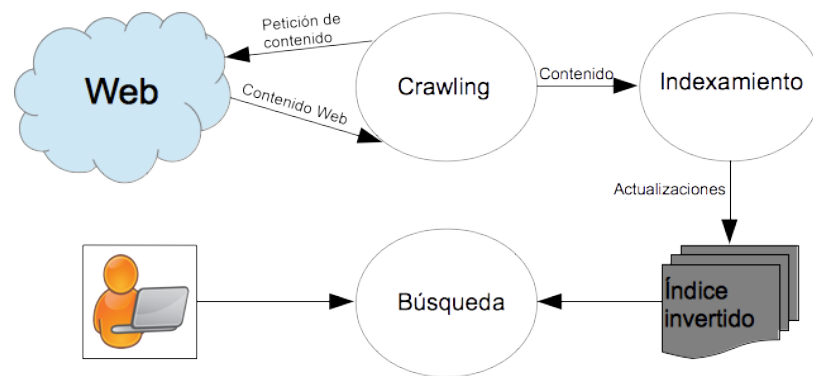


FIGURA 2.1: Arquitectura típica de un motor de búsqueda

indexamiento, este se encarga de crear un índice de los documentos ya visitados por el *crawler*.

Dado el volúmen de datos involucrado en el procesamiento, se debe tener una estructura de datos que permita encontrar cuáles documentos contienen las palabras presentes en la búsqueda que llega al sistema. Todo esto dentro de un período de tiempo aceptable. El índice invertido (Zobel & Moffat, 2006) es una estructura de datos que contiene una lista con todas las palabras que el proceso de *crawling* ha visto. Asociado a cada palabra se tiene una lista de todos los documentos Web donde ésta palabra aparece mencionada. El motor de búsqueda construye esta estructura con el objetivo de acelerar el proceso de las búsquedas que llegan al sistema. El proceso de búsqueda es el encargado de recibir la consulta (*query*), generar un *ranking* de los documentos Web que contienen las palabras de la *query* y finalmente generar una respuesta. Las diversas formas de calcular la relevancia de un documento será explicado en secciones posteriores.

En un motor de búsqueda se pueden encontrar diversos servicios tales como (a) cálculo de los mejores documentos Web para una cierta *query*; (b) construcción de la página Web en la que se mostrará al usuario los resultados de la *query*; (c) publicidad relacionada con las *queries*; (e) sugerencia de *queries*; entre muchos otros servicios.

En los sistemas de recuperación de la información modernos como los motores de búsqueda, lo que se hace hoy en día es agrupar computadores para procesar una *query* y

obtener la respuesta para ésta. Este conjunto de computadores recibe el nombre de *cluster*.

La diferencia entre un motor de búsqueda vertical y uno general, es que el primero se centra solo en un contenido específico de la Web. El *crawler* debe extraer contenido solo de aquellas páginas Web que están dentro del dominio permitido. Al ser un dominio acotado, los documentos Web a procesar serán menos y por lo tanto, la lista de los términos del índice invertido serán eventualmente de menor tamaño. Sin embargo, en un motor de búsqueda vertical las actualizaciones al índice invertido ocurren con mayor frecuencia.

2.2 ÍNDICE INVERTIDO

Es una estructura de datos que contiene todos los términos (palabras) encontrados por el *crawler*. A cada uno de los términos, está asociado una lista de documentos (páginas Web) que contienen dicho término. Adicionalmente, se almacena información que permita realizar el *ranking* de documentos para generar la respuesta a las *queries* que llegan al sistema, por ejemplo, el número de veces que aparece el término en el documento. Esta lista recibe el nombre de lista invertida.

Para construir un índice invertido se debe procesar cada palabra que existe en un documento Web, registrando su posición y la cantidad de veces que éste se repite. Cuando se procesa el término con la información asociada correspondiente, se almacena en el índice invertido (ver Figura 2.2).

