Anatomía de un motor de búsqueda a gran escala de web hipertextual.

Sergey Brin y Lawrence Page
{sergey, page}@cs.stanford.edu
Departamento de Informática, Universidad de Stanford, Stanford, California 94305

Traducción al castellano por José M. Dueñas Quesada <www.joseduenas.com>

Original disponible en: http://infolab.stanford.edu/~backrub/google.html

Versión preliminar

Extracto

En este artículo, presentamos Google, un prototipo de un motor de búsqueda a gran escala que hace un uso intensivo de la estructura presente en hipertexto. Google está diseñado para rastrear e indexar la Web de una forma eficiente y producir muchos mas resultados de búsqueda satisfactorios que los actuales sistemas de búsqueda. El prototipo con una base de datos de textos completos y de hiper enlaces de al menos 24 millones de páginas se encuentra disponible en http://google.stanford.edu/

El proceso de ingeniería de un motor de búsqueda es una desafiante tarea. Los motores de búsqueda indexan centenares de millones de páginas web, implicando otro numero equiparable de términos distintos. Ejecutan decenas de millones de consultas cada día. A pesar de la gran importancia de los motores de búsqueda a gran escala, se han llevado a cabo muy pocas investigaciones en el ámbito académico. Además, debido al rápido avance de la tecnología y la proliferación de webs, crear un motor de búsqueda hoy, difiere mucho de lo que sería crearlo hace tres años. Este artículo proporciona una profunda descripción de nuestro motor de búsqueda a gran escala – el primero en el que se hace pública su descripción de forma publica, que nosotros sepamos, al menos hasta la fecha.

Dejando aparte los problemas de la técnicas de búsqueda tradicionales de búsqueda aplicados a éstas magnitudes de datos, hay nuevos desafíos técnicos que involucran usar la información presente en el hipertexto para producir mejores resultados en las búsquedas. Este articulo aborda el tema de cómo construir un sistema a gran escala que sea práctico y que además pueda beneficiarse de la información presente en el hipertexto. También tratamos el problema de cómo tratar eficientemente con hipertexto sin control el cual ha podido ser publicado por cualquiera que haya querido.

Palabras clave: World Wide Web, Motores de búsqueda, Recuperación de la información, PageRank, Google

1. Introducción

(Nota: Hay dos versiones de éste artículo – una versión completa y otra mas corta en formato impreso. La versión completa se encuentra disponible en la web y en el CD-ROM de la conferencia.

La web crea nuevos retos para la recuperación de la información. La cantidad de información en la web está creciendo rápidamente, al igual que el número de nuevos usuarios inexpertos en el arte de la investigación de la web. La gente normalmente navega por la Web usando los enlaces, a menudo comenzando por directorios de web mantenidos manualmente, como es Yahoo! o con motores de búsqueda. Los directorios de webs cubren, en efecto, temas populares, pero son subjetivos, caros de construir y de mantener, difíciles de mejorar, y no pueden cubrir todos los temas. Los motores de búsqueda automatizada se basan en encontrar palabras clave que normalmente devuelven demasiados resultados de poca calidad. Para empeorar aun mas las cosas, algunos anunciantes en internet hacen uso de técnicas de mala praxis, lo cual minimiza la eficacia de éstos buscadores. Nosotros hemos construido un motor de búsqueda a gran escala que aborda muchos de los problemas de los sistemas actuales. Hace especial hincapié en el uso de la estructura adicional presente en el hipertexto para proporcionar unos resultados de mejor calidad. Escogimos el nombre de Google, porque es como se pronuncia googol, o 10^{100} lo cual se ajusta muy bien a nuestra meta de construir motores de búsqueda a gran escala.

1.1 Motores de búsqueda web - perfeccionándose: 1994 - 2000

La tecnología de motor de búsqueda ha tenido que mejorar rápidamente debido al crecimiento de la web. En 1994, uno de los primeros motores de búsqueda, el World Wide Web Worm (WWWW) [McBryan 94] tuvo un índice de 110,000 páginas web y documentos web accesibles. Así, en Noviembre de 1997, se indexaba de 2 millones de webs (WebCrawler) a 100 millones (de Search Engine Watch). Es previsible que para el año 2000, un extenso índice de la Web contenga sobre mil millones de documentos. Al mismo tiempo, el número de consultas en motores de búsqueda habrá crecido muchísimo también. En Marzo y Abril de 1994, el World Wide Web Worm recibió de media unas 1500 consultas por día. En Noviembre de 1997, Altavista aseguró haber manejado 20 millones de consultas por día, aproximadamente. Teniendo en cuenta el creciente número de usuarios en la web, y los sistemas automatizados de consulta en buscadores, se prevee que se pueda alcanzar los cientos de millones de consultas por día para el año 2000. La meta de nuestro sistema es abordar mucho de éstos problemas, tanto de escalabilidad como de calidad, provocados por enorme crecimiento del número de consultas en los motores de búsqueda.

1.2. Google: creciendo con la Web

Crear un motor de búsqueda que pueda sopòrtar el crecimiento tan grande del numero de webs de hoy en día, presenta muchos desafíos. Se requiere de una tecnología de rastreo rápida para recoger los documentos web y mantenerlos actualizados. El espacio de almacenamiento debe ser usado

eficientemente para almacenar índices y opcionalmente, los propios documentos web. El sistema de indexado debe procesar cientos de gigabytes de datos de una forma eficiente. Las consultas deben ser gestionadas rápidamente, a una tasa de cientos de miles por segundo.

Estas tareas van a ir dificultándose atendiendo al crecimiento de la Web. De todos modos, el coste y eficiencia del hardware han mejorado notablemente hasta el punto de dejar casi fuera de juego ésta dificultad. Hay , no obstante, varias excepciones notables a éste progreso, como lo son el tiempo de acceso a disco y la robustez de los sistemas operativos. En el diseño de Google, hemos tenido en cuenta, tanto el crecimiento de la Web, como los cambios en la tecnología. Google está diseño para tener una gran escalabilidad para grandes conjuntos de datos. Hace un uso eficiente del espacio de almacenamiento en disco para guardar el índice. Sus estructuras de datos están optimizadas para la velocidad y el acceso eficiente (ver sección 4.2). Además, esperamos que el coste de indexar y almacenar texto o HTML, irá disminuyendo respecto a la cantidad que esté disponible (ver Appendix B). Esto dará lugar a unas propiedades de escalabilidad muy favorables para sistemas centralizados como Google.

1.3 Diseñando metas.

1.3.1 Calidad de búsqueda mejorada.

Nuestra principal meta es mejorar la calidad de los motores de búsqueda. En 1994, algunas personas pensaban que un índice completo de búsqueda haría posible encontrar cualquier cosa de forma facil. De acuerdo al Best of the Web 1994 -- Navigators, "El mejor servicio de navegación tendría que hacer posible encontrar cualquier cosa en la Web (una vez que todos los datos estuvieran introducidos en el índice)." De todos modos, la Web de 1997 es bastante diferente. Cualquiera que haya usado un motor de búsqueda recientemente, puede asegurar categóricamente, que el hecho de que un índice sea mas completo o no, no es el único factor en la calidad de los resultados obtenidos en las búsquedas. Los “resultados basura” a menudo hacen fracasar a los resultados en lo que realmente está interesado el usuario. De hecho, en Noviembre de 1997, sólo uno de los cuatro mas importantes motores de búsqueda comerciales se encontraba a sí mismo (devolvía su propia página principal dentro de los 10 primeros resultados como respuesta a una búsqueda realizada con su propio nombre). Una de las principales causas de éste problema es que el numero de documentos a indexar ha ido incrementañdose en muchos órdenes de magnitud, pero la habilidad del usuario para mirar en los documentos, no. La gente suele mirar solo los diez primeros resultados de las búsquedas. A causa de esto, ya que la colección crece, necesitamos herramientas que tengan mucha precisión (numero de documentos devueltos aparezcan en el top ten). De hecho, queremos que la noción de “relevante” solo incluya a aquellos resultados que realmente sean los mejores ya que habría decenas de miles de documentos ligeramente relevantes. Esta gran precisión es importante incluso a costa de memoria (el total de documentos que el sistema es capaz de devolver). Existe bastante optimismo en el hecho de que elñ uso de mas información hipertextual puede ayudar a mejorar la búsqueda y otras aplicaciones [Marchiori 97] [Spertus 97] [Weiss 96] [Kleinberg 98]. En particular, la estructura de enlaces [Page 98] y el enlace de texto proveen de mucha información que facilitan el proceso de asignar relevancia y de un filtrado de calidad. Google hace uso de de las dos, tanto de la estructura de enlace como del texto ancla. (ver Secciones 2.1 y 2.2).

