RECUPERACIÓN DE INFORMACIÓN EN SISTEMAS DISTRIBUIDOS

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

Eric Sadit Téllez Avila

Dr. Edgar Leonel Chavez González

Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

Mayo 2006

Agradezco a mis padres, que bajo viento y marea siempre me han apoyado, siempre alentandome para superarme. A Lola, por compartirme su vida desde hace dos años. A mis hermanos, inmejorable compañia. A mis amigos que me han acompañado a lo largo de todos estos años. A mis profesores, en especial a los profesores Dr. Juan José Flores Romero y al Dr. Edgar Leonel Chávez Gonzalez.

Resumen

En ésta tesis se diseña e implementa una arquitectura distribuida para realizar Recuperación de Información, funcionando en una amplia gama de Sistemas Operativos (FreeBSD, GNU/Linux, Mac OS X, entre otros). La arquitectura define un Sistema de Recuperación de Información de propósito general y soporta redes distribuidas colaborativas sin servidores centralizados y redes distribuidas con servidores centralizados para redes de alto rendimiento con distribución de carga como punto primordial. La arquitectura está diseñada para operar en ambientes heterogéneos con altas fallas en los nodos.

Para alcanzar estos propósitos, se ha creado una infraestructura de manejo de objetos remotos, llamada "Simple Python Remote Objects" (SPyRO), la cual permite la comunicación entre procesos remotos utilizando objetos. A su vez, también permite la ejecución de métodos sobre objetos seleccionados. SPyRO crea redes de costo mínimo, gracias a su algoritmo de selección de conexiones. Además, SPyRO está integrado dentro de Python, permitiendo el desarrollo rápido y sencillo de aplicaciones de computo distribuidas.

La implementación de la arquitectura, llamada "Simple Python Distributed Indexing" (SPyDI), tiene soporte para código móvil y se ha probado su utilidad en dos especializaciones "A Fully Federated Academic Information Retrieval System" (AFFAIRS) y "Universal Index" (U-Index), aplicaciones para Recuperación de Información Distribuida, con sus propias características y propiedades.

Abstract

In this thesis we design and implement a distributed architecture to perform Information Retrieval. The developed architecture is a general purpose Information Retrieval System that can be used in colaborative distributed network in a central serverless fashion and can use central server to make load distribution in a high performance Information Retrieval network. The architecture was designed to operate in heterogeneus and faulty environment.

In order to achieve our purposes, we create a framework and object request broker, called "Simple Python Remote Objects" (SPyRO). SPyRO allows the communication between remote process using objects. In addition, SPyRO allows the execution of remote methods hosted in remote objects. SPyRO creates minimum cost network connections, due to its algorithm to perform connections. Even more, SPyRO is tightly integrated in the Python language. Allowing fast prototyping and simple development of distributed computing applications.

The architecture implementation was called "Simple Python Distributed Indexing" (SPyDI). SPyDI supports mobility code, and it has been proved its usefulness in two specializations. We call them "A Fully Federated Academic Information Retrieval System" (AFFAIRS) and "Universal Index" (U-Index), each of them are applications to perform Distributed Information Retrieval, and both have its own characteristics and properties.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Lista de Símbolos

D	Documento en representación vectorial.
Q	Consulta en representación vectorial.
F	Marco de trabajo para modelar la representación de los documentos, consultas y sus relaciones.
$R(q_i, d_j)$	Función de ponderación genérica
q_i	i-ésimo término de una consulta
k_i	i-ésimo término de un documento
C	Colección de documentos
d_i	i-ésimo documento de una colección C
w_i^j	Peso del término i en el documento j
$TF \times IDF$	Frecuencia de término múltiplicado por frecuencia inversa de documento
idf_i	Frecuencia inversa de documento del término i
f_i^j	Frecuencia del término i en el documento j
$f_m^j ax$	Frecuencia máxima sobre todas las frecuencias de los términos del documento j
Σ	Vocabulario de caractéres válidos para formar términos

Se agrega la lista de publicaciones

Lista de Publicaciones

"Java Library for Making Graph GUIs", Eric Sadit Téllez, Juan José Flores Romero, Mario Graff Guerrero. Computing, Comunications and Control Technologies (CCCT) 2004. Austin, Texas.

"SPyRO: Simple Python Remote Objects", Eric Sadit Téllez, Edgar Chavez, Juan Contreras-Castillo. *Latin American Web Congress* (LA-WEB'2006). Puebla, Mexico.

"A Universal Full Text Index with Access Control and Annotation Driven Information Retrieval", Edgar Chavez, Eric Sadit Téllez. CIC 2006. Cd. México, México.

"AFFAIRS: A Fully Federated Academic Information Retrieval System", Edgar Chavez, Eric Sadit Téllez, Cuauhtemoc Rivera-Loaiza. 1st International Workshop on Peer to Peer and Service Oriented Hypermedia: Techniques and Systems (HT05-p2p-SOA). No publicado por no poder asistir. En espera de un nuevo congreso.

Capítulo 1

Introducción

En la actualidad, la cantidad de información disponible en bancos de datos como los pertenecientes a grandes instituciones (expedientes médicos, registro de automóviles, habitantes, etc) es inmanejable por *Sistemas de Recuperación de Información* basados en servidores únicos centralizados. Además de los altos costos de los servidores de alto rendimiento, existen límites en las capacidades de un servidor. Estos límites son sobrepasados por configuraciones más baratas de varios servidores pequeños, colaborando entre sí para incrementar las funcionalidades de los servicios y expandiendo las posibilidades de escalamiento en el servicio.

En ambientes donde prescindir de servicios no es una opción, un servidor único representa un punto de fallo, por lo que, es necesario acondicionar los servidores con protecciones especiales para evitar interrupción o falla de los servicios. Las protecciones se realizan en servidores centralizados por medio de hardware costoso para asegurar el suministro de energía eléctrica (UPS, protecciones, múltiples fuentes de energía), conexión permanente a red (múltiples conexiones de red), etcétera.

En ambientes distribuidos sin puntos globales de fallos, el hardware generalmente se hace más barato y es posible mantener mediante software un sistema tolerante a fallos, de alto rendimiento y gran escalamiento. Sin embargo, esto no es una tarea sencilla, y requiere de aplicaciones robustas y especializadas para los ambientes distribuidos.

1.1. Antecedentes

Existe una gran variedad de sistemas capaces de realizar Recuperación de Información, algunos funcionan en ambientes distribuidos, algunos otros de manera centralizada. En la presente sección, se estudiarán algunos de los trabajos más importantes en el área tanto comerciales, como académicos, de código abierto y de código cerrado.

Citeseer [Bollacker98, Bollacker99, Lawrence99a, Lawrence99b] es un sistema enfocado a la recolección autónoma de publicaciones académicas disponibles en la Internet. Citeseer contiene un índice *siempre actualizado* (i.e. bajo modificaciones a la colección el índice se actualiza de manera incremental, sin necesidad de la creación total del índice) desarrollado bajo plataforma SQL. Esto nos lleva a una sobrecarga innecesaria en el procesamiento de la información, debido a que las bases de datos relacionales son herramientas de propósito general, no diseñadas específicamente para el desarrollo de Sistemas de Recuperación de Información. Citeseer opera de manera centralizada y la arquitectura utilizada es difícilmente escalable. Citeseer es incapaz de manejar una gran cantidad de peticiones (e.g. Bajo el uso cotidiano de Citeseer, continuamente el servicio es interrumpido por instantes debido a que llega al límite de sus capacidades).

En 1994, Udi Manber presentó Glimpse [Manber94], una herramienta para el indexamiento y consulta de sistemas de archivos. Glimpse se presentó como una novedad al generar índices muy pequeños, permitiendo únicamente consultas booleanas sobre términos exactos o difusos (expresiones regulares, términos mal escritos, etcétera). Glimpse resuelve consultas de una forma lenta en comparación con "índices invertidos", pero es suficiente para el uso personal. Más tarde, Glimpse evolucionó hacia WebGlimpse [Manber97], un indexador y un *crawler* o *araña web* (proceso capaz de recorrer la web recolectando documentos) capaz de indexar algunos cientos de sitios. En la actualidad ha cambiado a software comercial, y distribuye versiones reducidas de manera gratuita bajo licencias de no comercialización, uso personal o al gobierno de los Estados Unidos de Norte América. WebGlimpse cuenta con implementaciones optimizadas de indexador, *crawler*, y módulo resolución de consultas. Sin embargo, no existe ponderación por medidas de similitud y es un sistema centralizado.