El tamaño del índice invertido crece rápido y eventualmente la memoria RAM se agotará antes de procesar toda la colección de documentos. Cuando la memoria RAM se agota, se almacena en disco el índice parcial hasta aquel momento, se libera la memoria y se continúa

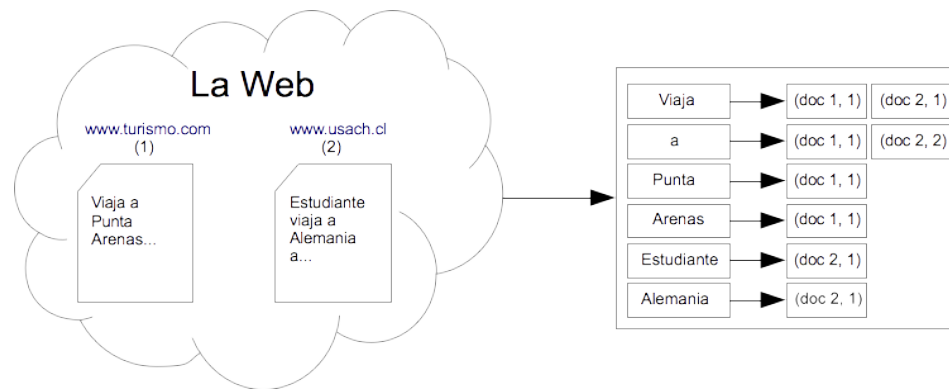


FIGURA 2.2: Índice invertido

con el proceso. Además, se debe hacer un *merge* de los índices parciales uniéndolos las listas invertidas de cada uno de los términos involucrados.

2.3 ESTRATEGIAS DE EVALUACIÓN DE *QUERIES*

Existen dos principales estrategias para encontrar los documentos y calcular sus respectivos puntajes de una determinada *query*. Estas son (a) *term-at-a-time* (TAAT) y (b) *document-at-a-time* (DAAT).

2.3.1 TAAT

Este tipo de estrategia procesa los términos de las *queries* uno a uno y acumula el puntaje parcial de los documentos. Las listas invertidas asociadas a un término son procesadas

secuencialmente, esto significa que los documentos presente en la lista invertida del término t_i , obtienen un puntaje parcial antes de comenzar el procesamiento del término t_{i+1} . La secuencialidad en este caso es con respecto a los términos contenidos en la *query*.

2.3.2 DAAT

En este tipo de estrategias se evalúa la contribución de todos los términos de la *query* con respecto a un documento antes de evaluar el siguiente documento. Las listas invertidas de cada término de la *query* son procesadas en paralelo, de modo que el puntaje del documento d_j se calcula considerando todos los términos de la *query* al mismo tiempo. Una vez que se obtiene el puntaje del documento d_j para la *query* completa, se procede al procesamiento del documento d_{j+1} .

2.3.3 Consideraciones

Cuando se tiene un índice invertido pequeño, las estrategias TAAT rinden adecuadamente, sin embargo cuando los índices invertidos son de gran tamaño las estrategias DAAT poseen dos grandes ventajas: (a) Requieren menor cantidad de memoria para su ejecución, ya que el puntaje parcial por documento no necesita ser guardado y (b) Explotan el paralismo de entrada y salida (I/O) más eficientemente procesando las listas invertidas en diferentes discos simultáneamente. Además existen técnicas para optimizar el proceso de las estrategias recientemente descritas

(Turtle & Flood, 1995).

2.4 OPERACIONES SOBRE LISTAS INVERTIDAS

Cuando una *query* llega al motor de búsqueda, cada término tiene asociado una lista con todos los documentos en los cuales aparece. El sistema debe decidir qué documentos se analizarán para obtener la respuesta con el conjunto de los K mejores. A continuación se presentan las diferentes formas de operar las listas invertidas de una *query*.

2.4.1 OR

Este operador toma las listas invertidas de cada uno de los términos de la *query* y ejecuta la disyunción entre ellas. El resultado de este operador es una lista invertida con todos los documentos que contengan al menos un término de la *query*. Finalmente, esta lista invertida se ocupará para obtener los mejores K documentos. Un simple ejemplo se muestra en la FIGURA 2.3.