1.3.2 Investigación universitaria en motores de búsqueda.

Aparte de éste crecimiento tan tremendo, la Web también se ha ido haciendo mas comercial a lo largo del tiempo. En 1993, el 1.5% de los servidores web estaban en dominios .com. Este numero creció mas del 60% en 1997. Al mismo tiempo, los motores de búsqueda, han ido migrando del dominio académico al comercial. Hasta el punto actual, en el que la mayoría del desarrollo en motores de búsqueda se ha ido a compañías que apenas publican detalles técnicos. Ésto causa que la tecnología de motores de búsqueda se conviertan en un arte oscuro y se oriente a la publicidad (ver Apendice A). Con Google, tenemos una meta importante, que no es otra que facilitar el desarrollo y comprensión en la realidad académica.

Otra meta importante en el diseño fue construir sistemas que pudieran usar un numero razonable de personas. El uso era importante para nosotros porque pensamos que una de las investigaciones mas interesantes involucraría el influenciar el uso de la ingente cantidad de datos que hay disponibles en los modernos sistemas web. Por ejemplo, hay decenas de millones de búsquedas realizadas cada día. Aun así, es muy difícil conseguir estos datos, principalmente porque es considerado comercialmente valioso.

Nuestra meta final en el desarrollo era construir una arquitectura que pudiera soportar nuevas actividades en la investigación de datos web a gran escala. Google almacena todos los actuales documentos que recoge de forma comprimida. Una de nuestras principales metas en el desarrollo de Google era establecer un entorno, a donde todos los investigadores pudieran entrar rápidamente, procesar grandes cantidades de trozos de la Web, y produjeran resultados interesantes que no podrían haber sido producidos de otra manera. En el corto plazo en el que el sistema ha sido puesto en marcha, ya hay varios artículos que hemos realizado, usando las bases de datos de Google, y mucho otros están en camino. Otra meta que tenemos es crear un entorno parecido a un laboratorio donde los investigadores o incluso estudiantes puedan proponer y realizar experimentos interesantes sobre nuestra gran cantidad de datos generados por Google.

2. Características del sistema

El motor de busqueda Google tiene dos características importantes que ayudan a la producción de resultados de gran precisión. El primero hace uso de la estructura de enlaces, para calcular una puntuación en un ranking de calidad para cada web. Este ranking se llama PageRank y se describe en detalle en [Page 98]. En el segundo, Google utiliza los enlaces para mejorar los resultados de la búsqueda.

2.1 PageRank: Poniendo orden en la web

Los enlaces en la Web, son un recurso muy importante que no ha venido siendo utilizado en los demás motores de búsqueda. Nosotros hemos creado mapas que contienen tantos como 518 millones de estos hiper enlaces, que son una muestra significante del total. Estos mapas permiten un calculo rápido del PageRank, una medida objetiva de la importancia de la web, que corresponde muy bien con la idea subjetiva de importancia que tienen las personas acerca de las webs. A causa de esta correspondencia, PageRank es un método excelente para priorizar los resultados de las búsquedas basadas en palabras clave. Para los temas mas populares, una simple coincidencia en la búsqueda de

texto que se restringe a los títulos de la web, actúa de forma efectiva cuando PageRank prioriza los resultados (demo disponible en google.stanford.edu). Para las búsquedas basadas en texto completo, en el sistema Google, PageRank también ayuda de forma fantástica.

2.1.1 Descripción del calculo de PageRank.

Hemos aplicado la forma de hacer referencias y citas de los libros a la web. El numero de citas o referencias a un página en particular miden la importancia o calidad de esa página. PageRank extiende ésta idea, no contando todos los enlaces a una pagina como igual de valiosos, sino tenido en cuenta a su vez cuantos enlaces tiene ésa pagina desde la que se referencia. PageRank se define como sigue::

> *Consideramos que una página A es refenciada por las páginas T1...Tn . El parámetro d es un factor de amortiguamiento el cual puede ser establecido entre 0 a 1 . Por lo general lo establecemos a 0.85. Hay mas detalles sobre d en la siguiente sección. También C(A) se define como el numero de enlaces que salen de A. Por lo tanto el PageRank es:*
>
> *PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))*
>
> *El PageRank es una distribución de probabilidad sobre paginas web, por lo tanto la suma de todos los PageRanks de las página web será un.*

El PageRank o *PR(A)* puede ser calculado usando un simple algoritmo iterativo, que corresponde con el vector principal de auto valores de la matriz normalizada de los enlaces. También, un PageRank para 26 millones de webs puede ser computado en unas cuantas horas en un ordenador de potencia media. Existen muchos otros detalles que se escapan a la finalidad de éste artículo.

2.1.2 Una Justificación Intuitiva

PageRank puede ser concebido como un modelo de comportamiento del usuario. Asumimos que hay un "navegante aleatorio" al que se le da una web aleatoria y empieza a navegar por los enlaces, no presionando nunca el botón de atrás, hasta que se aburre y empieza con otra página aleatoria. La probabilidad de que el "navegante aleatorio" visite una página en cuestión, es el PageRank de ésa página. Y el factor de amortiguamiento d es la probabilidad que existe en cada página que hace que el navegante se aburra y pida otra página nueva. Una variación importante es añadir el factor de amortiguamiento a una sola pagina o a un grupo de paginas. esto permite la personalización y puede hacer casi imposible engañar al sistema deliberadamente para conseguir un PageRank mayor.

Otra idea intuitiva es que una página tendrá mayor PageRanks si hay muchas páginas que apunten a ella o si hay menos páginas pero que tengan un PageRank alto. Intuitivamente, las páginas mas referenciadas desde otras paginas merecen la pena verlas. O también, páginas que sean solamente referenciadas desde la página principal de Yahoo, también merecerían tenerlas en cuenta. Si una pagina no tuviera buena calidad, o tuviera enlaces rotos, seria bastante probable que Yahoo no la enlazara. PageRank maneja todos estos casos, propagando recursivamente los pesos de importancia de las web recursivamente.

2.2 Texto ancla

El texto de los enlaces es tratado de una manera especial en nuestro motor de búsquedas. La mayoría de los motores de búsqueda asocian el texto de un enlace con la pagina donde está dicho enlace. Nosotros además, lo asociamos con la página a la que apunta. Esto presenta varias ventajas. Primero , el texto ancla a menudo proporcionan descripciones mas precisas de las paginas web que las páginas en sí mismas. Segundo, el texto ancla puede existir para documentos que no puedan ser indexados por un motor de busqueda basado en búsqueda de texto, como las imágenes, programas y bases de datos.Esto permite devolver paginas web que no hayas sido en realidad rastreadas. Nótese que las páginas que no han isdo rastreadas pueden causar problemas, ya que no han sido validadas antes de devolverlas al usuario. En este caso, el motor de búsqueda puede incluso devolver una página que nunca ha existido, pero tenia hiper enlaces apuntando a ella. DE todos modos es posible seleccionar los resultados para que este problema raramente ocurra.

Esta idea de la propagación del texto ancla a la pagina desde la que se hace referencia fue implementada en el World Wide Web Worm [McBryan 94] especialmente porque ayuda a buscar información que no sea texto, y expande la cobertura de la búsqueda con menos domucnetos descargados. Nosotros usamos la propagación de texto ancla, principalmente puede ayudar a proporcionar mejores resultados en las búsquedas. Usando texto ancla de forma eficaz es técnicamente complicado debido a la gran cantidad de datos que tienen que ser procesados. De las 24 millones de paginas rastreadas actualmente por nosotros, hemos indexado sobre 259 millones de texto ancla.

