Capítulo 2

Estado del Arte

Las máquinas de búsqueda para la Web alcanzan un rendimiento eficiente mediante la combinación de varias técnicas de indexación, caching, multi-threading, heurísticas, métodos de ranking de documentos, entre otras optimizaciones. En este capítulo se revisa la bibliografía y conceptos básicos que sirven de base para este trabajo de tesis.

2.1. Clusters

Las máquinas de búsqueda para la Web están construidas sobre clusters de procesadores, los cuales ejecutan los distintos servicios que componen el sistema. Los procesadores están interconectados entre sí mediante una red de alta eficiencia que les permite enviarse mensajes entre ellos. Estos mensajes se utilizan para recolectar la información necesaria para resolver una determinada tarea, como por ejemplo la solución a una consulta de un usuario. En el cluster cada procesador tiene su propia memoria RAM y disco para almacenar información. Cada procesador puede leer y escribir información en su propia memoria y si necesita información almacenada en otro procesador debe enviarle un mensaje y esperar la respuesta. Una descripción simplificada de lo que hacen las máquinas de búsqueda para resolver las consultas de usuario es la siguiente.

En un cluster utilizado como máquina de búsqueda, cada procesador almacena una parte de la información del sistema completo. Por ejemplo, si tenemos una colección de texto que ocupa n bytes y tenemos un cluster con P procesadores, entonces se puede asignar a cada uno de los P procesadores una fracción n/P del tamaño de la colección. En

la práctica si la colección completa tiene n_d documentos o páginas Web, entonces a cada procesador del cluster se le asignan n_d/P documentos.

Las consultas de los usuarios llegan a un procesador recepcionista llamado *broker*, el cual distribuye las consultas entre los P procesadores que forman el cluster.

Dado que cada procesador del cluster tiene un total de n_d/P documentos almacenados en su memoria, lo que hacen las máquinas de búsqueda más conocidas es construir un índice invertido (detalles en la siguiente sección) en cada procesador con los documentos almacenados localmente en cada uno de ellos. Un índice invertido permite acelerar de manera significativa las operaciones requeridas para calcular las respuestas a las consultas de los usuarios.

Cada vez que el broker recibe una consulta de un usuario, éste envía una copia de la consulta a todos los procesadores del cluster. En el siguiente paso, todos los procesadores en paralelo leen desde su memoria las listas invertidas asociadas con las palabras o términos de la consulta del usuario. Luego se realiza la intersección de las listas invertidas para determinar los documentos que contienen todas las palabras de la consulta. Al finalizar este paso todos los procesadores tienen un conjunto de respuestas para la consulta. Sin embargo, la cantidad de respuestas puede ser inmensamente grande puesto que las listas invertidas pueden llegar a contener miles de identificadores de documentos que contienen todas las palabras de la consulta. Es necesario entonces hacer un ranking de los resultados para mostrar los mejores K resultados al usuario como solución a la consulta.

Para realizar el ranking de documentos es necesario colocar en uno de los procesadores del cluster los resultados obtenidos por todos los otros. Esto con el fin de comparar esos resultados unos con otros y determinar los mejores K. Sin embargo, enviar mensajes conteniendo una gran cantidad de resultados entre dos procesadores puede consumir mucho tiempo. Es deseable reducir la cantidad de comunicación entre procesadores.

Ahora, si cada procesador ha calculado los mejores resultados para la consulta considerando los documentos (listas invertidas) que tiene almacenados en su memoria local, entonces no es necesario enviarlos todos al procesador encargado de realizar el ranking final. Basta con enviar a este procesador los K mejores de cada uno de los $P-1$ procesadores restantes. Es decir, el ranking final se puede hacer encontrando los K mejores entre los $K \times P$ resultados aportados por los P procesadores.

Pero esto se puede mejorar más aun y así reducir al máximo la cantidad de comunicación entre los procesadores [59]. Dado que los documentos están uniformemente distribui-

dos en los P procesadores es razonable pensar que cada procesador tendrá más o menos una fracción K/P de los mejores K resultados mostrados al usuario. Entonces se puede trabajar en iteraciones. En la primera iteración todos los procesadores envían sus mejores K/P resultados al procesador encargado de hacer el ranking final. Este procesador hace el ranking y luego determina si necesita más resultados desde los otros procesadores. Si es así, entonces pide nuevamente otros K/P resultados y así hasta obtener los K mejores. En el peor caso podría ocurrir que para una consulta en particular uno de los procesadores posea los K mejores resultados que se le van a entregar al usuario, caso en que se necesitan P iteraciones para calcular la respuesta al usuario. Pero es muy poco probable que esto ocurra para todas las consultas que se procesan en una máquina de búsqueda grande. En la práctica se ha observado [59] que se requieren una o a lo más dos iteraciones para la inmensa mayoría de las consultas, lo cual permite reducir considerablemente el costo de comunicación entre los procesadores del cluster.