Ht://Dig [Group04] es una máquina de búsqueda con los componentes básicos necesarios para realizar Recuperación de Información. Cuenta con un crawler, indexador e interfaz web de consulta, y su código fuente está bajo la licencia GPL, por lo que no hay restricciones de uso y modificación. Desafortunadamente, Ht://Dig no está diseñado para indexar grandes porciones del Web, así como tampoco grandes colecciones. Ht://Dig soporta consultas booleanas y ordenamiento sobre un gran número de entidades como tiempo y fecha de modificación, título, y puntuación. El método utilizado para la puntuación no está debidamente documentado, pero parece ser alguna medida simple relacionada a la frecuencia de términos en el documento. Esta maquinaria es completamente centralizada y poco escalable para grandes colecciones.

Lucene [Cutting05] es una máquina de indexamiento de texto, de alto rendimiento, escrita completamente en Java [Sun], es gratuito y de código abierto, soporta búsquedas ponderadas, por campos, fechas, con soporte para trabajo distribuido y soporta actualizaciones y búsquedas simultaneas. Sin embargo, no contiene repositorios, crawler o interfaz, ya que consiste únicamente del indexador y acceso por medio de una *Interfaz para la Programación de Aplicaciones* (*API* por sus siglas en Inglés). Desgraciadamente, Java requiere de una gran cantidad de memoria para su ejecución, lo que no es adecuado para *ambientes pervasivos*[1].

Swish-E [eDevelopment Team05] es una máquina de indexamiento y consulta creada para procesar diferentes tipos de documentos, contiene un crawler para la adquisición de datos y es capaz de crear funciones de ponderación sin necesidad de recompilar el programa. Desgraciadamente, Swish-E es costoso en memoria, y el uso de memoria no puede ser manejado, además de no soportar cómputo distribuido.

MG de Bell, Moffat y Zobel [Bell95] es un sistema de indexamiento y consulta utilizado por la biblioteca digital de Nueva Zelanda. Esta especialmente enfocado a la compresión de índices y archivos invertidos. MG no soporta indexamiento incremental de documentos, y por sí mismo, no soporta cómputo distribuido.

Google [Brin98] es un Sistema de Recuperación de Información, probablemente el

[1]Existen versiones de Java para sistemas pervasivos pero los programas generados, a pesar de la filosofía de Java, no son compatibles con los ambientes tradicionales de ejecución de Java y viceversa

más popular e importante del planeta. Google nace como un proyecto académico de dos estudiantes de doctorado, Sergey Brin y Lawrence Page, en la Universidad de Stanford. El proyecto tenía como objetivo probar la efectividad del algoritmo llamado PageRank y de mostrar la anatomía de una máquina de búsqueda para la World Wide Web. Basada en cómputo distribuido para la nivelación de cargas en los nodos, permitiendo cómputo barato y tolerante a fallos. Google, resulto ser altamente eficiente y escalable en comparación de máquinas de búsqueda para el WWW existentes como las desarrolladas Manber [Manber94], MG, Altavista o Yahoo!. Su principal característica es permitir que equipo de cómputo barato, basado en Linux y otros UNIX, crezca tanto como sea necesario (con algunos límites detectados [Ghemawat03]) para alcanzar los requerimientos del problema de indexamiento en la WWW. Google en sí mismo, es una máquina optimizada para el balance de carga, con alta replicación entre sus nodos, de modo que en su granja de más de 15000 computadoras es posible responder a miles de peticiones por segundo [Barroso03]. Si una computadora presenta fallas, puede ser remplazada por otra sin ningún problema, gracias a la replicación de los datos. Contiene un repositorio especialmente creado para la tarea de indexamiento en la WWW optimizado para acceso aleatorios y envíos de flujos. En Google, borrar un documento requiere mucho trabajo, y debido a que en la Web se actualiza cerca del 40 % del contenido total por semana [Baeza-Yates99], borrar es un gran problema en Google. El índice utilizado no soporta actualizaciones, en cambio el índice completo es reconstruido cada cierto tiempo. Google, a pesar de su tolerancia a fallas, replicación de contenido e indexamiento distribuido contiene puntos críticos que no se han podido resolver. Por ejemplo, existen manejadores globales de información, que son computadoras altamente protegidas por hardware (contrario a la filosofía general de Google) que significan puntos de falla. Estos puntos globales son índices de documentos, controlador de replicación, controladores de cosechadores, ruteadores de peticiones, etc. Google no es un sistema distribuido donde cada uno de los nodos colabore de forma autónoma para conseguir un beneficio global, es más bien, una máquina de búsqueda altamente optimizada, diseñada para soportar fallas en el equipo de cómputo general, pero es un sistema que debe funcionar como una unidad y es incapaz de ser segmentado y seguir funcionando [Barroso03]. Google contiene a la fecha (Agosto 2005) $8,168,684,336$ páginas web indexadas, así como una serie de servicios

basados en su misma infraestructura.

WebBase [Cho04, Melnik01] es la rama académica de Google, mantenida por los académicos de Stanford desde la comercialización de Google a finales de los noventas. WebBase comienza a partir del código abierto de Google y contiene principalmente, los mismos componentes iniciales de Google: un *Crawler*, repositorio de páginas web recolectadas, un indexador para texto e hiper-vínculos y distribución de contenido. Es utilizado principalmente como fuente para realizar investigaciones sobre lingüistica y la topología de la WWW. WebBase, al igual que Google ha evolucionado para estar altamente optimizado, y bajo la misma anatomía de Google, es decir, utilizando balance de carga, más que colaboración entre nodos.

Omnipaper [Technologies05a, Technologies05c, Technologies05b] es una capa de enlace y navegación multilenguaje diseñada para operar en recursos distribuidos, orientado a noticias y específicamente a diarios de la unión europea. El proyecto Omnipaper está siendo desarrollado por un consorcio formado por importantes Universidades europeas e iniciativa privada. Actualmente Omnipaper cuenta con un prototipo funcional pero incompleto de las especificaciones documentadas.

Omnipaper utiliza técnicas de Recuperación de Información e Inteligencia Artificial para alcanzar estructuras de vocabulario multilenguaje, ontologías que dan sentido a la información y aprendizaje de la navegación y comportamiento de los usuarios. El diseño es centrado en el usuario y utiliza tecnologías que actualmente definen el estado del arte como SOAP, RDF y Mapas Tópicos para proporcionar un sistema de Recuperación de Información con navegación flexible, filtros de información, y soporte multilenguaje. El soporte para varios lenguajes es provisto en la interfaz de usuario y además las consultas pueden ser hechas en un lenguaje y los resultados pueden contener documentos relevantes en cualquier lenguaje soportado. La presentación de los resultados puede ser en un formato basado en texto o en gráficas SVG que muestran visualmente los elementos y relaciones (ie. deducidas a partir de las ontologías).

La recolección de los datos utiliza el método *pull* (recolección de los datos), que es utilizada por robots de páginas web o arañas web. El método *push* (envío por parte de los usuarios o sistemas recolectados) es considerado, pero no implementado. El proyecto

Omnipaper es cerrado en código fuente, y únicamente se publican avances, características y especificaciones. Omnipaper, al igual que Google y WebBase, es distribuido pero no colaborativo entre sus nodos.

Son pocas las herramientas estables de código libre que soporten el escalamiento necesario para indexar y consultar largas colecciones de documentos. Muchas menos son lo suficientemente robustas para soportar fallas de hardware, red, fallas parciales del sistema, corrupción, etcétera. En aplicaciones más elaboradas como las comerciales, la principal preocupación está dirigida hacia el balance de carga, olvidando la colaboración y la tolerancia hacia la segmentación de redes.

Las redes colaborativas, sin puntos centralizados de fallos, son llamadas *P2P peer-to-peer*, un excelente estudio sobre las redes P2P la da [Androutsellis-Theotokis04]. Las redes P2P no deben ser confundidas con el cómputo Grid [Frattolillo05, Casanova02, Snavely03, Foster98]. Pues a diferencia de Grid, P2P tiene como principal objetivo el escalamiento y la tolerancia a fallos, mientras que Grid se enfoca en eficiencia, estandarización y coordinación entre los componentes de la red. En un futuro, se piensa que tanto Grid como P2P se fusionarán aprovechando las características de ambas tecnologías [Androutsellis-Theotokis04].

Las redes P2P son utilizadas para *distribución de contenido*, *cómputo distribuido*, *prestación de servicios de Internet*, y *bases de datos* [Androutsellis-Theotokis04]. Generalmente las redes P2P son conocidas en aplicaciones relacionadas con la distribución de contenido en aplicaciones como Napster, Gnutella [Ripeanu01, Kan01], Kazaa [Kazaa05], Kademlia [Maymounkov02], Freenet [Clarke02, Clarke01] o Bittorrent [Cohen03, Qiu04].

Sin embargo, la distribución de contenido solo es una de tantas aplicaciones posibles. Es posible utilizar arquitecturas P2P para la recuperación eficiente de información, como lo han demostrado Klampanos [Klampanos04], Bawa [Bawa03], Baquero [Baquero03] y Crainiceanu [Crainiceanu04].