2.3 Otras características

Aparte del PageRank y el uso de texto ancla, Google tiene otras varias características. Primero, guarda una copia local de la información de todos los resultados obtenidos, lo cual hace extender el uso de la proximidad de la búsqueda. Segundo, Google guarda algunos detalles de la presentación visual de las webs tales como el tamaño de la letras. Que el tamaño de las letras de una palabra sea mayor, indica que es mas importante que otras de menor tamaño. Tercero, el contenido total del HTML de las webs es almacenado en un repositorio (localmente).

3 Trabajos relacionados

La investigación en búsquedas en la web tiene una corta y concisa historia. El World Wide Web Worm (WWWW) [McBryan 94] fue uno de los primeros motores de búsqueda. A éste le siguieron varios mas que del ámbito académico se pasaron al comercial. Sobre el crecimiento de la Web comparado con la importancia de los motores de búsqueda, hay varios artículos importantes sobre recientes motores de búsqueda [Pinkerton 94]. De acuerdo a Michael Mauldin (científico jefe de Lycos Inc) [Mauldin], "los distintos servicios (incluyendo Lycos) guardan una estrecha relación con las bases de datos". De todos modos, ha habido una gran cantidad de trabajos en características especificas de los motores de búsqueda. Especialmente se puede hablar del trabajo el cual consigue resultados mediante el post-procesamiento de resultados de motores de búsqueda comerciales, o producir motores de búsqueda “personalizados” de pequeña escala. Finalmente, ha habido mucha investigación en cuanto a la recuperación de la información, especialmente en colecciones de datos bien controladas. En las próximas dos secciones, discutiremos sobre algunas áreas donde se necesita mas investigación para

poder trabajar mejor en la red.

3.1 Recuperación de la información

El estudio sobre sistemas de recuperación de la información se remonta muchos años atrás y está bien desarrollado [Witten 94]. De todos modos, la mayoria de la investigaciones en recuperación de la información se encuentra en pequeñas colecciones como coleccionesde artículos científicos. De hecho, la primera prueba experimental para la recuperación de la información, la Text Retrieval Conference [TREC 96], usa una colección bastante pequeña y bien controlada. La prueba "Very Large Corpus" tiene solamente 20GB comparados con los 147GB obtenidos de nuestros rastreos en 24 millones de paginas webs. Las cosas que normalmente funcionan bien enla TREC (Text Retrievla Conference) a menudo no producen buenos resultados en la web. Por ejemplo, el modelo standard de vector espacial intenta devolver el documento que mas se aproxime a la consulta, dado que tanto la consulta como el documento son vectores definidos por las ocurrencias de las palabras que contienen. En la web, esta estrategia a menudo devuelve muy pocos documentos que tengan las consulta mas algunas palabras mas. Por ejemplo, nosotros hemos visto que un motor de búsqueda grande devuelve una pagina conteniendo solamente "Bill Clinton apesta" y una imagen de la consulta "Bill Clinton". Algunos se quejan que la culpa es del usuario que debería afinar mas sus consultas, posición que nosotros rechazamos completamente. Si un usuario realiza una consulta “Bill Clinton” debería obtener datos suficentemente relevantes debido a la gran cantidad de información que exsite en la web sobre este tema. Con ejemplos como éste, creemos que los estudios sobre la recuperación estándar de la información deben ser ampliados para tratar de una forma mas efectiva con la web.

3.2 Diferencias entre la Web y las colecciones bien controladas

La web es una vasta colección de documentos completamente incontrolados. Los documentos en la web presentas una extrema variación interna en los documentos y también de forma externa, la meta información que está disponible. Por ejemplo, los documentes difieren internamente en su idioma (tanto humano como de programación), vocabulario (direcciones de email, enlaces, códigos postales, números de teléfono, números de producto), tipo o formato (texto, HTML, PDF, imágenes, sonidos) e incluso pueden ser generados por un ordenador (archivos de log o salida de una base de datos). Por otra parte, nosotros definimos meta información externa que incluye cosas como la reputación de la fuente, frecuencia de actualización, calidad, popularidad o uso y referencias.

Otra gran diferencia entre la web y las tradicionales colecciones bien controladas es que virtualmente no hay control sobre que gente puede ponerlas en internet. Ésto junto al hecho que existen compañías que con motores de búsqueda manipulados intencionalmente para producir unos determinados resultados, lo convierte en un serio problema. Este problema no ha sido tenido en cuenta en los sistemas tradicional de recuperación de la información. También sería interesante fijarse en que los esfuerzos en la meta-información de las webs ha fracasado estrepitosamente en los motores de búsqueda, porque no se puede garantizar que el texto de una web haya sido puesto ahí para hacer engañar al motor de búsqueda. Hay incluso compañías especializadas enésto.

4 Anatomía del Sistema

Para empezar, describiremos por encima cómo es la arquitectura. Luego nos pararemos mas en detalle en la descripción de algunas estructuras de datos. Por último, veremos las aplicaciones mas importantes en detalle: el rastreo, la indexación y la búsqueda.

4.1 Visión en conjunto de la arquitectura Google

En esta sección veremos a grandes rasgos las características del funcionamiento del sistema completo, que mostramos en la Figura 1. Las sesiones posteriores versarán sobre las aplicaciones y estructuras de datos no vistas en esta sección. La mayor parte de Google está implementada en C ó C++ para una mayor eficiencia y es ejecutada sobre Solaris o Linux.

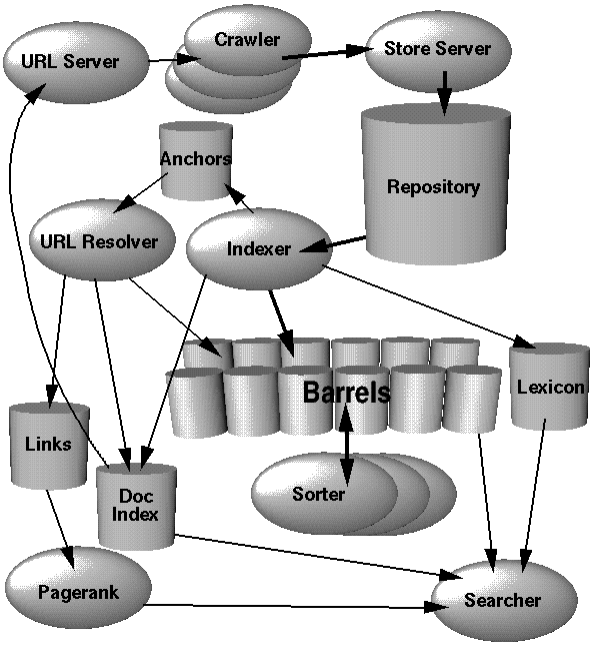

Figure 1. High Level Google Architecture

En Google, el rastreo de la web (la descarga de páginas web) es realizada por varios sistemas distribuidos rastreadores. Un servidor de URLs le pasa a los sistemas rastreadores las páginas a rastrear. Éstas paginas web son enviadas a sistemas de almacenamiento. El servidor de almacenamiento las comprime y las guarda en un repositorio. A cada página web se le asocia un numero de identificación denominado docID el cual es asignado cada vez que una nueva URL es examinada. La función de indexación es llevada a cabo por el indexador y el clasificador. El indexador lleva a cabo un numero determinado de funciones. Lee el repositorio, descomprime los documentos y los analiza. Cada documento se convierte a un conjunto de ocurrencias de palabras llamadas hits. Los hits almacenan la palabra, posición en el documento, tamaño de letra y si es letra mayúscula o minúscula. El indexador distribuye estos hits en unas "cubetas", creando una primera instancia de índice ordenado. El indexador lleva a cabo otra importante función. Analiza todos los enlaces de una página y almacena toda la información importante en un archivo ancla. Este archivo contiene bastante información que determina desde y adonde apunta cada enlace.

El URLresolver lee los archivos ancla y convierte las URLs relativas en URLs absolutas y se le asigna el docID a cada URL. Pone éste archivo ancla en el primer índice que se ha creado con las cubetas, asociándolo con el docID al que apunta su texto ancla. También genera una base de datos de enlaces que son pares de docIDs. Ésta base de datos es usada para calcular los PageRanks de todos los documentos.