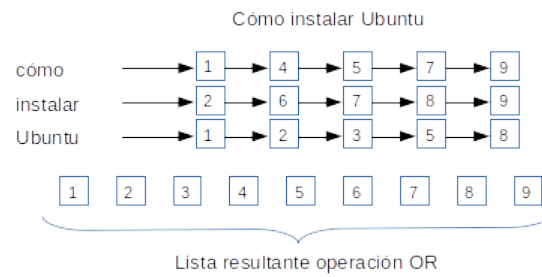


FIGURA 2.3: Operación OR

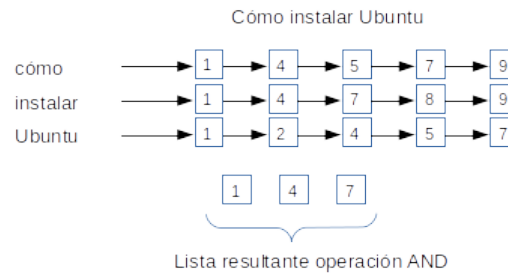


FIGURA 2.4: Operación AND

2.4.2 AND

Este operador ejecuta la conjunción entre las listas invertidas de los términos de una *query*. Se obtiene una lista invertida con los documentos que contengan todos los términos de la *query*. Se debe notar que aquí se obtiene una lista resultante de menor tamaño que la obtenida en el operador OR (Ver FIGURA 2.4).

2.4.3 WAND

Método de evaluación de *queries* para obtener eficientemente el conjunto de K documentos que mejor satisfacen una *query* dada. *WAND* (Broder et al., 2003) (*Weak AND*) es un proceso menos estricto que el método *AND* y es basado en dos niveles. Dentro del proceso de evaluación de una *query*, uno de los procesos más costoso en términos de tiempo es el proceso de *scoring*. Este proceso corresponde a entregarle a cada uno de los documentos analizados, un puntaje que representa la relevancia del documento para una *query* dada, esto se denomina evaluación completa o cálculo del puntaje exacto del documento.

El objetivo de *WAND* es minimizar la cantidad de evaluaciones completas de los documentos ejecutando un proceso de dos niveles. En el primer nivel se intenta omitir rápidamente grandes porciones de las listas invertida, lo que se traduce en ignorar el cálculo del puntaje exacto de grandes cantidades de documentos. El intento de omitir el cálculo completo de un documento es debido a que en motores de búsqueda a gran escala, este es un proceso que requiere de mucho tiempo para llevarse a cabo y depende de factores como la cantidad de ocurrencia del término dentro del documento, el tamaño del documento, entre otros.

En el primer nivel se itera sobre los documentos del índice invertido de cada término y se identifica candidatos usando una evaluación aproximada. En el segundo nivel, aquellos documentos candidatos son completamente evaluados y su puntaje exacto es calculado. De esta forma se obtiene el conjunto final de documentos.

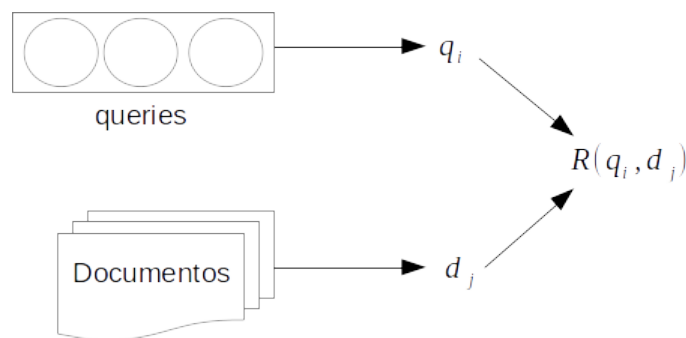


FIGURA 2.5: Operación AND

2.5 RANKING

Los sistemas de recuperación de información como los motores de búsqueda deben ejecutar un proceso el cual asigna un puntaje a documentos con respecto a una determinada *query*, este proceso se denomina *ranking* (Baeza-Yates & Ribeiro-Neto, 2011). Como se puede ver en la Figura 2.5, este proceso toma como entrada la representación de las *queries* y documentos, y asigna un puntaje (*score*) a un documento d_j dada una *query* q_i .