Iraklis Klampanos y Joemon Jose presentan en el 2004 una arquitectura novedosa de indexamiento y consulta [Klampanos04]. La arquitectura está basada en redes P2P. Cada nodo en la red es capaz de identificarse mediante un vector, por lo que el ruteo de consultas es realizado mediante la elección correcta de nodos para realizar consultas. Una vez que la colección de nodos es elegida, se procede a realizar la consulta en cada nodo, uniendo

los resultados y ponderándolos mediante el modelo vectorial. Este tipo de estrategia para escoger la colección de nodos importantes para una consulta es un área importante de investigación y puede verse mucha información al respecto en [Khoussainov04, Sun04, Si02, Rasolofo01, Suciu02, Kossmann00, Bawa03].

SETS [Bawa03] es un sistema destinado a mejorar la calidad de los resultados en las consultas, basado en la teoría de redes sociales. SETS no es una máquina indexadora de documentos externos, es una máquina de resolución de consultas e indexadora de documentos de usuario. SETS agrupa nodos con tópicos parecidos de manera que los nodos que contienen tópicos parecidos estén cerca. Se crean centroídes de tópicos, que son los responsables de decidir si una consulta es relevante al tópico, y en caso de serlo, el centroíde es responsable de propagar la consulta. SETS protege contra la falla de centroídes creando varios centroídes.

Carlos Baquero y Nuno Lopes presentan un estudio acerca de las perspectivas a futuro del indexamiento de contenido utilizando redes P2P [Baquero03]. Enfocado en la utilización de tablas hash distribuidas (*DHT* por sus siglas en inglés), la eficiencia de ruteo y segmentación de llaves. Se demuestra que la segmentación del índice no es viable para un sistema P2P. Esto ya es mostrado para redes distribuidas de alto rendimiento como WebBase [Melnik01].

Crainiceanu et Al presenta un desarrollo a futuro llamado Framework para indexamiento sobre redes P2P [Crainiceanu04]. Éste desarrollo describe la propuesta sobre los requerimientos y características del Framework. El Framework se enfoca en la operación de ruteo, almacenamiento y replicación de datos a través de la red P2P.

Las arquitecturas P2P presentadas hasta el momento, se limitan al estudio del procesamiento de las consultas, así como la topología necesaria para realizar consultas eficientes, sin importar la adquisición de datos.

1.2. Solución Propuesta y Objetivos

Los sistemas de Recuperación de Información presentados son variados, algunos capaces de escalar para permitir el manejo de grandes colecciones de documentos, mientras

que otros únicamente son capaces de realizar el trabajo para colecciones pequeñas.

Por otra parte, los sistemas distribuidos de alto rendimiento mostrados olvidan la colaboración entre nodos de la red, enfocándose en el escalamiento y eficiencia.

Los ambientes de Recuperación de Información basados en P2P actuales están formados de módulos de indexamiento y consulta. La principal preocupación de los sistemas P2P de Recuperación de Información es la realización de las consultas y el ruteo de contenido y peticiones. En P2P aún no existe una arquitectura básica a seguir para la creación de un sistema de Recuperación de Información, con todos los módulos requeridos para que ésta se lleve a cabo.

En cuanto a funciones de similitud, algunos de ellos (especialmente los basados en redes P2P) usan métodos booleanos para la resolución de consultas, estos son incapaces de ponderar las relevancias de los documentos contra la consulta realizada.

Ésta tesis consiste en el diseño e implementación de una arquitectura para la Recuperación de Información en Ambientes Distribuidos, que soporte tanto conexiones P2P, como arreglos para distribución de carga con servidores centralizados. La arquitectura debe ser robusta, escalable, modular y fácilmente extensible para llenar el vacío de sistemas de recuperación de información libres y no libres.

Adicionalmente, una serie de objetivos se persiguen:

- Creación de un sistema de recuperación distribuido basado en la arquitectura de ésta tesis que contenga un indexador de documentos, una máquina para resolución de consultas, y una interfaz mediadora de objetos remotos que permita la unión de los módulos de manera transparente de forma que permita el fácil y rápido desarrollo de aplicaciones y módulos.

- Que el mediador de objetos sea capaz de permitir el uso de Servicios Web para ampliar el rango de posibles aplicaciones y clientes, adicionalmente debe permitir código móvil [Fuggetta98].

- Que los sistemas de recuperación de información generados sean capaces de operar con diferentes tipos de adquisición de datos, tales como los métodos recolección (*pull*) o métodos de alimentación guiados por los clientes (método *push*).

- Las aplicaciones desarrolladas deben poder operar en redes heterogeneas, tanto en *hardware* como en *software*.
- Que sea posible crear aplicaciones cuyos nodos sean de índole general o de propósito específico, habilitando su uso para redes P2P y de distribución de carga.
- Creación de módulos distribuibles, es decir, cada módulo por si mismo debe soportar encontrarse en diferentes computadoras.
- Que los sistemas generados sean autónomos y de bajas dependencias para facilitar su utilización por la comunidad.
- Que el trabajo desarrollado sirva como punto de partida para futuros trabajos en las áreas de Recuperación de Información y P2P.

La implementación de la arquitectura es código libre[2], por tanto, cualquiera que desee probarlo, usarlo, o estudiarlo es libre de hacerlo teniendo acceso al código fuente de la implementación.

1.3. Descripción del Contenido de la Tesis

El contenido de la Tesis se distribuye de la siguiente manera: El presente Capítulo describe el objetivo de la tesis y muestra un estudio de las herramientas actuales para Recuperación de Información Distribuida. El Capítulo 2 está dedicado a dar una introducción al área de Recuperación de Información, así como al uso de ambientes distribuidos en la Recuperación de Información. En el Capítulo 3 se estudian los objetos remotos utilizados para enlazar programas y módulos a través de una red heterogénea. El módulo de adquisición de datos y el repositorio de los documentos se estudia en el Capítulo 4. A lo largo del Capítulo 5 se muestra la arquitectura y la implementación del módulo de indexamiento y consulta. En el Capítulo 6 se documenta un conjunto de aplicaciones desarrolladas directamente utilizando las tecnologías y arquitecturas creadas como fruto del desarrollo de ésta tesis. Por último, en el Capítulo 7 se comenta el trabajo a futuro y las conclusiones generadas por éste trabajo.

[2]La implementación está disponible en http://www.spyron.org

Capítulo 2

Recuperación de Información

La Recuperación de Información (*RI*) trata de la representación, almacenamiento, organización y acceso a entidades de información. La representación y la organización de entidades de información permiten al usuario el acceso sencillo a la información de interés.

Desafortunadamente, la caracterización de la *necesidad de información de un usuario* no es un problema simple. El problema viene en dos sentidos: usuario y sistema. Los usuarios deben interpretar sus necesidades y dar *consultas* preparadas para los Sistemas de RI (*SRI*). Los SRI, por el contrario, deben interpretar las necesidades de los usuarios de la manera más acertada para proceder a la búsqueda de la información relevante para cada consulta en el banco de datos. En el mejor de los casos un usuario daría la consulta en lenguaje natural y el SRI sería capaz de interpretarla satisfactoriamente y de encontrar la mejor información posible para satisfacer la consulta.

Desde el punto de vista computacional, el problema de la recuperación de documentos puede ser visto desde dos ángulos: Recuperación de Información y Recuperación de Datos. La recuperación de datos consiste en determinar el subconjunto de documentos que contienen las *palabras clave* dadas en una consulta, la cual frecuentemente, no es suficiente para satisfacer las necesidades de información del usuario. En cambio, en un SRI es necesario recuperar *información*, más que recuperar datos.

En un sistema de recuperación de datos, un sólo objeto erróneo entre miles de objetos recuperados significa un error total. Por el contrario en la Recuperación de Infor-

mación se admiten pequeños errores e inexactitudes en las respuestas. Ésta tolerancia a fallos es debida a que generalmente la RI maneja texto en lenguaje natural, el cual no siempre está bien estructurado y puede ser semánticamente ambiguo. En el otro lado, un sistema de recuperación de datos (como una base de datos relacional) trata con datos que tienen una estructura definida y una semántica estricta.

Para realizar recuperación de información es necesario *interpretar* la consulta para poder determinar la subcolección de documentos que son *información relevante* al usuario. Los documentos pueden o no pueden contener las mismas palabras, así como es posible dar un *grado de relevancia* de acuerdo a la consulta dada. La interpretación del contenido de un documento consiste en extraer información sintáctica, semántica, y estadística del texto del documento y usar ésta información para intentar resolver el problema de información de los usuarios. La dificultad no reside únicamente en conocer como extraer la información, si no también en usarla de la mejor manera para decidir la *relevancia*. La noción de *relevancia* es el centro de la recuperación de información. De hecho, la meta principal de un SRI es recuperar todos los documentos que son relevantes a una consulta de usuario con el menor número de documentos no relevantes.