Una manera de explotar al máximo la capacidad de los procesadores del cluster es hacerlos trabajar en paralelo. Esto se puede lograr asignando los procesadores de manera circular para hacer el ranking final de las consultas. Por ejemplo, el procesador broker elige al procesador 0 para hacer el ranking de la consulta q_0, al procesador 1 para la consulta q_1, ..., el procesador $P-1$ para la consulta q_{p-1}, el procesador 0 para la consulta q_p, y así sucesivamente de manera que en un instante dado del tiempo se pueda tener a P procesadores haciendo el ranking de P o más consultas distintas en paralelo.

2.2. Índices Invertidos

Consiste en una estructura de datos formada por dos partes. La tabla del vocabulario que contiene todas las palabras (términos) relevantes encontradas en la colección de texto. Por cada palabra o término existe una lista de punteros a los documentos que contienen dichas palabras junto con información que permita realizar el ranking de las respuestas a las consultas de los usuarios tal como el número de veces que aparece la palabra en el documento. Ver figura 2.1. Dicho ranking se realiza mediante distintos métodos [9]. En este trabajo de tesis se utiliza el llamado método del vector [9, 80, 81].

Para construir un índice invertido secuencial [31, 38, 87], es necesario procesar cada documento para extraer las palabras o términos de importancia, registrando su posición y la cantidad de veces que éste se repite. Una vez que se obtiene el término, con su información correspondiente, se almacena en el índice invertido.

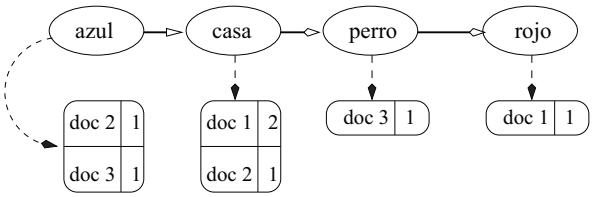


Figura 2.1: Estructura de un índice invertido.

El mayor problema que se presenta en la práctica, es la memoria RAM. Esta se termina antes de procesar toda la colección de texto. Cada vez que la memoria RAM se agota, se graba en disco un índice parcial, se libera la memoria y se comienza de cero con un nuevo conjunto de documentos. Al final de esta operación, se realiza un *merge* de los índices parciales, el cual no requiere demasiada memoria por ser un proceso en que se unen las dos listas invertidas para cada término y resulta relativamente rápido.

Respecto de la paralelización eficiente de índices invertidos existen varias estrategias. Las más utilizadas consisten en (1) dividir la lista de documentos en p procesadores y procesar consultas de acuerdo a esa distribución y en (2) distribuir cada término con su lista completa uniformemente en cada procesador.

2.2.1. Distribución por documentos

Los documentos se distribuyen uniformemente al azar en los procesadores. El proceso para crear un índice invertido aplicando esta estrategia (figura 2.2), consiste en extraer todos los términos de los documentos asociados a cada máquina y con ellos formar una lista invertida por procesador. Es decir, las listas invertidas se construyen basándose en los documentos que cada máquina posee. Cuando se resuelve una consulta, ésta se debe enviar a cada procesador (broadcast).

2.2.2. Distribución por términos

Consiste en distribuir uniformemente entre los procesadores, los términos del vocabulario junto con sus respectivas listas invertidas. Es decir, la colección completa de documentos es utilizada para construir un único vocabulario para luego distribuir las listas invertidas completas uniformemente al azar entre todos los procesadores. En esta estrategia, no es conveniente ordenar lexicográficamente las palabras de la tabla vocabulario, ya que si se

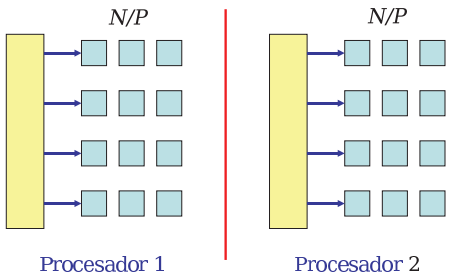


Figura 2.2: Estrategia de Distribución por Documento.

mantienen desordenadas, se obtiene un mejor balance de carga durante el procesamiento de las consultas. Ver figura 2.3. En este caso la consulta es separada en los términos que la componen, los cuales son enviados a los procesadores que los contienen.

2.2.3. Diferencias entre ambas estrategias

En el índice particionado por términos la solución de una consulta es realizada de manera secuencial por cada término de la consulta, a diferencia de la estrategia por documentos donde el resultado de la consulta es construido en paralelo por todos los procesadores. Entonces, la estrategia por documentos soporta más paralelismo (i.e., cada consulta se resuelve en paralelo), mientras que la estrategia por términos soporta mayor concurrencia (i.e., dos o más consultas pueden ser resueltas en paralelo).