Un motor de búsqueda guarda billones de documentos que están formados por términos o palabras, estos términos no todos poseen la misma utilidad para describir el contenido del documento. Determinar la importancia de una palabra en un documento no es tarea sencilla, para ello se asocia un peso positivo $w_{i,j}$ a cada término t_i del documento d_j . De esta forma, para un término t_i que no aparezca en el documento d_j se tendrá $w_{i,j} = 0$. La asignación de pesos a los términos permite generar un *ranking* numérico para cada documento en la colección.

2.5.1 TF-IDF

El tf-idf (*term frequency - inverse document frequency*) es un estadístico que tiene por objetivo reflejar cuán importante es una palabra para un documento en una colección o corpus. Este estadístico se divide en dos partes, el primero corresponde a la frecuencia de la palabra o término en un documento (tf) y que en su versión más sencilla se utiliza la frecuencia bruta del término t en el documento d ($f(t,d)$). El segundo término corresponde a la frecuencia inversa de documento (idf) y se utiliza para observar si es que el término es común en el corpus. El idf obtiene calculando el logaritmo de la división entre el número total de documentos del corpus y el número de documentos que contienen el término. De esta forma se tiene:

$$tf(t, d) = \frac{f(t, d)}{\max f(w, d) : w \in d}$$

$$idf(t, D) = \log \frac{|D|}{1 + |d \in D : t \in d|}$$

$$tfidf(t, d, D) = tf(t, d) * idf(t, D)$$

Por lo que el estadístico *TF-IDF* incrementa proporcionalmente al número de veces que la palabra aparece en el documento, sin embargo es compensado por la frecuencia de la palabra en la colección completa de documentos o corpus. Notar que la compensación ayuda a controlar el hecho de que algunas palabras son generalmente más comunes que otras.

2.5.2 BM25

Es una función de *ranking* de documentos basada en los términos que aparecen en la *query* que llega al motor de búsqueda. *BM25* pertenece a una amplia gama de funciones de puntuación y está basada en los modelos probabilísticos de recuperación de la información (Baeza-Yates & Ribeiro-Neto, 2011).

Dada una *query* Q que contiene los términos q_1, \dots, q_n , el *ranking BM25* del documento D se calcula como:

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) * \frac{f(q_i, D) * (k + 1)}{f(q_i, D) + k * (1 - b + b * \frac{|D|}{prom(docs)})}$$

donde $f(q_i, D)$ es la frecuencia en que aparece el término q_i en el documento D ; $|D|$ es el número de palabras o términos en el documento D ; $prom(docs)$ es la media de número de palabras de los documentos en el corpus; k y b son constantes que depende de las características del corpus en el que se está haciendo la búsqueda, por lo general se asignan los valores de $k = 2$ o $k = 1.2$ y $b = 0.75$; finalmente, $IDF(q_i)$ es la frecuencia inversa de documento para el término q_i .

CAPÍTULO 3. *SCHEDULING* EN MOTORES DE BÚSQUEDA

Los sistemas de recuperación de la información como los motores de búsqueda, no solo se preocupan de la calidad de los resultados de las búsquedas (efectividad), sino que también de la velocidad con la que los resultados son obtenidos (eficiencia). Existen varias estrategias para mejorar la velocidad en la obtención de los resultados, una de ellas muy utilizada es el *caching*. Consiste en guardar en memoria de acceso rápido (memoria caché) datos temporales, que luego pueden ser sobrescrito. Una opción es hacer *caching* de los resultados de las búsquedas, de esta forma cuando una *query* es encontrada en caché el motor de búsqueda puede generar la respuesta rápidamente, reduciendo los tiempos de calculos considerablemente. Otra opción es, guardar en caché la intersección de las listas invertidas de pares comunes de términos que llegan al motor de búsqueda. Por ejemplo, si llega al sistema una consulta con los términos ('casa', 'árbol', 'perro'), se puede guardar en caché la intersección de las listas de 'casa' y 'árbol', para luego reutilizar esta información en otras *queries* que lleguen en el futuro. Para ver más técnicas de *caching* y ver el detalle de las técnicas mencionadas, ver (Büttcher et al., 2010).