El presente Capítulo está dedicado a los fundamentos de la Recuperación de Información[1] En la sección 2.1 se presentan los conceptos básicos. En la sección 2.2 se muestra la forma en la cual se modelan los documentos dentro de un Sistema de Recuperación de Información. La sección 2.3 describe el proceso de Recuperación de Información, haciendo énfasis en el indexamiento y la consulta. En la sección 2.4 se da una perspectiva básica de los modelos de recuperación de información utilizados. Las bases para comprender la Recuperación de Información Distribuida se presenta en la sección 2.5. Por último, la sección 2.6 resume el Capítulo.

[1] Algunos de los temas tratados en la tesis no son mencionados en ésta introducción (objetos remotos, adquisición de datos, redes P2P, etcétera), en cambio, son explicados con detalle en cada uno de los respectivos Capítulos.

2.1. Conceptos Básicos

La recuperación efectiva y la relevancia de información es afectada directamente tanto por las *tareas de usuario* como por la *vista lógica de los documentos* adaptada al sistema de recuperación. La tarea del usuario es trasladar la información requerida a un lenguaje permitido por el sistema. Esto es generalmente un conjunto de palabras que semánticamente expresen la información requerida.

Una tarea posterior es escoger que documentos son relevantes de los entregados por el sistema. Ésta es una tarea donde interviene en gran parte la interfaz proporcionada por el sistema. Para la Web, la interfaz presentada es una página Web con ligas a documentos. Cada documento mostrado es potencialmente relevante para el usuario, pero en última instancia el usuario es quien decide si un documento es relevante. En esta etapa el usuario *navega* entre los documentos entregados. Durante ésta etapa, la retroalimentación para encontrar los requerimientos de información para el usuario es fundamental.

2.2. Vista Lógica de los Documentos

Por razones históricas, los documentos en una colección son frecuentemente representados a través de un conjunto de términos o palabras clave, también muchas veces llamados términos índice. Cada término puede ser extraído directamente del documento o especificado por un humano (e.g. en textos científicos es común que un asesor humano clasifique un documento como perteneciente a una determinada área).

Las computadoras modernas hacen posible representar un documento como el conjunto completo de sus términos. En éste caso, es posible decir que el sistema de recuperación adopta la vista lógica de *texto completo*. Sin embargo, aún para las computadoras modernas el manejo de colecciones largas de documentos es un trabajo excesivo. Por tanto, es muchas veces necesario reducir el tamaño de los conjuntos de términos necesarios para describir los documentos. Existen varias técnicas que realizan ésta reducción, son llamadas operaciones de texto o *transformaciones* [Baeza-Yates99]. Las más comunes las siguientes:

- **Borrado de palabras basura o *stopwords*.** Las palabras comunes para toda la

colección de documentos pueden ser removidas, ya que evidentemente, ellas no ayudan a localizar un documento. En ésta categoría entran comúnmente los adjetivos, adverbios, preposiciones, etcétera.

- **Minimización de palabras o *stemming*.** Se realiza reduciendo un conjunto de palabras a su raíz común.
- **Identificación de sustantivos.** Se descartan las entidades de poco peso semántico utilizando únicamente los sustantivos, que son las entidades con más importancia semántica (desde el punto de vista recuperación de información, y para determinados textos).
- **Normalización.** Consiste en estandarizar el alfabeto usado para formar términos, eliminando las posibles variaciones una misma palabra (i.e. convertir todos los caracteres a mayúsculas o minúsculas, remover acentos, diéresis, etcétera).

Las operaciones sobre texto reducen la complejidad de la representación de los documentos y permiten mover de una representación de documentos (en texto completo) a una representación de conjunto términos de indexamiento.

El texto completo es la manera cabal de ver un documento, pero su uso implica altos costos computacionales. Otro enfoque es la asignación de una categorías, donde número pequeño de categorías (creada por especialistas humanos) provee de una vista lógica más concisa de un documento, pero su uso implica una recuperación de baja calidad. En la recuperación de texto estructurado es útil el reconocimiento de partes esenciales en documentos como el título, secciones, capítulos, etcétera. En algunas colecciones, como la Web, es recomendable utilizar la representación completa de los documentos, utilizando reconocimiento de ciertas etiquetas de los documento de hipertexto.

2.3. El proceso de Recuperación

El primer paso en el proceso de RI es la selección de la representación lógica de los documentos (texto completo o definir las transformaciones necesarias). Una vez que la vista lógica de los documentos es definida, con la ayuda de estructuras especializadas se

crea un índice del texto. Un índice es una estructura de datos crítica debido a que permite búsquedas rápidas a lo largo de grandes volúmenes de datos. Los recursos (tiempo y espacio de almacenamiento) gastados en definir el índice son amortizados por la gran cantidad de consultas en el sistema.

Una vez que los documentos en la base de datos están indexados, el proceso de recuperación puede ser iniciado. El usuario debe especificar sus requerimientos de información, los cuales son analizados y transformados mediante el mismo procedimiento que los documentos. La consulta es procesada y se recuperan los documentos seleccionados. Los procesos de consulta son rápidos debido a las estructuras de índice previamente creadas.

Antes de que los documentos recuperados sean devueltos al usuario que realiza la consulta, estos deben ser ponderados por importancia relativa a la consulta. En éste punto un ciclo de retroalimentación puede ser realizada para mejorar los resultados. En tal ciclo, es posible mejorar la calidad de los resultados.

2.3.1. Indexamiento

Existen varios tipos de técnicas de indexamiento, las más populares son *archivos invertidos y arreglos de sufijos*. En éste trabajo se utilizan archivos invertidos, debido a su simplicidad de creación y mantenimiento. Sin embargo, los arreglos de sufijos se desempeñan mejor espacialmente y para consultas de frases, pero su construcción requiere de mayor complejidad y son inherentemente estáticos. A continuación, se explicarán los archivos invertidos (la cual es la forma más popular de indexamiento en Internet y usada por las máquinas de búsqueda modernas), una excelente referencia para arreglos de sufijos es [Manber90, Baeza-Yates99].

Un *archivo invertido* o *índice invertido* es un mecanismo orientado a palabras usado para indexamiento de colecciones de documentos. Es utilizado para acelerar las tareas de consulta. La estructura de los archivos invertidos está compuesta por dos elementos: El vocabulario y las listas de ocurrencia. El vocabulario es el conjunto de todos los términos en el texto. Las listas de ocurrencia indican los documentos que contienen el término, junto con información necesaria para ponderar el peso del témino en el documento. Para cada palabra, opcionalmente se almacena una lista de las posiciones donde el término ocurre.

La unión de todas las listas de ocurrencias es el conjunto de *ocurrencias*. Estas posiciones pueden ser referidas a palabras o carácteres. Las posiciones de las palabras facilitan el acceso a las posiciones de las ocurrencias en el texto.

En colecciones de documentos las posiciones son remplazadas por los documentos donde aparecen. Las posiciones de los términos en cada uno de los documentos también pueden ser especificadas, permitiendo busquedas por frases o por proximidad de palabras, otorgando una significado semántico a la consulta. Adicionalmente, es necesario añadir información en las listas de ocurrencias que nos ayude a ponderar la importancía de un término en el documento.

Construcción

La construcción de los archivos invertidos es un proceso muy simple. Aunque para la realización eficiente de estos es muchas veces un problema serio a enfrentar. Una de las maneras más simples es realizar el análisis del texto de manera secuencial. Mientras se analiza el texto se reconocen palabras, las cuales son añadidas al vocabulario, y a su vez se reconocen las posiciones de las palabras para crear la lista de ocurrencias.

Para textos de grandes dimensiones, no es posible crear el índice invertido utilizando la memoria primaria, ya que la lista de correspondencias es $O(n)$ espacialmente al tamaño del texto. Por tanto, es necesario realizar el indexamiento segmentando los índices, que consiste en realizar el indexamiento de segmentos de la colección y realizar uniones de varios índices parciales hasta obtener un índice completo. Una versión paralela del algoritmo puede ser fácilmente deducida, por lo que es posible auxiliarse de una o más computadoras con uno o más procesadores para acelerar el proceso de construcción de índices.

El tiempo total para generar los índices parciales es de $O(n)$ al igual que el tiempo usado para hacerlo en memoria. El número de índices parciales es de $O(n/M)$. Cada nivel de fusión de índices parciales realiza un proceso lineal sobre el índice entero (no importa como está partido en índices parciales) y entonces su costo final es de $O(n)$. Para unir $O(n/M)$ índices parciales, $\log_2(n/M)$ niveles de fusión son necesarios. Por lo tanto, el algoritmo es $O(n \log(n/M))$ [Baeza-Yates99].

Si varios índices son unidos a la vez, la velocidad de unión aumenta debido a los

menos pasos de unión y menos índices temporales.