Por otra parte, la estrategia de indexación por documentos permite ir ingresando nuevos documentos de manera fácil, el documento se envía a la máquina $id\_doc \ mod \ P$, y se modifica la lista local de ese procesador. Sin embargo, para el caso de ranking mediante el método del vector (descrito más abajo), se requiere que las listas se mantengan ordenadas por frecuencia, entonces no es tan sencillo modificar la lista. No obstante, para la estrategia por términos, también es necesario hacer esa actualización cuando se modifican las listas. La gran desventaja del índice particionado por términos está en la construcción del índice debido a la fase de comunicación global que es necesario realizar para distribuir las listas invertidas. Inicialmente el índice puede ser construido en paralelo como si fuese un índice particionado por documentos para luego re-distribuir los términos con sus listas invertidas.

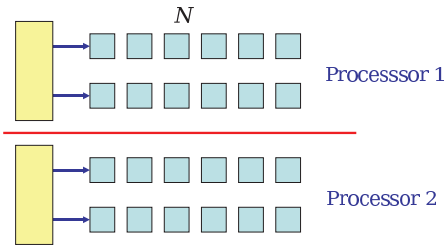


Figura 2.3: Estrategia de Distribución por Término.

2.3. Ranking de documentos

El método vectorial [9] es bastante utilizado en recuperación de información para hacer ranking de documentos que satisfacen una consulta. Las consultas y documentos tienen asignado un peso para cada uno de los términos (palabras) de la base de texto (documentos). Estos pesos se usan para calcular el grado de similitud entre cada documento almacenado en el sistema y las consultas que puedan hacer los usuarios. El grado de similitud calculado, se usa para ordenar de forma decreciente los documentos que el sistema devuelve al usuario, en forma de clasificación (ranking).

Se define un vector para representar cada documento y consulta:

- El vector d_j está formado por los pesos asociados de cada uno de los términos en el documento d_j.
- El vector q está compuesto por los pesos de cada uno de los términos en la consulta q.

Así, ambos vectores estarán formados por tantos pesos como términos se hayan encontrado en la colección, es decir, ambos vectores tendrán la misma dimensión.

El modelo vectorial evalúa el grado de similitud entre el documento d_j y la consulta q, utilizando una relación entre los vectores d_j y q. Esta relación puede ser cuantificada. Un método muy habitual es calcular el coseno del ángulo que forman ambos vectores. Cuanto más parecidos sean, más cercano a 0 será el ángulo que formen y en consecuencia, el coseno de este ángulo se aproximará más a 1. Para ángulos de mayor tamaño el coseno tomará valores que irán decreciendo hasta -1, así que cuanto más cercano de 1 esté el coseno, más similitud habrá entre ambos vectores, luego más parecido será el documento d_j a la consulta q.

La frecuencia interna de un término en un documento, mide el número de ocurrencias del término sobre el total de términos del documento y sirve para determinar cuan relevante es ese término en ese documento. La frecuencia del término en el total de documentos, mide lo habitual que es ese término en la colección, así, serán poco relevantes aquellos términos que aparezcan en la mayoría de documentos de la colección. Invirtiéndola, se consigue que su valor sea directamente proporcional a la relevancia del término.

Una de las fórmulas utilizadas para el ranking vectorial es la siguiente:

- Sea $\{t_1...t_n\}$ el conjunto de términos y $\{d_1...d_n\}$ el conjunto de documentos, un documento d_i se modela como un vector:

  $$d_i \rightarrow \vec{d_i} = (w(t_1, d_i), ..., w(t_k, d_i))$$

  donde $w(t_r, d_i)$ es el peso del término t_r en el documento d_i.

- En particular una consulta puede verse como un documento (formada por esas palabras) y por lo tanto como un vector.

- La similitud entre la consulta q y el documento d está dada por:

  $$0 <= sim(d, q) = \sum_t (w_{q,t} * w_{d,t})/Wd <= 1.$$

  Se calcula la similitud entre la consulta q y el documento d como la diferencia coseno, que geométricamente corresponde al coseno del ángulo entre los dos vectores. La similitud es un valor entre 0 y 1. Notar que los documentos iguales tienen similitud 1 y los ortogonales (si no comparten términos) tienen similitud 0. Por lo tanto esta fórmula permite calcular la relevancia del documento d para la consulta q.

- El peso de un término para un documento es:

  $$0 <= w_{d,t} = f_{d,t}/max_k * idf_t <= 1.$$

  En esta fórmula se refleja el peso del término t en el documento d (es decir qué tan importante es este término para el documento).

- $f_{d,t}/max_k$ es la frecuencia normalizada. $f_{d,t}$ es la cantidad de veces que aparece el término t en el documento d. Si un término aparece muchas veces en un documento, se supone que es importante para ese documento, por lo tanto $f_{d,t}$ crece. max_k es la frecuencia del término más repetido en el documento d o la frecuencia más alta de cualquier término del documento d. En esta fórmula se divide por max_k para normalizar el vector y evitar favorecer a los documentos más largos.