El sorter (clasificador) coge esas cubetas, que están organizadas según los docIDs delos documentos contenidos (ésto es de manera simplificada, para mas detalle ver Seccion 4.2.5), y los reclasifica según el wordID para generar un índice inverso. Para este proceso se necesita de un pequeño espacio en memoria temporal. El sorter (clasificador) también produce una lista de wordIDs y

desplazamientos dentro del índice invertido. Un programa llamado DumpLexicon toma esta lista junto con el léxico producido por el indexador y genera un nuevo léxico que será usado por el searcher (buscador). El searcher (buscador) es ejecutado en un servidor web y usa el lexico construido por DumpLexicon junto al índice invertido y los PageRanks calculados.

4.2 Estructuras de Datos principales

Las estructuras de datos de Google están optimizadas para posibilitar el rastreo, indexado y búsqueda de una gran colección de documentos a un muy bajo coste. Aunque las CPUs han ido evolucionando considerablemente a lo largo de los años, el acceso a disco aun sigue requiriendo sobre 10ms . Google está diseñado para evitar el acceso a disco siempre que sea posible, y esto ha sido una influencia considerable a la hora de diseñar la estructura de datos.

4.2.1 BigFiles

Los BigFiles son archivos virtuales ocupando múltiples sistemas de archivos y son direccionados por 64 bit. El reparto entre los múltiples sistemas de archivo es llevado a cabo automáticamente. El paquete BigFiles también se encarga de la reserva del espacio y liberación del mismo de los descriptores de archivo, ya que los sistemas operativos usados no nos proporcionan un sistema adecuado a nuestras exigencias. BigFiles también soporta opciones simples de compresión.

4.2.2 El Repositorio

El repositorio contiene el código HTML completo de todas las páginas web. Cada página es comprimida usando la librería zlib (ver RFC1950). El hecho de haber elegido esta técnica de compresión es el resultado de adquirir un compromiso entre velocidad y ratio de compresión. Escogimos la velocidad que proporciona zlib frente a una mejor compresión ofrecida por bzip. El ratio de compresión de bzip era aproximadamente de 4 a 1 en el repositorio, comparado con el 3 a 1 de zlib. En el repositorio los documentos son almacenados uno tras otro y son prefijados por el docID, longitud y URL como se puede ver en la figura 2. El repositorio no requiere mas estructuras de datos para acceder a los datos. Esto ayuda con las consistencia en los datos y hace que el desarrollo sea mucho mas fácil; así podemos reconstruir todas las demás estructuras de datos usando solamente el repositorio y un archivo que contienen los errores devueltos por el crawler (rastreadeor)

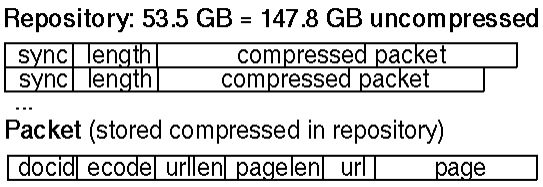

Figure 2. Repository Data Structure

4.2.3 Document Index

El Document Index (Indice de documento) contiene información acerca de cada documento. Es un indice de tipo ISAM de anchura fija (modo de acceso secuencial al índice), ordenado por el docID. La información almacenada en cada entrada del índice incluye el estado actual del documento, un puntero al repositorio, un código de comprobación de validez (checksum) y varias estadísticas. Si el documento ha sido rastreado, tambien contiene un puntero a una variable de archivo llamado docinfo ,

el cual contiene su URL y título. Si no, el puntero solo apunta a las lista de URL (URLlist) las cual solo contiene la URL. Esta decisión de diseño se tomó en aras de un diseño de la estructura de datos razonablemente compacta y para poder permitir que la búsqueda de un registro pudiera tomar como mucho un solo acceso a disco.

Adicionalmente, hay un archivo que se usa para convertir las URLs a docIDs. Es una lista de sumas de comprobación de URL con sus correspondientes docIDs, que son clasificados según sus checksum. Para encontrar el docID de una URL en particular, se calcula el checksum de la URL y se realiza una búsqueda binaria en el archivo de checksums para encontrar su docID. Las URLs pueden ser convertidas en docIDs (en lotes) haciendo una mezcla con éste archivo. Esta es la técnica que el URLresolver utiliza para devolver la URL de un docID dado. Este modo en lote de actualización es crucial porque de otro modo nosotros deberíamos hacer una búsqueda en disco por cada enlace, con lo cual nos llevaría mas de un mes de tiempo en realizar este proceso para nuestros 322 millones de enlaces que tenemos actualmente.

4.2.4 Lexicon (léxico)

El léxico tiene varias formas distintas. Un cambio importante comparado con los sistemas precedentes es que nuestro léxico cabe en memoria a un coste razonable. En nuestro sistema actual, somos capaces de meter el léxico en un la memoria de una máquina con 256 MB de memoria principal. El léxico actual contiene 14 millones de palabras (aunque algunas palabras poco frecuentes no fueron añadidas al léxico). Está implementado en dos partes – una lista de palabras (concatenadas pero separadas por espacios) y una tabla hash de punteros. Además, para otras funciones, la lista de palabras contiene información auxiliar, la cual se escapa del objetivo de éste articulo.

4.2.5 Hit Lists

Una lista de hits es una lista que almacena las ocurrencias de una determinada palabra en un document en particular incluyendo posición, tipo de letra, y si es mayúscula o minúscula. Ésta lista ocupa la mayoría del espacio tanto del índice como del índice invertido. Por ello, es importante representarla los mas eficientemente posible. En su momento barajamos varias posibilidades para codificar la posición, tipo de letra, etc – una codificacion simple (una terna de enteros), una codificación compacta (una colocación manual de bits), y una codificación Huffman. Al final escogimos la codificación manual optimizada ya que requería bastante menos espacio que la codificación simple y la Huffman. Los detalles de los hits se muestran en la Figura 3.

Nuestra codificación compacta usa dos bytes para cada hit. Hay dos tipos de hits: fancy hits y plain hits. Los fancy hits incluyen los hits que aparecen en una URL, titulo, texto ancla o etiquetas meta. Los plin hits incluyen todo lo demás. Un plain hit está formado por un bit que indica si es mayúscula o minúscula la capitalización, tamaño de fuente, y 12 bits de la posición de la palabra en el documento (las posiciones mayores a 4095 son etiquetadas como 4096). El tamaño de letra se representa de forma relativa a al resto del documento usando 3 bits (solo 7 valores se usan normalmente porque 111 indica un fancy hit). Un fancy hit consiste de un bit que indica si es mayúscula o minúscula, el tamaño de fuente se establece a 7 para indicar que es un fancy hit, 4 bits para codificar el tipo de fancy hit, y 8 bits de posición. Para los hits anclas, los 8 bits de posición son divididos en 4 bits para la posición y otros 4 para un hash del docID en el que el ancla aparece. Ésto nos da un a limitada búsqueda de

frases dependiendo de la cantidad de anclas que haya para una palabra en particular. Esperamos actualizar el modo en que los hits ancla son almacenados para mejorar la precisión en la posición y campos docIDhash. Usamos el tamaño de fuente de forma relativa, ya que no nos interesa por ejemplo que dos documentos que sean idénticos pero uno con el tamaño de letras mas grande, sea considerando como mas importante.

La longitud de una lista de hits es almadenado incluso antes que los hits propiamente dichos. Para ahorrar espacio, la longitud de la lista de hits se combina con la wordID del índice principal y el docID del índice invertido. Esto limita a 8 5 bits respectivamente (hay algunos trucos que permiten 8 bits que se piden prestado del wordID). Si la longitud fuera mayor, un código especial se usa en estos bits, y los 2 próximos bytes contienen la verdadera longitud.