Otra estrategia para acelerar el proceso de resolución de *queries* que llegan al sistema es el uso de algoritmos de planificación (*scheduling*). Un algoritmo de *scheduling* es el proceso en el cual se cambia el orden en que llegan las *queries* al motor de búsqueda con el objetivo de mejorar la eficiencia.

Existen dos clases de algoritmos de *scheduling*, estáticos y dinámicos. Los estáticos son aquellos en que se conoce el conjunto completo de tareas y las características de cada una de ellas, como por ejemplo, el tiempo de procesamiento. Los algoritmos de *scheduling* dinámicos son aquellos en que no se conoce las tareas que llegarán en el futuro, también se desconoce el momento en que éstas llegarán. La filosofía de los algoritmos de *scheduling* dinámicos es

ajustarse a los cambios que pueden haber en el sistema.

En el contexto del presente trabajo de tesis, el objetivo de hacer *scheduling* es minimizar el tiempo en que las *queries* son procesadas por un motor de búsqueda. Los motores de búsqueda como *Google*¹ o *Yahoo!*² trabajan en un contexto *online*. Esto significa que cuando las *queries* llegan al sistema (una a una), éste está obligado a tomar una decisión para planificarla sin saber cuáles *queries* llegarán en un momento posterior. A esto se le conoce como algoritmo de *scheduling online* (Albers, 2003; Borodin & El-Yaniv, 1998).

Los sistemas IR a gran escala despliegan una arquitectura distribuída (Dean, 2009), en donde el índice invertido está particionado a lo largo de servidores (*shard servers*), los cuales están encargados de procesar las *queries* que llegan al sistema. Es fácil notar que resolver una *query* con varios *shard servers* mejoraría la eficiencia. Ahora bien, para asegurar un alto rendimiento (*throughput*) del sistema, cada uno de los *shard servers* posee réplicas, de esta forma, más *queries* pueden ser procesadas en paralelo en copias idénticas del mismo *shard server*. Esto implica que el tiempo de espera de las *queries* que vienen llegando al sistema se reduce.

Como en un sistema con arquitectura como el de la Figura 3.1, una *query* puede ser procesada por varios *shard servers*, el *broker* debe escoger la réplica más apropiada para procesar la parte de la *query* asignada al *shard server*, con el objetivo de reducir el tiempo de espera de ésta. El *broker* podría seleccionar el *shard server* con el menor número de *queries* en la cola, sin embargo, este no es un parámetro adecuado, ya que el tiempo de respuesta de las *queries* puede variar considerablemente, especialmente si se usa poda dinámica (Broder et al., 2003; Moffat & Zobel, 1996).

El estudio (Broccolo et al., 2013) analiza métodos de *dropping* y *stopping* para el procesamiento de *queries* bajo altas carga de trabajo en un sistema distribuído donde existen múltiples servidores en el que cada uno resuelve una parte de la *query* para luego enviar las