- $idf_t = log10(N/nt)$, donde N es la cantidad de documentos de la colección, nt es el número de documentos donde aparece t. Esta fórmula refleja la importancia del término t en la colección de documentos. Le da mayor peso a los términos que aparecen en una cantidad pequeña de documentos. Si un término aparece en muchos documentos, no es útil para distinguir ningún documento de otro (idf_t decrece). Lo que se intenta medir es cuanto ayuda ese término a distinguir ese documento de los demás. Esta función asigna pesos altos a términos que son encontrados en un número pequeño de documentos de la colección. Se supone que los términos raros tienen un alto valor de discriminación y la presencia de dicho término tanto en un documento como en una consulta, es un buen indicador de que el documento es relevante para la consulta.

- $Wd = (\sum(w_{d,t}^2))^{1/2}$. Es utilizado como factor de normalización. Es el peso del documento d en la colección de documentos. Este valor es precalculado y almacenado durante la construcción de los índices para reducir las operaciones realizadas durante el procesamiento de las consultas.

- $w_{q,t} = (f_{q,t}/max_k) * idf_t$, donde $f_{q,t}$ es la frecuencia del término t en la consulta q y max_k es la frecuencia del término más repetido en la consulta q, o dicho de otra forma, es la frecuencia mas alta de cualquier término de q. Proporciona el peso del término t para la consulta q.

2.4. Trabajo Relacionado

2.4.1. Índices invertidos

Existe abundante literatura que describe como resolver de manera eficiente consultas sobre índices distribuidos sobre un conjunto de P procesadores [4, 15, 17, 32, 33, 37, 50, 85, 86]. En esos trabajos se discute la eficiencia de los métodos de indexación basados en partición por documentos y partición por términos. Para cada una de estas estrategias, se han propuesto diferentes técnicas para resolver consultas sobre el índice distribuido en la memoria de los procesadores y también se han propuesto técnicas híbridas para mejorar el rendimiento y/o balance de carga (ver por ejemplo [6, 19, 28, 61, 65, 66, 73, 74, 79, 95, 97, 103]).

2.4.2. Compresión de índices invertidos

En la compresión de índices invertidos [72, 75, 76, 101, 104], y en particular de la lista de posteos o invertida, la literatura actual habla de Variable-Byte [90], S9 [3], S16 [89] y PForDelta [110] como métodos modernos que ofrecen la mejor velocidad de descompresión sin dejar de lado la tasa de compresión [14, 89, 105]. Es por esto que han sido seleccionados para presentar una breve descripción de cada uno de ellos a modo de estado del arte. Para ello se considerará una lista de posteos ordenada por IDsdoc, donde la lista comienza con el IDdoc del primer documento y de allí en adelante cada número representa la diferencia con el IDdoc del documento anterior.

Variable-Byte : Expresa un número en la menor cantidad de bytes posibles. Para ello ocupa el bit más significativo de cada byte como un bit de marca, que le indica si el byte actual es el último de la secuencia, el último byte de la secuencia es marcado con un 0 en el bit más significativo. Los restantes 7 bits se ocupan para representar el número a codificar en formato binario. Así por ejemplo el número 513, que en binario es 1000000001, codificado con Variable-Byte queda 10000100 00000001.

Es un método muy sencillo y también eficiente a la hora de descomprimir. El inconveniente que éste presenta para las listas de posteo, es que cuando son muy largas y las diferencias se hacen pequeñas, Variable-Byte representa en un byte números que sólo requieren un par de bits, desperdiciando gran cantidad de espacio.

S9 : Este método trata de guardar la mayor cantidad posible de números en una palabra de máquina (32 bits). Para hacer esto, los cuatro primeros bits son ocupados como bits de estado, donde se especifica como son ocupados los restantes 28 bits. S9 obtiene su nombre de los 9 posibles casos que se dan al dividir 28 bits en cantidades iguales; 1 número de 28 bits (y su recíproco 28 números de 1 bit), 2 números de 14 bits (y su recíproco), 3 números de 9 bits (y su recíproco, desperdiciando 1 bit), 4 números de 7 bits (y su recíproco) y 5 números de 5 bits (desperdiciando 3 bits).

Para diferencias muy pequeñas, con este método se logra una gran cantidad de números empaquetados en una palabra de máquina y además cuenta con una interesante velocidad de descompresión. Sin embargo, si se intercala con los números pequeños, se tienen diferencias muy grandes y es fácil ver que el método no se comporta bien en cuanto al ahorro de espacio.

S16 : Variante de S9, que toma algunos casos especiales que no dividen los 28 bits en forma regular, logrando con esto completar 16 casos (de ahí su nombre). De esta manera se

pretende ocupar la mayor cantidad de bits posibles en forma eficiente, eliminando algunos casos que desperdiciaban bits, como por ejemplo 5 números de 5 bits, se reemplaza por 2 casos; el primero es 3 números de 6 bits y 2 números de 5 bits, el segundo es 2 números de 5 bits y 3 números de 6 bits.

En compresión este método logra resultados levemente mejores a S9, manteniendo su velocidad de descompresión, sin embargo aún mantiene las debilidades del método original.