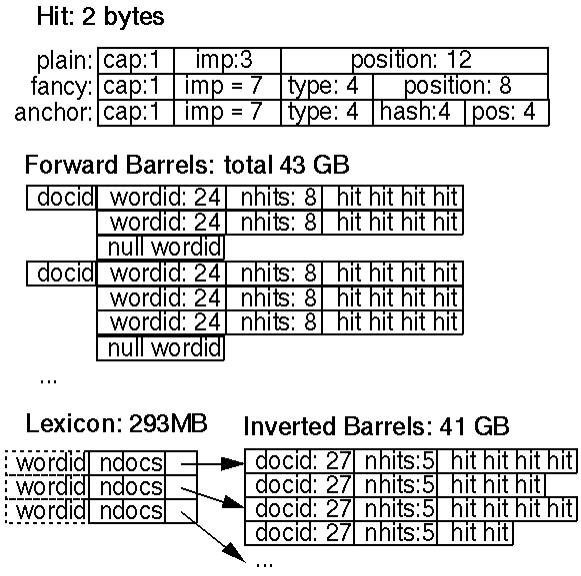

Figure 3. Forward and Reverse Indexes and the Lexicon

4.2.6 Índice principal

El índice principal ya se encuentra en realidad parcialmente ordenado. Se almacena en cubetas (unas 64). Cada cubeta almacena un rango de wordIDs. Si un documento contiene palabras que vana una determinada cubeta, el docID se almacena en esa cubeta, seguido de una lista de wordIDs con listas de hits que corresponde a esas palabras. Este esquema requiere prácticamente mas almacenamiento a causa de los docIDs duplicados pero la diferencia es muy pequeña para una numero razonable de cubetas y ahorrara un tiempo y complejidad considerable en la fase final de indexado realizado por el clasificador (sorter). Mas aun, en lugar de almacenar los wordIDs reales, nosotros almacenamos cada wordID como una diferencia relativa entre el mínimo wordID que va a la cubeta y el wordID en sí. De este modo, podemos usar solo 24 bits para los wordID en las cubetas sin ordenar, dejando 8 bits para la longitud de la lista de hits.

4.2.7 Indice invertido

El índice invertido consiste de las mismas cubetas que el indice principal, excepto que estas han sido procesadas por el clasificador (sorter). Para cada wordID válida, el léxico contiene un puntero a la cubeta a la que va el wordID. Ésta apunta a una lista de docID junto a sus correspondiente histlist. Esta lista de docID representa todas las ocurrencias de esa palabra en todos los documentos.

Un aspecto importante es el orden en el que los docID debieran aparecer en las lista de docIDs. Una solución simple es almacenarlos por el docID. Esto permite una rápida mezcla de diferentes doclists para múltiples consultas de palabra. Otra opción es almacenarlos clasificados por ranking de ocurrencia de la palabra en cada documentos. Ésto hace que la respuesta a una consulta de una palabra sea trivial. De todos modos, mezclar es mucho mas difícil. Esto también hace el desarrollo mucho mas difícil, ya que un cambio en la función que genera el ranking conlleva volver a construir el índice. Nosotros escogimos un compromiso entre estas dos opciones, mantener dos conjuntos de

cubetas invertidas – un conjunto para listas de hits que incluyeran titulo o hits ancla y otro conjunto para todas las listas de hits. De este modo, comprobamos el primer conjunto de cubetas y si no hay suficientes coincidencias en estas cubetas, entonces comprobamos las otras mas grandes.

4.3 Rastreando la web

Ejecutar un rastreador de webs es una tarea ardua. Hay muchos aspectos tenciso a tener en cuantam incluso mas importante, cuestiones sociales. Rastrear es la tarea mas frágil, ya que conlleva interactuar con cientos de miles de servidores web y varios servidores de nombre los cuales estan fuera de nuestro control.

Para llegar a esos cientos millones de webs, Google tiene un rápido sistema de indexado distribuido. Un único URLserver sirve listas de URLs a varios rastreadores (por lo general usamos 3). Tanto el URLserver como los rastreadores están implementados en Python. Cada rastreador mantiene hasta 300 conexiones abiertas a la vez. Esto es necesario para poder leer webs a una velocidad considerable. En picos de velocidad, el sistema puede rastrear sobre 100 webs por segundo usando 4 rastreadores. Estamos hablando de casi 600K de datos por segundo. Un escollo en el rendimiento son los DNS lookup. Cada rastreador mantienen su propia cache de DNS por lo que no necesita realizar una resolución de DNS antes de rastrear cada documento. Cada una de las 100 conexiones puede estar en un estado diferente: resolviendo DNS, conectando al host, enviando peticiones y recibiendo peticiones. Ésto hace que el rastreador sea un componente complejo del sistema. Usa Entrada/Salida asíncrona para gestionar los eventos y varias colas para cambiar de un estado a otro.

La consecuencia de que un rastreador que está ejecutándose y conectando con mas de millón y medio de servidores y genera millones y millones de lineas de logs, es que recibamos innumerables llamadas telefónicas y correos electrónicos. A causa de la gran cantidad que cada vez se conecta mas a internet, hay mucho que no saben lo que es un rastreador web ya que es el primero que ellos han visto. Casi a diario recibimos emails del tipo: “Wow, visitaste muchas páginas de mi web. ¿Te gustaron?” Otros muchos desconoces los mecanismo de exclusión de robots, y piensan que su web por el mero hecho de contener un mensaje del tipo “Esta pagina tiene CopyRight y no puede ser indexada” ya no va a ser indexada por un rastreador, ya que un rastreador no entiende estos mensajes. También por la gran cantidad de datos involucrados, pasan cosas inesperadas. Por ejemplo, nuestro sistema intento rastrar un juego online. Esto devolvió muchos mensajes basura en la mitad del juego. De todos modos estos son problemas fáciles d eresolver. Aunque este problema no apareció hasta que habiamos indexado decenas de millones de paginas. Debido a la gran variedad de servidores y webs, es complicado hacer un rastreador a prueba de fallos, por lo que se requiere que su diseño sea lo mas robusto posible.

4.4 Indexando la Web

- **Analizando –** Cualquier analizador (parser) que sea concebido apra ejecutarse en la Web, debe contener un vector enorme que maje todos los posibles errores. Esto conlleva desde fallos en el etiquetado html hasta caracteres no Ascii por ejemplo.
- **Indexando documentos en las cubetas – Después** de que cada documento es analizado, se codifica en varias cubetas. Cada palabra se convierte a un wordID usando una tabla hash en memoria – el léxico (lexicon). Las nuevas entradas a la tabla hash del léxico son registradas en un archivo. Una vez las palabras son convertidas a wordID, sus ocurrencias en el documento

actual se traduce a listas de hits y se guardan en cubetas. La principal dificultad es que el léxico tiene que ser compartido. A diferencia de la fase de indexación en la que el léixco tiene que ser compartido aquí no hace falta. En lugar de compartirlo, se toma la determinación de escribir un registro con las palabras extra que no estaban en el léxico base, el cual nosotros fijamos en 14 millones de palabras. De esta manera se multiplica los indexadores que pueden ejecutarse en paralelo y ese registro de palabras extra es procesado por un indexador final.

- **Clasificación** – Para generar el índice invertido, el clasificador toma cada una de las cubetas y las clasifica por el wordID para producir una cubeta invertida clasificada por titulo y hits anclan y otra cubeta con el texto completo. Este proceso tiene lugar en una sola cubeta al mismo tiempo, de este modo requiere poca memoria temporal. También paralelizamos la fase de clasificación para poder usar tantas maquinas como tengamos simplemente ejecutando varios clasificadores, con lo cual se puede procesar varias cubetas a la vez. Si las cubetas no cogen en memoria principal, el clasificador las divide en partes mas pequeñas. Después el clasificador carga cada parte en memoria, la clasifica y guarda su contenido en la cubeta invertida ya clasificada y en la cubeta con el texto completo.

4.5 Búsqueda

La meta de la búsqueda es proveer de calidad en los resultados de las búsquedas. Muchos motores de búsqueda comerciales parecer haber mejorado en términos de eficiencia. Nosotros lo hemos enfocado mas bien en la calidad de la búsqueda, aunque creemos que nuestras soluciones son escalables a estos motores comerciales con un poco mas de esfuerzo. El proceso de evaluación de las peticiones se muestra en la Figura 4.

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.
   Sort the documents that have matched by rank and return the top k.

Figure 4. Google Query Evaluation

Para pone run limite en el tiempo de respuesta, una vez que un numero concreto (normalmente 40,000) de documentos que coinciden con la búsqueda son encontrados, el buscador (searcher) pasa al punto 8 de la Figura 4. Esto quiere decir que es posible que sub-optimos resultados se devolverían. Estamos actualmente investigando otros modos para corregir este problema. Antes, clasificábamos los hits en función del PageRank, lo cual parecía que mejoraba la situación.