2.3.2. Consultas

El algoritmo de búsqueda en índices invertidos sigue tres pasos generales (algunos de ellos ausentes para consultas específicas).

- **Búsqueda del vocabulario.** Las palabras y los patrones presentados en las consultas son aislados y buscados en el vocabulario. Debe tenerse presente que las frases y las consultas con proximidad son divididas en palabras simples.
- **Recuperación de ocurrencias.** La lista de ocurrencias para cada palabra encontrada es recuperada.
- **Manipulación de ocurrencias.** Las ocurrencias son procesadas para resolver frases, proximidad, u operaciones booleanas.

El proceso de buscar en un archivo invertido siempre comienza por el vocabulario. Debido a esto, es una buena idea mantener en un archivo separado el vocabulario. Es posible que éste archivo pueda ser almacenado en memoria para colecciones de texto de tamaño moderado. Para colecciones grandes y diversas pueden ser utilizadas otras técnicas, tales como distribución de vocabularios o almacenamiento secundario. En el presente trabajo se hace uso del almacenamiento secundario apoyado en *cachés* especializadas para aumentar el rendimiento del sistema en colecciones largas.

Las consultas se realizan apoyandose en estructuras de datos especializadas para la búsqueda e indexamiento de texto: como tablas *hash*, *tries*, o *B-trees* [Cormen01, Johnson93, Baeza-Yates99]. Las tablas hash tienen un costo de búsqueda de $O(1)$ en caso promedio, pero un peor caso $O(t^2)$ donde t es el número de términos en el índice o vocabulario. Los tries dan un tiempo de búsqueda de $O(m)$ donde m es el tamaño de la búsqueda, independiente del tamaño del vocabulario, del tamaño de la colección y tamaño de los documentos. Los B-trees son $O(\log(t))$. Con B-trees y tries es posible realizar búsquedas por rango, pero con tablas hash no.

2.4. Modelos de Recuperación de Información

El problema más importante de la recuperación de información es el concepto de predecir cuales documentos son relevantes y cuales no lo son. Tal decisión es usualmente dependiente de un algoritmo de ponderación que trata de establecer un orden en los documentos recuperados. Los documentos que aparecen en el tope de éste ordenamiento tienen la mayor probabilidad de ser relevantes. Entonces, los algoritmos de ponderación son el corazón de los SRI.

Un algoritmo de ponderación opera de acuerdo a premisas básica con respecto a la noción de relevancia en los documentos. Conjuntos distintos de premisas (con respecto a la relevancia de los documentos) llevan a distintos modelos de recuperación de información.

2.4.1. Caracterización Formal de los Modelos de Recuperación de Información

Definición 2.4.1 *Un modelo de recuperación de información es una tupla de cuatro elementos* $(D, Q, F, R(q_i, d_j))$ *donde:* D *es un conjunto de vistas lógicas (representaciones) para los documentos en la colección.* Q *es un conjunto de vistas lógicas de las necesidades de información del usuario. Tal representación es llamada* consulta. F *es un marco de trabajo para modelar la representación de los documentos, consultas y sus relaciones.* $R(q_i, d_j)$ *es una función de ponderación asociada a un número real con la consulta* $q_i \in Q$ *y la representación del documento* $d_j \in D$*. La función de ponderación define un orden entre los documentos con respecto a la consulta* q_i*.*

Por comodidad, en lo sucesivo no se instanciarán de manera explícita los componentes $D, Q, F, R(q_i, d_j)$ de los modelos utilizados. Estos componentes son claros y fácilmente inferibles en lo sucesivo para evaluar el potencial de un término como tal.

Conceptos Básicos de los Modelos

Definición 2.4.2 *Sea* t *el número de términos índice en el sistema y* k_i *sea el término índice genérico.* $K = \{k_1, k_2, \cdots, k_t\}$ *es el conjunto de todos los términos índice. Un peso* $w_i^j > 0$ *es asociado con cada término índice* k_i *de un documento* d_j*. Para un término índice*

que no aparece en el texto del documento, $w_{i,j} = 0$. *Con el documento* d_j *está asociado un vector de términos índice* $\vec{d_j}$ *representado por* $\vec{d_j} = (w_1^j, w_2^j, \cdots, w_t^j)$. *Además, sea* g_i *una función que regresa el peso asociado con el término índice* k_i *en cualquier vector t-dimensional, esto es,* $g_i(\vec{d_j}) = w_i^j$.

Dado un conjunto de términos índice es posible notar que los términos tienen diferentes importancias para describir el contenido de los documentos. Decidir la importancia de los términos para describir el contenido de los documentos es una tarea complicada. A pesar de ésta dificultad, hay propiedades de los términos índice que son fácilmente medidas y resultan de gran utilidad. Por ejemplo, considerese una colección de cientos de miles o millones de documentos. Una palabra que aparece en cada uno de los documentos de la colección es completamente inútil como un término índice debido a que no dice nada importante acerca de un documento. Por lo contrario, una palabra que aparece en solo cuatro documentos es realmente útil, debido a que la palabra estrecha considerablemente el espacio de documentos que pueden ser utiles para los usuarios.

Es claro que cada término tiene una importancia distinta, dependiendo del contenido de cada documento. Éste efecto es capturado a través de la asignación de pesos numéricos para cada término índice del documento.

Sea k_i un término índice, d_j un documento y w_i^j un peso asociado con el par (k_i, d_j). El peso cuantifica la importancia de los términos índice para describir el contenido semántico del documento.

A lo largo de ésta tesis, se asume que los pesos de los términos son independientes, para más información sobre modelos que consideran la dependencia entre términos ver [Baeza-Yates99, Turtle92, Hiemstra00]. La simplificación de términos independientes se debe a la inherente complejidad de los métodos que toman ventaja de la correlación entre documentos, no apta para aplicaciones prácticas. Además, ningún modelo con soporte para términos dependientes ha demostrado tener demasiadas ventajas en colecciones con tópicos generales, y por el contrario, muchos de los documentos seleccionados debidos a los términos dependientes pueden elevar el número de documentos no relevantes en las respuestas [Baeza-Yates99].

Modelo Booleano

El modelo booleano es el más simple de los modelos de recuperación. Este modelo está basado en la Teoría de Conjuntos y Álgebra Booleana. Debido a que el concepto de conjunto es intuitivo, el modelo booleano provee de una plataforma de trabajo que es fácilmente comprendida por el usuario común de un sistema de RI. Además, las consultas son especificadas como expresiones booleanas, las cuales tienen una semántica precisa.

El modelo booleano sufre de varios problemas. Los criterios de decisión son binarios, es decir, un término es o no es relevante. Además aunque una expresión booleana tiene una semántica precisa, muchas veces no es fácil transformar una expresión de lenguaje natural a una semántica booleana. Desde estos puntos de vista, el modelo booleano es en realidad una recuperación de datos.

El modelo booleano considera los términos como presentes o ausentes en un documento. Como resultado, los pesos de los términos se asumen binarios, i.e. $w_i^j \in \{0,1\}$. Una consulta q está compuesta de términos ligados por tres conectivas *no, y, o* (NOT, AND y OR, respectivamente).

Definición 2.4.3 *Para el modelo booleano, los pesos de los términos índices son binarios i.e.* $w_i^j \in \{0,1\}$. *Una consulta* q *es un expresión booleana típica. Sea* $q_{\vec{dnf}}$ *la forma normal disyuntiva de la consulta* q. *Además, sea* $\vec{q_{cc}}$ *uno de los componentes de* $q_{\vec{dnf}}$. *La similitud de un documento* d_j *hacia una consulta* q *está definida como:*

$$sim(d_j, q) = 1 \; si \; \exists \vec{q_{cc}} | (\vec{q_{cc}} \in q_{dnf}) \wedge (\forall_{k_i}, g_i(\vec{d_j}) = g_i(\vec{(q_{cc})})) \quad (2.1)$$

$$sim(d_j, q) = 0 \; de \; otra \; forma \quad (2.2)$$

Si $sim(d_j, q) = 1$ *entonces el modelo booleano predice que el documento* d_j *es relevante para la consulta* q. *De otra forma, la predicción es que el documento es no relevante.*

El modelo booleano predice para cada documento si es relevante o no relevante. No hay noción de correspondencia aproximada a las condiciones de consulta. Éste enfoque nos lleva a un Sistema de Recuperación de Datos. Sin embargo, su simplicidad y su base

teórica sólida hacen del modelo booleano una buena opción para el desarrollo rápido de un SRI.

Modelo Vectorial

El modelo vectorial reconoce que el uso de pesos binarios es muy limitante y propone un marco de trabajo en el cual es posible una correspondencia parcial. Esto se logra asignando pesos no binarios a los términos índice en consultas y en documentos. Estos pesos son usados para computar el grado de similitud entre cada documento almacenado en el sistema y la consulta del usuario. Ordenando los documentos recuperados en orden decreciente a la similitud, el modelo vectorial toma en cuenta los documentos para los cuales la consulta solo corresponde parcialmente. El efecto resultante es ponderar los documentos respuesta para obtener respuestas más precisas y ordenadas (en el sentido de que mejores correspondencias para las necesidades de información) que los documentos recuperados por el modelo booleano.