PForDelta : Este método deriva originalmente del método de compresión FOR [35] (Frame Of Reference). Para un cierto rango de números, FOR toma ambos extremos, observa cuantos casos posibles existen entre el menor y el mayor de los extremos, codificando cada caso en binario desde 0 para el menor hasta el número binario que sea necesario para cubrir todo el rango, ocupando así la menor cantidad posible de bits para representar cada caso. Por ejemplo, se tiene un rango en la lista de posteos donde se repiten muchas diferencias entre 2 y 5, se sabe que los posibles casos son {2,3,4,5}, para diferenciar entre ellos nos basta con usar 2 bits, así cada caso quedaría; 00 para 2, 01 para 3, 10 para 4 y 11 para 5.

El método FOR presenta problemas al tener rangos con diferencias demasiado grandes, PForDelta ataca ese problema. Para esto, PFordelta separa la mayoría (aproximadamente el 90 % según el autor [110]) de los números que pueden ser representados en b bits (coded) y los que no (exceptions). De esta manera los números codificables son guardados con FOR y las excepciones se guardan descomprimidas.

2.5. Caches

Inicialmente los motores de búsqueda para la Web mantenían los resultados de las consultas frecuentes a lo largo del tiempo en un cache estático. Posteriormente, en [69] se muestra que el rendimiento del cache estático es deficiente, ya que muchas consultas que llegan a la máquina de búsqueda son frecuentes por periodos cortos de tiempo y por lo tanto, el cache estático no las considera. Dado esto, técnicas de caching dinámico basadas en políticas de reemplazo como LRU presentan mejor rendimiento. En [53] se calcula un puntaje para las consultas y se estima la probabilidad de que una consulta se repita en el futuro cercano, esta técnica es llamada *Probability Driven Caching* (PDC).

En [27] se propone una estrategia de caching donde se mantiene en el cache de resultados una sección estática y otra dinámica, obteniendo buenos resultados, *Static-Dynamic*

Caching (SDC). En SDC, la parte estática mantiene las consultas más frecuentes en largos periodos del tiempo, y la parte dinámica es implementada con políticas de reemplazo como LRU. Esta estrategia, logra una tasa de aciertos superior al 30 % en experimentos basados en el log de Altavista. En [54] se muestra que al almacenar intersecciones de pares de términos frecuentes, determinados por la co-ocurrencia en el log de consultas, es posible mejorar el rendimiento significativamente. En [8] se muestra que al tener un cache de listas invertidas, se logran mejores resultados que al tener un cache de resultados únicamente. En [30] se estudia el cache de resultados con ponderación (*Weighted Result Caching*), donde el costo de procesar una consulta se considera en el momento de decidir su admisión en el cache de resultados.

Las técnicas anteriores entregan una respuesta exacta a las consultas a través de la información que existe en el cache. El uso de técnicas de aproximación mediante análisis semántico puede ayudar a sacar mejor utilidad del contenido en el cache de resultados. En [34] se utiliza esta técnica donde se proporciona una pauta para identificar los distintos casos surgen al considerar distintas relaciones entre los términos de la nueva consulta y los términos de las consultas almacenadas en el cache. Esto permite obtener resultados aproximados para una consulta que, aún cuando no está exactamente en el cache, comparte términos con otras consultas. Esta técnica se puede utilizar para permitir a la máquina de búsqueda seguir respondiendo consultas incluso en casos en que sus procesadores están saturados debido a subidas bruscas en el tráfico de consultas.

En [20, 21] se presentan nuevos métodos semánticos para resolver consultas. Se propone mantener clusters de respuestas co-ocurrentes identificadas en una región, cada uno de ellos asociados a una firma. Las nuevas consultas frecuentes también son identificadas con una firma, donde las regiones con firmas similares son capaces de entregar una respuesta. Los resultados experimentales muestran que el rendimiento de los diferentes métodos semánticos dependen del tipo de solapamiento. En [2] se estudia el problema de escalabilidad para métodos de caching semántico.

En [83] se propone un método que utiliza un log de consultas para formar P clusters de documentos y Q clusters de consultas. Los clusters son formados mediante el uso de un algoritmo de co-clustering. Esto permite la creación de una matriz donde cada elemento indica que tan pertinente es un cluster de consultas a un cluster de documentos. Además, para cada cluster de consultas, se mantiene un archivo de texto que contiene todos los términos presentes en las consultas del cluster. Cuando la máquina broker recibe una consulta q, ésta calcula que tan pertinente es q a uno de los clusters de consultas mediante el uso de métodos de similitud. Estos valores de similitud se utilizan en combinación

con los elementos de la matriz de co-clustering, para calcular un ranking de clusters de documentos. Los primeros lugares del ranking representan los procesadores que son capaces de responder a una consulta con mayor precisión. Es decir, son los procesadores que tienen mayor probabilidad de aportar documentos, y formar parte de los resultados top-k globales de una consulta. La idea es enviar la consulta a los procesadores más probables de aportar documentos para la respuesta.