4.5.1 El sistema de Ranking

Google mantiene mucha mas información sobre documentos web que los típicos motores de búsqueda. Cada hitlist incluye posición, tipo de letra e información sobre capitalización de las letras. Adicionalmente, tenemos en cuenta el texto ancla y el PageRank. Combinar toda esta información para generar un ranking es difícil. Diseñamos nuestra función de ranking para que ningún factor tuviera demasiada influencia. Primero, consideramos el caso mas

simple – una búsqueda de una sola palabra. Google mira en la lista de hits para esa palabra. Google considera cada hit atendiendo a cada tipo diferente (titulo, texto ancla...), cada uno de lo cuales tiene su propio peso de importancia. Se construye un vector indexado con estos datos. Google cuenta el numero de hits para cada tipo de la lista de hits. Éste numero se convierte a numero*peso. El numero*peso crece linealmente hasta que llega a un determinado numero en el que practica ya no crece. YA tenemos por un lado el vector de tipos y otro el vector de numero*peso, se realiza el producto escalar entre ambos para obtener una puntuación IR. Finalmente esta puntuación IR se combina con el PageRank para obtener la puntuación final del documento dentro del ranking.

Para una búsqueda de múltiples palabras el proceso es mas complicado. Ahora las distintas listas de hits deben ser escaneadas a la vez para que se puntúen mas alto aquellos que aparezcan juntos que los que aparezcan solos. Se buscan las correspondencias entre los hits de las distintas listas para que se emparejen aquellos hits cercanos. Para cada conjunto de hits, se calcula una proximidad. La proximidad se basa en la distancia a la que se encuentran los hits en el documento (o texto ancla) que se clasifica en 10 valores que van desde una coincidencia total a ninguna. No solo se computa para cada tipo de hit, sino también para cada tipo y proximidad. Cada par de tipo y proximidad tiene una peso tipo-proximidad. Estos cálculos se convierten a calculo-peso y realizamos el producto escalar de los calculo-peso con los pesos tipo-proximidad para calcular un IR. Estos cálculos pueden visualizarse usando el modo de depuración de Google, lo cual ha sido muy útil a la hora de desarrollar el sistema.

4.5.2 Feedback

La función de ranking tiene muchos parámetros como los pesos de importancia de los tipos y los pesos de proximidad de los tipos. Dando a entender casi que los valores correctos para estos parámetros es casi un arte. Para ésto, nosotros tenemos un mecanismo de comunicación con el usuario del motor de búsqueda. Un usuario confiable puede opcionalmente evaluar todos los resultados que son devueltos por el motor. Esta evaluación se guarda. Luego cuando cambiemos el mecanismo de la función de ranking, podemos ver el impacto de este cambio en todas las búsquedas previas que fueran evaluadas. lejos de ser perfecto, esto nos da alguna idea de como un cambio en la función del ranking afecta a los resultados de la búsqueda.

5 Resultados y rendimiento

La medida mas importante de un motor de búsqueda es la calidad en los resultados de las búsquedas. Nuestra experiencia con Google nos ha demostrado que la calidad de sus resultados es mejor que la mayoría de los motores comerciales. Como ejemplo que ilustra el uso del PageRank, texto ancla y proximidad, la Figura 4 muestra los resultados obtenidos al buscar "bill clinton". Estos resultados muestra algunas de las características de Google. Los resultados son agrupados por el servidor. Esto ayuda considerablemente a cribar grandes conjuntos de resultados. Varios resultados son del dominio whitehouse.gov que es lo que uno puede esperar de una búsqueda de éste tipo. Normalmente, la mayoría de los motores comerciales son devolverían ningún resultado de éste dominio. Dése cuenta de que no hay titulo para el primer resultado. Esto es porque no fue rastreado. En lugar de eso, Google echa mano al texto ancla para determinar que es un buen resultado a mostrar. De forma parecida, el quinto resultado es una dirección de correo electrónico, que por supuesto no es rastreable. Es también un resultado del texto ancla.

Query: bill clinton

http://www.whitehouse.gov/

100.00% (no date) (0K)

http://www.whitehouse.gov/

Office of the President

99.67% (Dec 23 1996) (2K)

http://www.whitehouse.gov/WH/EOP/OP/html/OP_Home.html

Welcome To The White House

99.98% (Nov 09 1997) (5K)

http://www.whitehouse.gov/WH/Welcome.html

Send Electronic Mail to the President

99.86% (Jul 14 1997) (5K)

http://www.whitehouse.gov/WH/Mail/html/Mail_President.html

mailto:president@whitehouse.gov

99.98%

mailto:President@whitehouse.gov

99.27%

The "Unofficial" Bill Clinton

94.06% (Nov 11 1997) (14K)

http://zpub.com/un/un-bc.html

Bill Clinton Meets The Shrinks

86.27% (Jun 29 1997) (63K)

http://zpub.com/un/un-bc9.html

President Bill Clinton - The Dark Side

97.27% (Nov 10 1997) (15K)

http://www.realchange.org/clinton.htm

$3 Bill Clinton

94.73% (no date) (4K)

http://www.gatewy.net/~tjohnson/clinton1.html

Figure 4. Sample Results from Google

Todos los resultados son páginas de razonable calidad y se puede comprobar que ninguna tiene el enlace roto. Esto es así porque todas tienen un PageRank alto. Los PageRanks son las barras de color rojo de porcentaje. Finalmente, no hay ningún Bill que no sea Clinton o un Clinton que no sea un Bill. Esto es porque hacemos especial hincapié en la proximidad de las palabras. De todos modos, para poder demostrar la eficacia de Google necesitaríamos de un estudio mucho mayor que no tiene cabida en este artículo, asi que le emplazamos a que lo pruebe usted mismo en http://google.stanford.edu.

5.1 Requerimientos de almacenamiento

Al igual que en la calidad de búsqueda, Google está diseñado para ser escalable en cuanto al coste que

deriva del crecimiento de la web. Un aspecto de ésto es usar eficientemente el almacenamiento. La Tabla 1 tiene un breve resumen de algunas estadistas y requerimientos de almacenamiento de Google. debido a la compresión el tamaño total del repositorio es de unos 53 GB, sobre una tercera parte del total de datos que almacena. Con los precios actuales de los discos duros, el repositorio sale barato. El total de datos usado para las búsquedas requieren de unos 55 GB. La mayoría de las búsquedas pueden ser resueltas usando solo el índice invertido corto. Con una mejor codificación y compresión del Indice de Documentos, se podría tener un motor de búsqueda de calidad con tan solo 7GB de un disco duro de un ordenador moderno.

5.2 Rendimiento del sistema

Estadísticas de Páginas Web	
Numero de webs buscadas	24 million
Numero de URL vistas	76.5 million
Numero de correos electronicos	1.7 million
Number de Errores 404	1.6 million

Tabla 1. Estadísticas

Para un motor de búsqueda es muy importante que el proceso de rastreo e indexación sea eficiente. De esta manera se logra mantener una información actualizada y los cambios importantes en el sistema pueden ser comprobados relativamente rápido. Para Google, las operaciones mas importantes son el rastreo, indexado y clasificación. Es difícil de determinar cuanto tiempo llevará un rastreo, mas que nada por que pueden que los discos se llenen, los servidores de nombre se caigan o que ocurra cualquier otro problema que interrumpa el sistema. En total suele llevar casi 9 días en descargar las 26 millones de paginas (incluyendo los errores). DE todos modos, una vez el sistema esté ejecutándose correctamente, se ejecutarán los procesos mas rapidamente descargando 11 millones de páginas en solo 63 horas, con un promedio de 4 millones de páginas por dia o 48.5 páginas por segundo. Ejecutamos el rastreador y el clasificador a la vez. Esto es importante ya que dedicamos el suficiente tiempo en optimizar el indexador para que no se producieran cuellos de botella. Estas optimizaciones incluían actualizaciones generales y copias de las estructuras de datos críticas en un disco local. El indexador procesaba alrededor de 54 páginas por segundo. Los clasificadores pueden ser ejecutados en paralelo; usando 4 maquinas, el proceso completo de clasificación puede llevar sobre 24 horas.