Definición 2.4.4 *Para el modelo vectorial, el peso w_i^j asociado con el par (k_i, d) es positivo y no binario. Más aún, los términos en la consulta son también ponderados. Sea w_i^q un peso asociado al par (k_i, q), donde $w_i^q \geq 0$. Entonces el vector consulta $\vec{q}$ está definido como $\vec{q} = (w_1^q, w_2^q, \cdots, w_3^q)$ donde t es el número total de términos en el sistema. El vector de un documento $\vec{d_j}$ está representado por $\vec{d_j} = (w_i^j, w_2^j, \cdots, w_t^j)$.*

Así, un documento d_j y una consulta q están representadas como vectores t-dimensionales. El modelo vectorial propone evaluar el grado de similitud de un documento d_j con la consulta q como la correlación entre los vectores $\vec{d_j}$ y $\vec{q}$. Ésta correlación está cuantificada, por ejemplo, por el coseno del ángulo θ entre dos vectores:

$$sim(d_j, q) = \cos\theta \tag{2.3}$$

$$= \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}||\vec{q}|} \tag{2.4}$$

$$= \frac{\sum_{i=1}^{t} w_i^j \times w_i^q}{\sqrt{\sum_{i=1}^{t} (w_i^j)^2} \times \sqrt{\sum_{i=1}^{t} (w_i^q)^2}} \tag{2.5}$$

Donde $|\vec{d_j}|$ y $|\vec{q}|$ son la norma del documento y los vectores consulta, respectivamente. El factor $|\vec{q}|$ no afecta la ponderación u orden de los documentos debido a que es el mismo para todos los documentos. El factor $|\vec{d_j}|$ permite la normalización en el espacio de los documentos.

Debido a que $w_i^j \geq 0$ y $w_i^q \geq 0$, $sim(d_j, q)$ varia de 0 a 1. Entonces, en lugar de tratar de predecir si un documento es relevante o no, el modelo vectorial pondera los documentos de acuerdo a un grado de *similitud* a la consulta.

Definición 2.4.5 *Sea N el número total de documentos en el sistema y n_i el número de documentos en los cuales un término k_i aparece. Sea F_i^j la frecuencia de un término k_i en el documento d_j (i.e. el número de veces que el término k_i es mencionado en el texto de un documento d_j). La frecuencia normalizada f_i^j del término k_i en un documento d_j está dada por la ecuación 2.6.*

$$f_i^j = \frac{F_i^j}{f_{max}^j} \tag{2.6}$$

Donde f_{max}^j es calculada como la frecuencia máxima sobre todos los términos mencionados en el documento d_j. Si el término k_i no aparece en el documento d_j entonces $f_i^j = 0$. Además, sea idf_i, la frecuencia inversa del documento para k_i, la cual está dada por:

$$idf_i = \log \frac{N}{n_i} \tag{2.7}$$

Algunos de los esquemas de recuperación modernos más populares y poderosos (i.e. PageRank [Brin98], CiteRank [Bollacker98]) usan pesos basados en la ecuación 2.8. Estos esquemas de ponderación de términos son nombrados $TF \times IDF$.

$$w_i^j = f_i^j \times \log \frac{N}{n_i} \tag{2.8}$$

Las principales ventajas del modelo vectorial son:

- Los esquemas de ponderación de términos mejoran el desempeño de recuperación.
- La estrategia de correspondencia parcial permite recuperación de documentos para condiciones de consulta aproximada.
- La ponderación por medio de la fórmula del coseno ordena los documentos de acuerdo al grado de similitud a la consulta.

Modelo Probabilístico

El modelo probabilístico intenta envolver el problema de Recuperación de Información dentro de un marco de trabajo probabilístico. La idea fundamental es la siguiente: dada una consulta por parte de un usuario, existe un conjunto de documentos que contienen exactamente los documentos relevantes y no otros. Éste conjunto de documentos será llamado el conjunto de *respuesta ideal*. Por tanto, el proceso de consulta puede ser visto como el proceso de especificar las propiedades de un conjunto respuesta ideal, el problema es determinar cuales son éstas propiedades. El proceso se inicia con una estimación inicial basada en los términos índice. Después comienza un proceso de retroalimentación por parte del usuario, donde éste debe escoger que documentos son realmente relevantes y cuales no lo son. Con la ayuda de ésta selección, es posible redefinir la descripción de documento relevante. Éste es un proceso iterativo.

El principio probabilístico en el que se basa el modelo es el siguiente: Dada un consulta q por parte del usuario, el modelo probabilístico trata de estimar la probabilidad de que un usuario encuentre relevante al documento d_j. El modelo asume que esta probabilidad de relevancia depende únicamente de la consulta y de los documentos representados. Más aún, el modelo asume que hay un subconjunto de todos los documentos que el usuario prefiere como un conjunto respuesta para la consulta q. Tal conjunto *ideal* de respuesta es llamado R y debe maximizar la probabilidad de relevancia para el usuario. Se predice que los documentos en el conjunto R son relevantes para la consulta. Los documentos que no se encuentren en R son irrelevantes por predicción.

Definición 2.4.6 *Para el modelo probabilístico, los pesos de los términos índices son todos binarios (i.e. $w_i^j \in \{0,1\}, w_i^q \in \{0,1\}$). Una consulta q es un subconjunto de términos índice.*

Sea R el conjunto de los documentos relevantes conocidos (o supuestos relevantes al inicio). Sea $\bar{R}$ el complemento de R, en otras palabras, el conjunto de los documentos no relevantes. Sea $P(R|\vec{d_j})$ la probabilidad de que el documento d_j sea relevante a la consulta q y $P(\bar{R}|\vec{d_j})$ la probabilidad que d_j sea no relevante a q. La similitud $sim(d_j, q)$ del documento d_j para la consulta q esta definida como la razón:

$$sim(d_j, q) = \frac{P(R|\vec{d_k})}{P(\bar{R}|\vec{d_j})} \tag{2.9}$$

Usando la regla de Bayes:

$$sim(d_j, q) = \frac{P(\vec{d_j}|R) \times P(R)}{P(\vec{d_j}|\bar{R}) \times P(\bar{R})} \tag{2.10}$$

Entiéndase $P(\vec{d_j}|R)$ como la probabilidad de seleccionar aleatoriamente el documento d_j de el conjunto R. Además, $P(R)$ da la probabilidad que un documento seleccionado aleatoriamente de la colección completa sea relevante. Los significados de $P(\vec{d_j}|\bar{R})$ y $P(\bar{R})$ son análogos y complementarios. Debido a que $P(R)$ y $P(\bar{R})$ son los mismos para todos los documentos en la colección, se puede escribir:

$$sim(d_j, q) \sim \frac{P(\vec{d_j}|R)}{P(\vec{d_j}|\bar{R})} \tag{2.11}$$

Asumiendo términos independientes:

$$sim(d_j, q) \sim \frac{(\prod_{g_i(\vec{d_j})=1} P(k_i|R)) \times (\prod_{g_i(\vec{d_j})=0} P(\bar{k_i}|R))}{(\prod_{g_i(\vec{d_j})=1} P(k_i|\bar{R})) \times (\prod_{g_i(\vec{d_j})=0} P(\bar{k_i}|\bar{R}))} \tag{2.12}$$

$P(k_i|R)$ es la probabilidad de que un término índice k_i está presente en un documento seleccionado al azar del conjunto R. $P(\bar{k_i}|R)$ es la probabilidad de que el término índice k_i no esté presente en un documento aleatoriamente seleccionado del conjunto R. Las probabilidades asociadas con el conjunto $\bar{R}$ tienen un significado análogo.

Tomando logaritmos, teniendo en cuenta que $P(k_i|R) + P(\bar{k_i}|R) = 1$, e ignorando factores que son constantes para todos los documentos en el contexto de la misma consulta, es posible escribir:

$$sim(d_j, q) \sim \sum_{i=1}^{t} w_{i,j} \times w_{i,j} \times (\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\bar{R})}{P(k_i|barR)}) \tag{2.13}$$

La cual es la expresión para el computo del ranking en el modelo probabilístico.