Los métodos presentados en [28, 71, 77] persiguen un objetivo similar pero utilizando técnicas diferentes. Estos métodos están basados en el uso de un cache de aplicación llamado *Location Cache* (LCache) el cual registra los procesadores que aportan los top-K documentos para consultas almacenadas en cache. Los métodos propuestos utilizan técnicas semánticas y de aprendizaje sobre la LCache. Siempre el objetivo final es enviar la consulta a un conjunto reducido de procesadores con el fin de evitar saturar al motor de búsqueda frente a situaciones de alzas en el tráfico de consultas.

El trabajo presentado en [28] propone usar el LCache como un cache semántico. El método semántico para la evaluación de las nuevas consultas, utiliza la frecuencia inversa de los términos de las consultas almacenadas en cache, que determinan cuando los resultados recuperados desde el cache son buenas aproximaciones al resultado exacto. La precisión de los resultados varía en función de la coincidencia semántica de los casos analizados. En [60] se propone una combinación de LCache y RCache (cache de resultados estándar) usando estrategias dinámicas de caching, obteniendo un desempeño eficiente en escenarios de tráfico variable de consultas, opuesta a la estrategia que sólo utiliza RCache la cual se satura ya que no es capaz de trabajar de manera eficiente con variaciones de tráfico. El trabajo en [71] aplica métodos de máquinas de aprendizaje sobre el LCache para determinar cual es el mejor nodo de búsqueda.

Una idea similar, pero basada en el modelo vectorial se aplica en [77] que predice los nodos de búsqueda para las consultas que no se encuentran en la LCache. En todos los casos, la motivación es reducir la carga sobre los nodos de búsqueda frente a cambios bruscos en el tráfico de consultas. La idea es entregar una buena aproximación de respuestas a las consultas utilizando pocos nodos de búsqueda por cada una y evitar de esta manera la saturación y el comportamiento inestable del motor de búsqueda.

2.6. Arquitectura de una máquina de búsqueda

Los centros de datos para motores de búsqueda en la Web, contienen miles de procesadores intercomunicados, llamados clusters. Por lo general, cada cluster se especializa en una sola operación relacionada con el procesamiento de una consulta. La operación que más tiempo demanda en el proceso de resolver una consulta, es determinar los mejores K (top-K [42, 52]) documentos (IDs) que mejor coinciden con una consulta dada. Hay otras operaciones atendidas por otros clusters, tales como la construcción de la página Web de resultados de los mejores K documentos y la publicidad relacionada con los términos de la consulta. Determinar los top-K documentos, es la parte más costosa [25] del proceso dado el gran tamaño de la colección de texto de la Web.

Para el conjunto top-K, una arquitectura estándar de cluster es un arreglo (array) de $P \times D$ procesadores, donde P indica el nivel de partición de la colección de texto y D el nivel de replicación de la colección.

El tiempo de respuesta de una consulta es proporcional al tamaño de la colección de texto y para obtener tiempos de respuesta por debajo de un cierto límite, la colección de texto es dividida en P particiones diferentes. Cada consulta es enviada a las P particiones y, en paralelo, se determinan los top-K locales a cada partición. A partir de los K locales, se generan los K globales de la consulta. Existe una máquina aparte llamada *broker* que se encarga de enviar las consultas a los nodos de búsqueda, para luego recopilar los resultados.

El nivel de replicación indica que cada una de las P particiones son replicadas D veces. El propósito de replicar es por dos motivos. Entregar un soporte de tolerancia a fallas y alanzar una tasa de solución de consultas por unidad de tiempo (throughput) eficiente. En cualquier instante de tiempo, diferentes consultas, pueden ser resueltas por diferentes réplicas. En cada partición una de las réplicas es seleccionada uniformemente al azar.

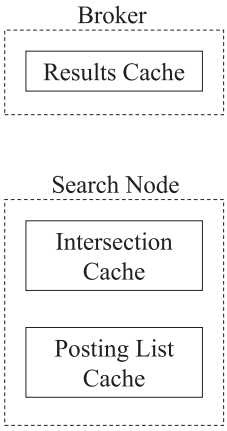
La implementación de una máquina de búsqueda en un arreglo logra un desempeño eficiente utilizando técnicas de indexación [26, 56, 98, 109] y caching [5, 7, 11, 16, 64, 68, 78, 93, 106, 107]. Con respecto a la indexación, cada nodo de búsqueda contiene una lista invertida que es usada para mapear de manera eficiente los términos de la consulta con los documentos relevantes [61, 74]. Las listas invertidas asociadas a los términos de la consulta, rápidamente entregan un conjunto de documentos relevantes que son ordenados mediante técnicas de ranking. Las listas invertidas pueden ser muy grandes y por lo general sólo se hace referencia a un subconjunto de ellas. Por este motivo, muchas de las listas invertidas son almacenadas en memoria secundaria o en formato comprimido en memoria principal,

y sólo un subconjunto de ellas se mantienen descomprimidas en memoria principal, a esto se llama cache de listas invertidas (*posting list cache*). Este cache es administrada bajo el estándar LRU.