5.3 Rendimiento de la búsqueda

La mejora del rendimiento de la búsqueda no era el principal objetivo de éste artículo hasta ahora. La versión actual de Google responda a la mayoría de las peticiones de búsqueda entre 1 a 10 segundos. Esta vez es llevado a cabo mayormente por discos de E/S mas que por NFS (ya que los discos están diseminados por todas las maquinas). Google no tienen ninguna optimización como el cacheo de de búsquedas, subíndices de términos comunes y otras optimizaciones típicas. Nuestra objetivo era acelerar Google mediante la distribución y el hardware, software y mejoras algorítmicas. Nuestro objetivo es poder gestionar varios cientos de peticiones por segundo. La Tabla 2 algunos tiempos de ejemplos de búsquedas de a versión actual de Google. Están repetidas para mostrar la mejora debido al cacheo de las E/S.

Estadísticas de almacenamiento	
Tam. total de webs buscadas	147.8 GB
Repositorio comprimido	53.5 GB
Indice corto invertido	4.1 GB
Indice completo invertido	37.2 GB
Lexico	293 MB
Datos ancla temporales (no son los totales)	6.6 GB
Indice de documentos incluyendo los datos de anchira variable	9.7 GB
Enlace de base de datos	3.9 GB
Total sin el Repositorio	**55.2 GB**
Total con el Repositorio	**108.7 GB**

	Búsqueda inicial		**Búsuqeda repetida (E/S cacheada)**	
búsqueda	**Tiempo CPU**	**Tiempo Total**	Tiempo CPU	**Tiempo Total**
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

Tabla 2. Tiempos de búsqueda

6 Conclusiones

Google está diseñado para ser un motor de búsqueda escalable. La meta principal es proporcionar resultados de alta calidad sobre una cada vez mas creciente WWW. Google incorpora varias técnicas de mejora en la calidad de búsqueda como son PageRank, texto ancla e información de proximidad. Además, Google es un sistema completo capaz de rastrear webs indexarlas y procesar peticiones de búsqueda sobre ellas.

6.1 Trabajo futuro

Un motor de búsqueda a gran escala es un complejo sistema y aun mucho se debe hacer. Nuestras metas inmediatas son mejorar la eficiencia de búsqueda y escalar nuestro sistemas para que permita gestionar 100 millones de páginas web. Algunas mejoras simples para la eficiencia incluyen el cacheo de las consultas o peticiones, gestión de espacio de disco inteligente y subindices. Otro área que requiere muchas mejoras son las actualizaciones delas webs indexadas. Debemos tener algoritmos que nos permitan dilucidad cuando una web debe ser rastreada de nuevo en caso de ser una versión antigua. Trabajos en este sentido se comentan en [Cho 98]. Un area prometedora en la investigación es el uso de proxy-caches para construir bases de datos con las webs mas demandadas. Estamos planeando añadir pequeñas características que ya permiten buscadores comerciales, como son operadores booleanos, negación y stemming (reducir una palabra a su raíz). De todos modos, otras características están empezando a ser exploradas como la interacción con el usuario y el clustering (Google soporta actualmente un clustering basado en nombre de dominios). También planeamos dar soporte al contexto de usuario (cómo tener en cuenta dónde se encuentra el usuario) y un resumen de

resultados. Tambien estamos trabajando en extender el uso de la estructura de enlaces. Algunos experimentos simples indican que PageRank puede ser personalizado incrementando el peso de una pagina principal de usuario o favoritos. Tambien, para el texto de enlaces, estamos experimentando añadir otro texto alrededor del enlace ademas del del enlace propiamente. Un motor de búsqueda es un entorno rico en experimentación de ideas. Estamos aun empezando asi que es de esperar que esta sección de Trabajo Futuro se amplie bastante en un futuro cercano.

6.2 Búsqueda de alta calidad

El problema mas grande al que hacen frente los usuarios de los motores de búsqueda es la calidad de los resultados que obtienen en sus búsquedas. Aunque estos resultados son a veces divertidos y expanden el horizonte de los usuarios, a menudo son frustrantes y consumen un preicoso tiempo. Por ejemplo, el top de resultados de una búsqueda para "Bill Clinton" en uno de los buscadores comerciales mas conocidos era el de Bill Clinton Joke of the Day: April 14, 1997. Google ha sido diseñado para devilver resultados de calidad al mismo ritmo que la web va creciendo rápidamente, de forma que se pueda obtener información de forma rápida. Para lograr esto, Google hace un uso intensivo de la estructura de enlaces web y del texto ancla, asi mismo se beneficia del PageRank para priorizar resultados. El análisis de la estructura web a través de PageRank le permite a Google evaluar la calidad de las páginas web. El uso de enlaces de texto como una descripción de donde apunta ese texto, ayuda al motor de busque a devolver resultados de calidad. Finalmente, el uso de la información de proximidad ayuda a saber que resultados son mas importantes que otros.

6.3 Arquitectura Escalable

A parte de la calidad de búsqueda, Google está diseñado pensando en que pueda ser facilmente escalable. Debe ser eficiente tanto en tiempo como espacio, y factores constantes son muy importantes cuando hablamos de la Web entera. Implementado Google, hemos visto cuellos de botella en la CPU, acceso a memoria, capacidad de la memoria, busuqedas en disco, capacidad del disco, y E/S en red. Google ha salido indemne en bastante cuellos de botella durante varias operaciones. La mayor parte de las estructuras de datos de Google hacen un uso eficiente del espacio disponible en disco. Ademas, el rastreo, indexado y clasificacion son bastantes eficientes a la hora de construir un indice con una parte importante de la Web – 24 millones de páginas, en menos de una semana. Esperamos ser capaces de construir un indice de 100 millones de páginas en menos de un mes.

6.4 Una Herramienta de Investigación

Ademas de ser un motor de búsqueda de alta calidad, Google es una herramienta de investigación. Los datos que Google ha ido recogiendo ya han sido publicados en muchos artículos de investigación que han sido enviadas a conferencias y otras tantas que están en camino. Una investigación reciente como en [Abiteboul 97] ha mostrado un numero de limitaciones en las consultas que deben ser respondidas sin tener la Web localmente. Esto significa que Google (un sistema similar) no es solo una valiosa herramienta de investigación, sino que es necesaria para una gran gama de aplicaciones. Esperamos que Google sea un recurso para investigadores y buscadores de todo el mundo y que seamos la referencia en la tecnología de motor de búsqueda.

7 Agradecimientos

Scott Hassan y Alan Steremberg han sido imprescindibles en el desarrollo de Google. Sus magníficas contribuciones son irreemplazables, y por ello les estamos muy agradecidos. Tambien nos gustaria agradecer a Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman, y Terry Winograd y por extension al grupo completo WebBase por su ayuda y pountos de vista. Por ultimos, nos gustaria reconocer la generoso soporte y patrocinio de IBM, Intel, y Sun. La investigación aquí descrita ha sido dirigida en parte a traves del Stanford Integrated Digital Library Project, apoyado por National Science Foundation bajo el acuerdo de cooperacion IRI-9411306. La financiación de este acuerdo de colaboración ha sido llevado a cabo tambien por DARPA y la NASA, y por el Interval Research, y los socios industriales de la Stanford Digital Libraries Project.