¹<http://www.google.com>

²<http://www.yahoo.com>

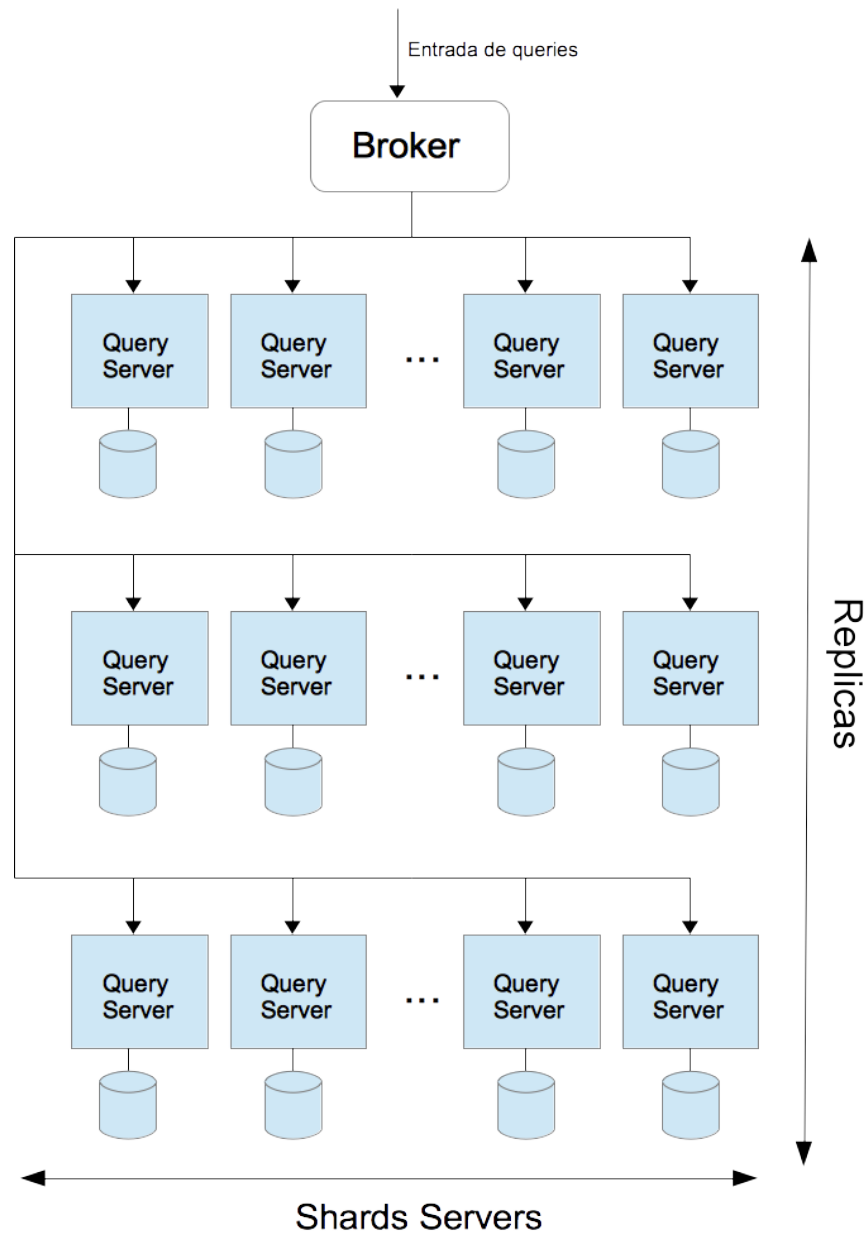


FIGURA 3.1: Arquitectura de un sistema de recuperación de la información con réplicas

consultas al *broker* y éste hace el *merge* de los resultados de acuerdo al *score* de los documentos. Se define un tiempo T , en el que la suma de el tiempo de espera de la *query* para ser procesada (t_w) y el tiempo de procesamiento de la misma (t_p) deben ser menor a T . Si es que se sobrepasa este tiempo, se tienen dos opciones (1) la *query* es desechada y se envía al *broker* una lista vacía, (2) se detiene el procesamiento de la *query* y se envía los resultados parciales hasta el momento. Finalmente se propone un método basado en la predicción de tiempo de respuesta ($\hat{pt}(q)$) de una *query* (Macdonald et al., 2012) de modo que si se cumple $\hat{pt}(q) \leq T - wt(q)$, entonces la *query* es desechada antes de comenzar a procesarse y se toma la siguiente desde la cola de espera. Notar que en estos métodos existe una pérdida de efectividad, puesto que eventualmente los servidores muchas veces no enviarán sus mejores documentos al *broker*, esto implica que el *broker* responderá al usuario un conjunto de K documento que no necesariamente son los mejores dentro del corpus completo.