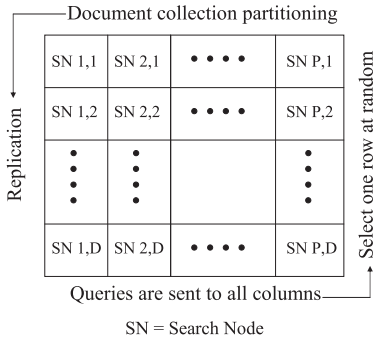
Un segundo cache, llamado cache de resultados (*results cache*), se mantiene en la máquina broker. Este se encarga de recibir las consultas, enviarlas al arreglo de cada procesador para que sean resueltas, y recolectar los resultados. El cache de resultados, almacena las respuestas a las consultas más frecuentes. Las políticas de administración para el cache de resultados, se basa en la estrategia SDC [27]. Se mantiene una sección de sólo lecturas para almacenar las respuestas de las consultas más frecuentes por un periodo largo de tiempo. Una primera parte se administra como un cache estática, con la política LFU. La segunda parte, representa el cache dinámico, administrado con LRU, se encarga de las consultas más populares por un corto periodo de tiempo. Experimentos presentados en [27] sugieren destinar un 80 % para la parte estática y un 20 % para la parte dinámica.

Recientemente [30] mostró que la parte dinámica es mejor administrarla utilizando una política llamada *Landlord*, la cual es una generalización de LRU. Esta política de administración, considera el costo de procesar una consulta cuando se decide si se encuentra o no en cache. Inicialmente la prioridad de entrada a el cache está dada por el costo promedio L de procesar una consulta en cada nodo de búsqueda. Cuando la entrada es re-usada por la misma consulta (cache hit) y la prioridad se incrementa por L unidades. Cuando una entrada es sustituida por una nueva consulta, todas las entradas que quedan en la memoria cache disminuyen su prioridad en L unidades. Una implementación eficiente de esta estrategia es utilizando colas de prioridad. Cada nueva entrada recibe una prioridad $s + L$ con un valor inicial $s = 0$. La siguiente entrada que va ser sustituida es aquella que tiene el menor valor numérico v en la cola de prioridad y se hace $s = v$ para la prioridad $s+L$ del nuevo elemento que se almacena en la entrada. Para la cache estática se considera $f \cdot L$ para los valores de prioridad donde f es la frecuencia de ocurrencia de las consultas en un log.

Alternativamente, una tercera cache es mantenida en este esquema. Esta es la cache de intersecciones (*intersection cache*) [54] la cual mantiene en cada nodo de búsqueda la intersección de las listas invertidas que están asociadas a los términos más frecuentes de las consultas. La operación de intersección es útil para detectar los documentos que contienen todos los términos de la consulta, requerimiento típico para encontrar los top-K resultados en los principales motores de búsqueda.



(a) Jerarquía estándar de Cache



(b) Camino de una consulta en un *search node*

Figura 2.4: Enfoque base del procesamiento de una consulta.

De esta manera, las tres caches definen una jerarquía que va desde los datos más específicos pre-computados para las consultas (cache de resultados) a los datos más genéricos para resolver una consulta (cache de listas invertidas).

La Figura 2.4.a ilustra el estado del arte para una jerarquía de caches para máquinas de búsqueda, donde un cache de intersecciones y listas invertidas son mantenidas en cada uno de los $P \times D$ nodos de búsqueda. La Figura 2.4.b muestra qué ocurre cuando una consulta no encuentra los resultados en el cache de resultados de la máquina broker. En este caso, la consulta es enviada a un nodo de búsqueda por cada columna. En cada columna, el respectivo nodo de búsqueda es seleccionado de manera uniformemente al azar.

2.7. Procesamiento Round-Robin de Consultas

Es un método diseñado para optimizar el uso de los recursos del cluster de procesadores frente a tráfico alto de consultas. El proceso de resolver una consulta utilizando P procesadores se ve reflejado en la Figura 2.5. (202) las consultas llegan a la máquina broker (204). Cada procesador 206(1) a 206(P) tiene 2 roles: *fetcher* y *ranker*. En general, el proceso llamado *fetcher* puede simplemente reunir datos (por ejemplo, desde memoria secundaria) y también puede ejecutar algunos pre-procesos. El proceso llamado *ranker* recibe los resultados parciales desde el fecther y calcula la respuesta final para la consulta (204).

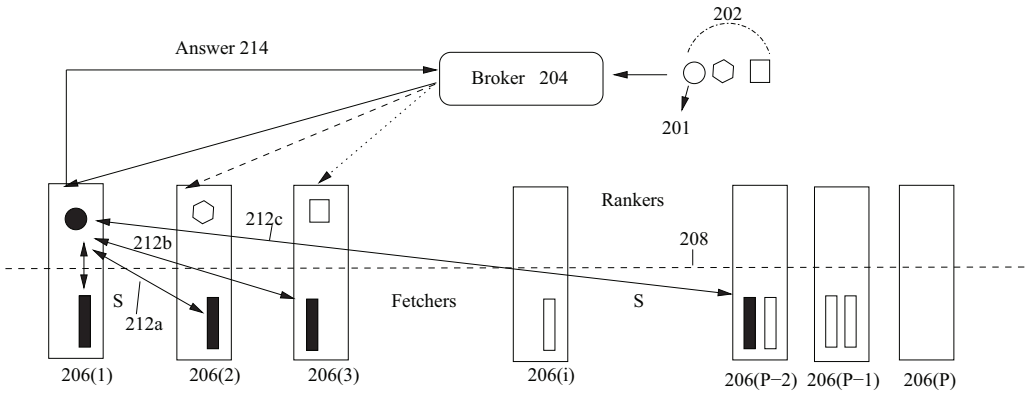


Figura 2.5: Arquitectura para el proceso de una consulta.