Si se desea un proceso no asistido por el usuario, es necesario disponer de un método que compute inicialmente las probabilidades $P(k_i|R)$ y $P(k_i|\bar{R})$. Por ejemplo, se puede simplificar asumiendo algunas propiedades como que $P(k_i|R)$ es constante para todos los términos índice (típicamente 0.5) k_i y se asume que la distribución de los términos entre los documentos no relevantes puede ser aproximada por la distribución de términos entre todos los documentos de la colección, entonces se puede escribir:

$$P(k_i|R) = 0.5$$

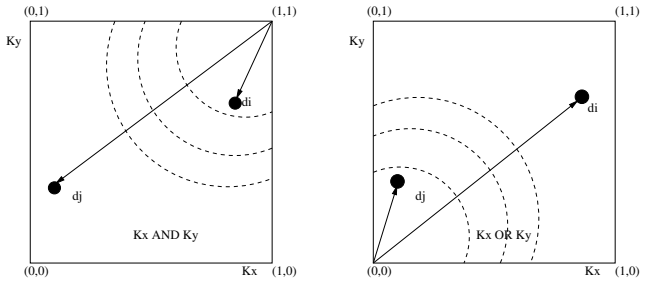
$$P(k_i|\bar{R}) = \frac{n_i}{N}$$

Donde n_i es el número de documentos que contienen el término índice k_i y N es el número total de documentos en la colección. Para más opciones para el cálculo inicial de probabilidades ver [Baeza-Yates99].

La principal ventaja del modelo probabilístico es su base teórica, la cual permite que el orden de presentación de los documentos está dado por la probabilidad de que los documentos sean relevantes. Las desventajas incluyen la necesidad de separación en conjuntos de relevancia y no relevancia, el hecho de que el método no toma en cuenta la frecuencia con la cual un término aparece dentro de un documento, todos los pesos son binarios, y claro, la independencia de términos (claro que no hay evidencia que éste supuesto sea una verdadera desventaja en situaciones prácticas).

Otros modelos

Existen muchos otros modelos, los cuales resuelven algunas de las debilidades inherentes de los modelos clásicos: modelo booleano, modelo vectorial y modelo probabilístico. Sin embargo, en la práctica los modelos más usados están basados en los modelos clásicos debido a su relativamente sencilla implementación, y principalmente su buen desempeño [Baeza-Yates99]. En el modelo vectorial se consideran pesos de los términos, lo cual determina la importancia de un término en un documento específico y de un término a la



(a) ***Operación booleana extendida AND*** **(b)** ***Operación booleana extendida OR***

Figura 2.1: Operaciones Booleanas Extendidas con dos términos k_x y k_y

colección, expresando naturalmente la importancia de los términos para una consulta dada.

Ésta experiencia previa, ha influido en la decisión que en ésta tesis se implementen solo éstos modelos, sin limitar la implementación futura de otros modelos. A continuación se dará un breve resumen de otros modelos, para un estudio más extenso es necesario revisar al menos las siguientes referencias [Baeza-Yates99, Turtle92, Hiemstra00].

Modelo Booleano Extendido El modelo booleano permite coincidencias parciales y pesos en los términos, pero con la formulación de consultas basada en el modelo booleano.

Un documento d_j se posiciona en el espacio a través de la adopción de pesos. Se asume que el pesos normalizados caen entre 0 y 1. El computo de los pesos pueden estar basados en factores $TF \times IDF$, por ejemplo:

$$w_x^j = f_x^j \times \frac{idf_x}{idf_{max}^j} \tag{2.14}$$

donde f_x^j es la frecuencia normalizada del término k_x en el documento d_j e idf_i es la frecuencia inversa del documento para un término genérico k_i. Los pesos de un término en un documento posicionan al vector documento en el espacio, cuantificando la factibilidad de que un documento sea relevante. Por ejemplo, supóngase dos términos, esto crea un espacio bidimensional, como en la Figura 2.1, donde una consulta conjuntiva $q_{and} = k_x \wedge k_y$ es mostrada en 2.1(a), donde la distancia al punto $(1, 1)$ es medido por las flechas, entre

mayor sea la distancia menor será la relevancia del documento. Por otra parte, una consulta disyuntiva $q_{or} = k_x \vee k_y$ se muestra en la Figura 2.1(b), en donde claramente se aprecia que entre más relevante sea un documento, éste deberá alejarse del punto $(0, 0)$. Las funciones de similitud usadas para dos términos es:

$$sim(q_{or}, d_j) = \sqrt{\frac{x^2 + y^2}{2}}$$

$$sim(q_{and}, d_j) = 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}$$

Ésta es la distancia euclidiana o norma, la cual puede ser extendida fácilmente para más de una dimensión. Adicionalmente, es posible usar *normas-p*, para más información ver [Baeza-Yates99].

Modelo de Indexamiento de Semántica Latente. La idea principal del Modelo de Indexamiento de Semántica Latente es mapear cada documento y vectores de consulta en un modelo dimensional menor, el cual está asociado a conceptos. Para más información pueden verse las referencias [Dumais88, Baeza-Yates99]

Modelos para Recuperación de Texto Estructurado y Semi-estructurado. Los modelos de recuperación de texto estructurado aprovechan la estructura del documento para permitir al usuario especificar con más precisión en que lugar desea encontrar ocurrencias. El texto estructurado y semi-estructurado se refiere a la clasificación del texto dentro del mismo, por ejemplo en un libro el texto se ordena en capítulos, apéndices, secciones, subsecciones, párrafos, títulos, etcétera. Este tipo de texto, es muy común en ámbitos formales, como en los artículos científicos o textos educativos. Existen varios modelos [Baeza-Yates99], dos de los más famosos son el modelo de **listas sin solape** y el modelo basado **en nodos próximos**. Ambos cuentan con lenguajes especializados, que permiten al usuario especificar con precisión las localidades de las ocurrencias.

Las *listas de ocurrencias*, dividen el texto en listas de diferentes niveles en la estructura, sin que las mismas se solapen en el mismo nivel de estructura, pero pueden solapen en diferentes niveles. Por ejemplo, para un libro se crean listas de capítulos, y cada

capítulo tiene asociado una sublista de secciones que no comparte con ningún otro capítulo, pero cada sublista forma parte una lista de secciones.

El modelo de *nodos próximos*, crea una jerarquía de estructuras de indexamiento sobre el mismo texto, y cada una de éstas estructuras está compuesta por capítulos, secciones, párrafos, páginas y lineas que son llamadas nodos. Cada uno de estos nodos está asociado a una región de texto. Este modelo puede resolver consultas más complejas que el modelo de listas sin solape.

2.5. Recuperación de Información Paralela y Distribuida

Las colecciones de documentos se han hecho cada vez más grandes, volviendo más costoso el mantenimiento de sistemas de recuperación de información. Los costos de buscar e indexar crecen con el tamaño de la colección de documentos; colecciones grandes de documentos invariablemente resultan en tiempos de respuesta mayores. Mientas más documentos son añadidos al sistema, el desempeño se puede ver deteriorado hasta el punto donde el sistema no es utilizable. Más aún, la sobrevivencia económica de sistemas comerciales y las máquinas de búsqueda de la Web dependen de sus habilidades para procesar altas cantidades de consultas eficientemente. De hecho, la mayoría de las compañías de Web, se mantienen de "avisos comerciales" (avisos incrustados en las páginas Web desplegadas o visitadas por los usuarios) cuyo número es proporcional al número de consultas resueltas.

Para soportar los requerimientos de ambientes modernos de búsqueda, es necesario buscar arquitecturas y algoritmos alternativos. Estas nuevas arquitecturas están basadas en cómputo paralelo y distribuido, el cual tiene la capacidad de escalar los recursos de un sistema adicionando procesadores o computadoras a una red. Los algoritmos necesarios para desempeñarse bajo estás nuevas arquitecturas son diseñados para explotar las características de éstas arquitecturas y sacar el mayor provecho posible.

El cómputo paralelo en la RI permite el uso de grandes cantidades de memoria compartida para utilizar bibliotecas o zonas de memoria compartidas como caches o índices. Existen diferentes aproximaciones para los algoritmos de recuperación de información paralela al resolver consultas. Las consultas pueden ser resueltas principalmente en dos

formas:

- Cada procesador resuelve una consulta, por lo que es posible resolver varias consultas a la vez.
- Cada procesador resuelve un término en la consulta y un procesador "controlador" controla el proceso y une los resultados.

Para aprovechar mejor los recursos de una máquina paralela es necesario realizar particiones de la colección en subcolecciones (semánticamente separadas o aleatoriamente distribuidas). Durante el indexamiento es necesario realizar algunas de consideraciones: Para crear índices por cada subcolección, términos segmentados, replicación total de índices, etcétera. En el proceso de consulta es necesario tener un "broker" que decida a que índices consultar para evitar la difusión de la consulta a todos los nodos.

En el cómputo distribuido se debe proceder de manera semejante al paralelo, pero teniendo en cuenta que la comunicación entre procesadores es de un alto costo, por lo tanto generalmente se tienen nodos que comparten poca o ninguna información. Es necesario remarcar que el cómputo distribuido es mucho más barato que el paralelo, y a gran escala, mucho más redituable.