References

- Best of the Web 1994 -- Navigators http://botw.org/1994/awards/navigators.html
- Bill Clinton Joke of the Day: April 14, 1997. http://www.io.com/~cjburke/clinton/970414.html.
- Bzip2 Homepage http://www.muraroa.demon.co.uk/
- Google Search Engine http://google.stanford.edu/
- Harvest http://harvest.transarc.com/
- Mauldin, Michael L. Lycos Design Choices in an Internet Search Service, IEEE Expert Interview http://www.computer.org/pubs/expert/1997/trends/x1008/mauldin.htm
- The Effect of Cellular Phone Use Upon Driver Attention http://www.webfirst.com/aaa/text/cell/cell0toc.htm
- Search Engine Watch http://www.searchenginewatch.com/
- RFC 1950 (zlib) ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html
- Robots Exclusion Protocol: http://info.webcrawler.com/mak/projects/robots/exclusion.htm
- Web Growth Summary: http://www.mit.edu/people/mkgray/net/web-growth-summary.html
- Yahoo! http://www.yahoo.com/

- [Abiteboul 97] Serge Abiteboul and Victor Vianu, *Queries and Computation on the Web*. Proceedings of the International Conference on Database Theory. Delphi, Greece 1997.
- [Bagdikian 97] Ben H. Bagdikian. *The Media Monopoly*. 5th Edition. Publisher: Beacon, ISBN: 0807061557
- [Chakrabarti 98] S.Chakrabarti, B.Dom, D.Gibson, J.Kleinberg, P. Raghavan and S. Rajagopalan. *Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text.* Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18, 1998.
- [Cho 98] Junghoo Cho, Hector Garcia-Molina, Lawrence Page. *Efficient Crawling Through URL Ordering.* Seventh International Web Conference (WWW 98). Brisbane, Australia, April 14-18, 1998.
- [Gravano 94] Luis Gravano, Hector Garcia-Molina, and A. Tomasic. *The Effectiveness of GlOSS for the Text-Database Discovery Problem.* Proc. of the 1994 ACM SIGMOD International Conference On Management Of Data, 1994.
- [Kleinberg 98] Jon Kleinberg, *Authoritative Sources in a Hyperlinked Environment*, Proc. ACM-SIAM Symposium on Discrete Algorithms, 1998.

- [Marchiori 97] Massimo Marchiori. *The Quest for Correct Information on the Web: Hyper Search Engines.* The Sixth International WWW Conference (WWW 97). Santa Clara, USA, April 7-11, 1997.
- [McBryan 94] Oliver A. McBryan. GENVL and *WWWW: Tools for Taming the Web. First International Conference on the World Wide Web.* CERN, Geneva (Switzerland), May 25-26-27 1994. http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps
- [Page 98] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Manuscript in progress. http://google.stanford.edu/~backrub/pageranksub.ps
- [Pinkerton 94] Brian Pinkerton, *Finding What People Want: Experiences with the WebCrawler.* The Second International WWW Conference Chicago, USA, October 17-20, 1994. http://info.webcrawler.com/bp/WWW94.html
- [Spertus 97] Ellen Spertus. *ParaSite: Mining Structural Information on the Web.* The Sixth International WWW Conference (WWW 97). Santa Clara, USA, April 7-11, 1997.
- [TREC 96] *Proceedings of the fifth Text REtrieval Conference (TREC-5).* Gaithersburg, Maryland, November 20-22, 1996. Publisher: Department of Commerce, National Institute of Standards and Technology. Editors: D. K. Harman and E. M. Voorhees. Full text at: http://trec.nist.gov/
- [Witten 94] Ian H Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* New York: Van Nostrand Reinhold, 1994.
- [Weiss 96] Ron Weiss, Bienvenido Velez, Mark A. Sheldon, Chanathip Manprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. *HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering.* Proceedings of the 7th ACM Conference on Hypertext. New York, 1996.

Vitae

Sergey Brin received his B.S. degree in mathematics and computer science from the University of Maryland at College Park in 1993. Currently, he is a Ph.D. candidate in computer science at Stanford University where he received his M.S. in 1995. He is a recipient of a National Science Foundation Graduate Fellowship. His research interests include search engines, information extraction from unstructured sources, and data mining of large text collections and scientific data.

Lawrence Page was born in East Lansing, Michigan, and received a B.S.E. in Computer Engineering at the University of Michigan Ann Arbor in 1995. He is currently a Ph.D. candidate in Computer Science at Stanford University. Some of his research interests include the link structure of the web, human computer interaction, search engines, scalability of information access interfaces, and personal data mining.

8 Appendix A: Advertising and Mixed Motives

Currently, the predominant business model for commercial search engines is advertising. The goals of the advertising business model do not always correspond to providing quality search to users. For example, in our prototype search engine one of the top results for cellular phone is "The Effect of Cellular Phone Use Upon Driver Attention", a study which explains in great detail the distractions and risk associated with conversing on a cell phone while driving. This search result came up first because of its high importance as judged by the PageRank algorithm, an approximation of citation importance on the web [Page, 98]. It is clear that a search engine which was taking money for showing cellular phone ads would have difficulty justifying the page that our system returned to its paying advertisers. For this type of reason and historical experience with other media [Bagdikian 83], we expect that advertising funded search engines will be inherently biased towards the advertisers and away from the needs of the consumers.

Since it is very difficult even for experts to evaluate search engines, search engine bias is particularly insidious. A good example was OpenText, which was reported to be selling companies the right to be listed at the top of the search results for particular queries [Marchiori 97]. This type of bias is much more insidious than advertising, because it is not clear who "deserves" to be there, and who is willing to pay money to be listed. This business model resulted in an uproar, and OpenText has ceased to be a viable search engine. But less blatant bias are likely to be tolerated by the market. For example, a search engine could add a small factor to search results from "friendly" companies, and subtract a factor from results from competitors. This type of bias is very difficult to detect but could still have a significant effect on the market. Furthermore, advertising income often provides an incentive to provide poor quality search results. For example, we noticed a major search engine would not return a large airline's homepage when the airline's name was given as a query. It so happened that the airline had placed an expensive ad, linked to the query that was its name. A better search engine would not have required this ad, and possibly resulted in the loss of the revenue from the airline to the search engine. In general, it could be argued from the consumer point of view that the better the search engine is, the fewer advertisements will be needed for the consumer to find what they want. This of course erodes the advertising supported business model of the existing search engines. However, there will always be money from advertisers who want a customer to switch products, or have something that is genuinely new. But we believe the issue of advertising causes enough mixed incentives that it is crucial to have a competitive search engine that is transparent and in the academic realm.

9 Appendix B: Scalability

9. 1 Scalability of Google

We have designed Google to be scalable in the near term to a goal of 100 million web pages. We have just received disk and machines to handle roughly that amount. All of the time consuming parts of the system are parallelize and roughly linear time. These include things like the crawlers, indexers, and sorters. We also think that most of the data structures will deal gracefully with the expansion. However, at 100 million web pages we will be very close up against all sorts of operating system limits in the common operating systems (currently we run on both Solaris and Linux). These include things like addressable memory, number of open file descriptors, network sockets and bandwidth, and

many others. We believe expanding to a lot more than 100 million pages would greatly increase the complexity of our system.

9.2 Scalability of Centralized Indexing Architectures

As the capabilities of computers increase, it becomes possible to index a very large amount of text for a reasonable cost. Of course, other more bandwidth intensive media such as video is likely to become more pervasive. But, because the cost of production of text is low compared to media like video, text is likely to remain very pervasive. Also, it is likely that soon we will have speech recognition that does a reasonable job converting speech into text, expanding the amount of text available. All of this provides amazing possibilities for centralized indexing. Here is an illustrative example. We assume we want to index everything everyone in the US has written for a year. We assume that there are 250 million people in the US and they write an average of 10k per day. That works out to be about 850 terabytes. Also assume that indexing a terabyte can be done now for a reasonable cost. We also assume that the indexing methods used over the text are linear, or nearly linear in their complexity. Given all these assumptions we can compute how long it would take before we could index our 850 terabytes for a reasonable cost assuming certain growth factors. Moore's Law was defined in 1965 as a doubling every 18 months in processor power. It has held remarkably true, not just for processors, but for other important system parameters such as disk as well. If we assume that Moore's law holds for the future, we need only 10 more doublings, or 15 years to reach our goal of indexing everything everyone in the US has written for a year for a price that a small company could afford. Of course, hardware experts are somewhat concerned Moore's Law may not continue to hold for the next 15 years, but there are certainly a lot of interesting centralized applications even if we only get part of the way to our hypothetical example.

Of course a distributed systems like G*l*oss [Gravano 94] or Harvest will often be the most efficient and elegant technical solution for indexing, but it seems difficult to convince the world to use these systems because of the high administration costs of setting up large numbers of installations. Of course, it is quite likely that reducing the administration cost drastically is possible. If that happens, and everyone starts running a distributed indexing system, searching would certainly improve drastically.

Because humans can only type or speak a finite amount, and as computers continue improving, text indexing will scale even better than it does now. Of course there could be an infinite amount of machine generated content, but just indexing huge amounts of human generated content seems tremendously useful. So we are optimistic that our centralized web search engine architecture will improve in its ability to cover the pertinent text information over time and that there is a bright future for search.