En (Freire et al., 2012) se estudia el impacto que tiene la técnica de predicción de tiempos de respuestas para *queries*, (Tonellotto et al., 2011) en sistemas de recuperación de la información con réplicas. Se llega a la conclusión que usando una buena predicción, se puede reducir el tiempo que la *query* tiene que esperar para ser procesada (t_w) y también se puede reducir el tiempo total requerido para procesar el conjunto (*log*) completo de *queries* (*completion time*). En (Freire et al., 2013), se propone un modelo híbrido de *scheduling* de *queries* a través de réplicas, en el que cuando el sistema se encuentre bajo altas cargas de trabajo, se utilice política de *scheduling* basada en la predicción de tiempo de respuesta de las *queries* (Macdonald et al., 2012) y cuando el sistema se encuentre con una baja carga de trabajo, se utilice una política de *scheduling* sencilla y de menor costo como *Round Robin*.

CAPÍTULO 4. CONCLUSIONES

REFERENCIAS

Albers, S. (2003). Online algorithms: a survey. *Mathematical Programming*, 97(1-2), 3–26.

URL <http://dx.doi.org/10.1007/s10107-003-0436-0>

Baeza-Yates, R. A., & Ribeiro-Neto, B. (2011). *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Borodin, A., & El-Yaniv, R. (1998). *Online Computation and Competitive Analysis*. New York, NY, USA: Cambridge University Press.

Broccolo, D., Macdonald, C., Orlando, S., Ounis, I., Perego, R., Silvestri, F., & Tonellotto, N. (2013). Query processing in highly-loaded search engines. En O. Kurland, M. Lewenstein, & E. Porat (Editores) *String Processing and Information Retrieval*, vol. 8214 de *Lecture Notes in Computer Science*, (pág. 49–55). Springer International Publishing.

URL http://dx.doi.org/10.1007/978-3-319-02432-5_9

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., & Zien, J. (2003). Efficient query evaluation using a two-level retrieval process. En *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, (pág. 426–434). New York, NY, USA: ACM.

URL <http://doi.acm.org/10.1145/956863.956944>

Büttcher, S., Clarke, C., & Cormack, G. V. (2010). *Information Retrieval: Implementing and Evaluating Search Engines*. The MIT Press.

Dean, J. (2009). Challenges in building large-scale information retrieval systems: Invited talk. En *Proceedings of the Second ACM International Conference on Web Search and Data*

Mining, WSDM '09, (pág. 1–1). New York, NY, USA: ACM.

URL <http://doi.acm.org/10.1145/1498759.1498761>

Freire, A., Macdonald, C., Tonellotto, N., Ounis, I., & Cacheda, F. (2012). Scheduling queries across replicas. En *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, (pág. 1139–1140). New York, NY, USA: ACM.

URL <http://doi.acm.org/10.1145/2348283.2348508>

Freire, A., Macdonald, C., Tonellotto, N., Ounis, I., & Cacheda, F. (2013). Hybrid query scheduling for a replicated search engine. En *Proceedings of the 35th European Conference on Advances in Information Retrieval*, ECIR'13, (pág. 435–446). Berlin, Heidelberg: Springer-Verlag.

URL http://dx.doi.org/10.1007/978-3-642-36973-5_37

Macdonald, C., Tonellotto, N., & Ounis, I. (2012). Learning to predict response times for online query scheduling. En *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, (pág. 621–630). New York, NY, USA: ACM.

URL <http://doi.acm.org/10.1145/2348283.2348367>

Moffat, A., & Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4), 349–379.

URL <http://doi.acm.org/10.1145/237496.237497>

Tonellotto, N., Macdonald, C., & Ounis, I. (2011). Query efficiency prediction for dynamic pruning. En *Proceedings of the 9th Workshop on Large-scale and Distributed Informational Retrieval*, LSDS-IR '11, (pág. 3–8). New York, NY, USA: ACM.

URL <http://doi.acm.org/10.1145/2064730.2064734>

Turtle, H., & Flood, J. (1995). Query evaluation: Strategies and optimizations. *Inf. Process. Manage.*, 31(6), 831–850.

URL [http://dx.doi.org/10.1016/0306-4573\(95\)00020-H](http://dx.doi.org/10.1016/0306-4573(95)00020-H)

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Comput. Surv.*, 38(2).

URL <http://doi.acm.org/10.1145/1132956.1132959>