En la Figura 2.5, el doble rol se ve reflejado en la linea 208. En este ejemplo, para el proceso fetching, cada procesador (206) reúne los datos desde el índice para una consulta en particular. Para el caso del ranking, cada procesador (206) realiza una unión (merge) de los datos asociados a la consulta. El ranker envía la consulta (201) a T fetchers (212), donde cada T trabaja de acuerdo al tipo de partición del índice. En relación al tráfico, el ranker puede trabajar con varias consultas en paralelo en un período de tiempo.

El broker es el encargado de enviar las consultas a los procesadores del cluster y recibir las respectivas respuestas (cada procesador es un nodo del chip-multiprocesador del cluster). La distribución de las consultas a través de los procesadores, es mediante heurísticas de balance de carga. En particular, la heurística depende de la partición del índice invertido con que se esté trabajando.

La máquina de búsqueda puede cambiar dinámicamente su modo de operación de acuerdo al tráfico de consultas observado.

- **Modo Asincrónico (Async).** Un tráfico bajo de consultas gatilla el modo Async. Cada consulta es atendida por un único thread maestro encargado de resolver la consulta. El thread maestro se puede comunicar con otros P threads esclavos; cada uno se ubica en un procesador del cluster.
- **Modo Sincrónico (Sync).** Un tráfico alto de consultas gatilla el modo Sync. Todos los threads activos son bloqueados y solo un thread tiene el control del proceso de consultas. En este caso los mensajes son almacenados en el buffer de todos los procesadores del cluster y enviados al comienzo de cada iteración, punto en el cual los procesadores son sincronizados. En este modo operacional, los recursos del sistema

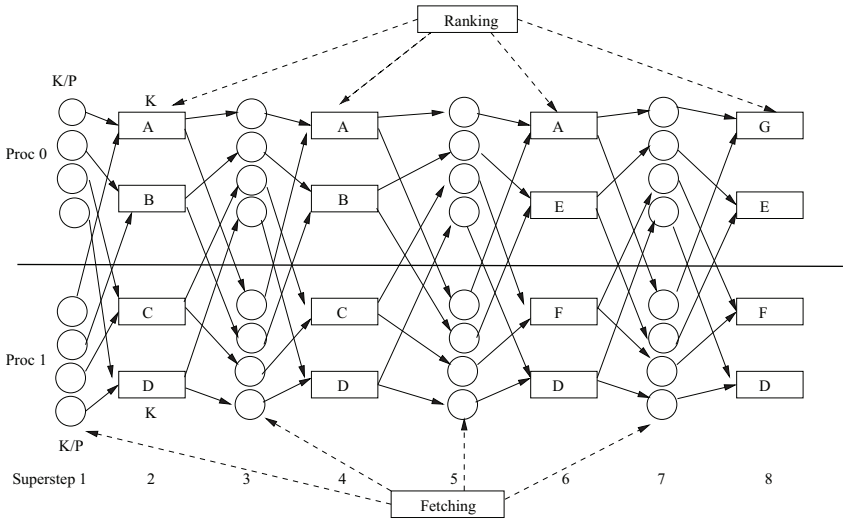


Figura 2.6: Procesamiento round-robin sincrónico para siete consultas.

son utilizados de mejor manera dada la forma de distribuir los threads. Se reducen los costos de sincronización y comunicación, que son ejecutados en bloques. Se le llama *superstep* al periodo que transcurre entre dos sincronizaciones en forma de barrera de los procesos que participan en la solución de una o más consultas.

El procesamiento de una consulta aplicando el modelo round-robin, se realiza en cualquier de los modos de funcionamiento de una máquina de búsqueda. La Figura 2.6 ilustra, para $P = 2$ procesadores y en cada procesador $T = 2$ procesos o threads, el proceso iterativo de 7 consultas etiquetas como A, B, C, D, E, F y G. El proceso es distribuido a través de 2 procesadores y 8 supersteps donde existe uso exclusivo de los supersteps para las operaciones de fetching y ranking. En la figura, las consultas son procesadas secuencialmente en cada procesador. Por ejemplo, el thread del procesador 0 procesa por completo la consulta A y, a continuación, se otorga el quantum a la consulta B.

El mismo esquema de la Figura 2.6 puede ser usado para operar en el modo asincrónico. La diferencia es que los periodos de ranking y fetching se van a solapar entre los procesos o threads, pero la tendencia global de sincronización de operaciones se va a mantener.