2.6. Sumario

Éste Capítulo es una introducción a las bases de la Recuperación de Información, por lo que una serie de conceptos nuevos fueron introducidos tales como texto completo, vista lógica de documentos, ponderación o ranking, indexamiento, consulta, modelos de Recuperación de Información, etcétera. Además, se muestra la diferencia entre Recuperación de Información y Recuperación de Datos, haciendo hincapié en la flexibilidad del modelo de RI en la tolerancia de errores, que no es factible en la Recuperación de Datos.

También se estudia el proceso general de recuperación es mostrado, que podemos resumir como:

- Adquisición de los documentos.

- Indexamiento de los documentos.
- Resolución de consultas y ponderación de resultados.
- Retroalimentación por parte del usuario sobre la efectividad del proceso de recuperación.

Adicionalmente se describe uno de los principales problemas de la Recuperación de Información, el cual consiste en encontrar información pertinente a consultas efectuadas por el usuario así como la ponderación de las respuestas. La ponderación, muchas veces llamada *ranking* se describe como un problema crítico para la RI. También, se muestra la caracterización formal de los modelos de Recuperación de Información, teniendo como objetivo ser capaces de entender, crear y extender los modelos de RI.

Como parte del contenido del Capítulo se muestra la vista lógica de los documentos, parte fundamental para iniciar el proceso de recuperación, así como piedra angular para la implementación y diseño de los SRI y de los modelos. La vista lógica mostrada es la aplicada para los modelos dirigidos por términos índice, es decir, caracterizar un documento como un vector de términos. También se muestran las transformaciones que pueden ser aplicadas sobre los documentos para reducir el vocabulario y simplificar la vista de los documentos, estos son: eliminación de *palabras basura*, *stemming*, identificación de sustantivos y *normalización*.

Se discuten los modelos de recuperación más importantes haciendo énfasis en el modelo booleano y el modelo vectorial, ya que estos modelos han sido escogidos para ser usados en ésta tesis, debido a la simplicidad y eficiencia en su implementación, así como su popularidad en el mundo comercial, ya que están probados sus buenos resultados tanto en eficiencia como en calidad.

Finalizando el Capitulo se muestran los conceptos básicos de la Recuperación de Información Paralela y Distribuida, la cual extiende a la RI clásica usando una cantidad mayor de entidades de procesamiento. Se escoge para ésta tesis la RI distribuida, ya que es más barata y escalable que la RI Paralela, a costa de la complicación de desarrollo y mantenimiento, lo cual es muchas veces aceptable.

Capítulo 3

Objetos Remotos

El manejo de objetos remotos es un tema central en Sistemas de Recuperación de Información (SRI) distribuidos, así como también en los Sistemas Colaborativos, P2P y programación orientada a agentes. Los sistemas actuales para Mediación de Objetos Remotos (ORB por sus siglas en inglés) enfocados a la unión de redes heterogeneas y tolerantes a fallos contienen grandes sobrecargas de procesos debidas al análisis de los mensajes. Estas sobrecargas representan una fracción significativa de la latencia total en tareas de Recuperación de Información, y otras áreas.

Para resolver estos problemas en ésta tesis se presenta una nueva arquitectura para la Mediación con Objetos Remotos, llamada SPyRO (*Simple Python Remote Objects*), que se enfoca en alcanzar comunicaciones eficientes en redes heterogeneas, manteniendo independencia de arquitectura. Cabe señalar que aunque SPyRO está escrito en Python [Foundation05], es posible el acceso desde desde otros lenguajes de programación, permitiendo crear programas en cualquier lenguaje de programación. Utilizar SPyRO con Python tiene ventajas adicionales que utilizarlo con cualquier otro lenguaje, por ejemplo, SPyRO provee acceso a objetos remotos de manera transparente en Python, permite implementar *código móvil* [Fuggetta98] (i.e. La capacidad para cambiar dinámicamente las ligaduras entre fragmentos de código y la localidad donde el código es ejecutado). Las conexiones en SPyRO no tienen estado, lo que significa que los objetos locales y las llamadas remotas no deben preocuparse por el estado de la conexión.

El protocolo de SPyRO soporta las operaciones estándar de objetos, tales como asignación de valores a atributos, recuperación de valores ligados a atributos y ejecución de métodos remotos. El uso de Python [Foundation05] asegura portabilidad y fácil mantenimiento del código.

SPyRO es una parte muy importante del presente trabajo, pues gracias a él se realiza la interconexión de los módulos y periféricos distribuidos, abstrayendo la comunicación entre nodos.

Éste Capítulo es presentado en el siguiente orden: En la sección 3.1 se presenta un panorama general del trabajo relacionado, haciendo énfasis en los productos más populares. La sección 3.2 presenta la arquitectura desarrollada con el fin de crear SPyRO. Los tipos de envío de argumentos y recepción de resultados son presentados en la sección 3.3. La sección 3.4 muestra la conveniencia de usar SPyRO para comunicar procesos remotos y de como realizar las conexiones. Los procesos de compartir archivos se muestran en la sección 3.6. En la sección 3.5 se da la demostración formal del esquema de conexiones utilizado por SPyRO. En la sección 3.7 se presentan los experimentos efectuados que avalan la efectividad de SPyRO al manejar objetos, y finalmente, la sección 3.8 es un breve sumario del capítulo.

3.1. Trabajo Relacionado

Además de SPyRO, existe un gran número de mediadores. A continuación se presentarán los trabajos algunos de los trabajos relacionados de mayor popularidad e importancia.

Existen muchos protocolos que permiten la comunicación entre nodos, algunos de los más populares son SOAP [Koftikian01], CORBA [Vinoski97] y RMI [Microsystems05] con orientación a objetos y XMLRPC [Allman03] y RPC [Srinivasan] para llamadas a procedimientos remotos.

CORBA [Vinoski97] es poderoso y resuelve los requerimientos en el manejo de objetos remotos, pero es débil por su complejidad. CORBA no es apropiado para sistemas pequeños, ya que no esta diseñado para *ambientes pervasivos*. Sin este enconveniente, CORBA es suficiente (y muchas veces sobrado) para la mayoría de aplicaciones basadas en

Objetos Remotos.

Ice (The Internet Communication Engine [ZeroC05]) es una poderosa plataforma de desarrollo con Objetos Remotos, en su lista de lenguajes de programación se encuentran C++, Java, Python, PHP y Visual Basic. Ice ha sido diseñado para programadores prácticos y es necesario utilizar clientes y servidores Ice. Desgraciadamente, sólo soporta solo un formato para la comunicación y códificación de mensajes excluyendo algunos de los ORB más populares. Adicionalmente, no es un prestador de Servicios Web.

DCOM (Distributed Component Object Model [Microsoft]) permite mediante RPCs comunicar procesos de manera transparente a lo largo de computadoras con el sistema operativo Windows. Claramente, DCOM es dependiente de la arquitectura y no soporta Servicios Web.

El Protocolo Simple de Acceso a Objetos (SOAP por sus siglas en inglés) es un estándar en los servicios web, está basado en XML [Yergeau04], definiendo el formato y reglas para describir su contenido. Está visto, en conjunto con WSDL y UDDI, como los tres estándares de servicios web, siendo el protocolo más popular para el intercambio de servicios web. Mantiene una independencia total de la arquitectura y la plataforma de las máquinas que intercambian mensajes, a costa de añadir una sobrecarga no siempre necesaria al análisis y transporte de los mensajes utilizados.

Los protocolos RPC y XMLRPC son capaces de manejar llamadas a procedimientos remotos, pero no pueden compartir objetos.

Existen implementaciones independientes basadas en Python para objetos remotos. Pyro [deJong05], está completamente escrito en lenguaje Python. Es pequeño y portable, pero no es un proveedor de Servicios Web y debe ser accedido exclusivamente por otros nodos Pyro, ya que usa Pickle[1] [Foundation05, vanRossum] para codificar y decodificar objetos. Pyro utiliza un servicio global de resolución de nombres, por lo que existe un punto de falla. Si el servidor de nombres falla, la comunicación entera estará comprometida. Este tipo de ORB no es recomendable para aplicaciones sometidas a un alto índice de fallas, tales como redes P2P. SPIRO [McNab95] es un ORB desarrollado en Python, especialmente creado para trabajar en Jython [Pedroni02]. Por otra parte, el paquete Twisted [Toolkit], es

[1]Pickle es un protocolo de serialización dependiente y exclusivo para objetos Python

una excelente herramienta, no es dependiente de Python y permitiendo varios protocolos de comunicación como SOAP y XMLRPC. Desgraciadamente, no provee acceso transparente a los objetos, por lo que las aplicaciones creadas deberán ser creadas para twisted, y lo más importante, no hay posibilidad de crear código móvil. Esto forza a siempre utilizar objetos remotos en una aplicación desarrollada, aún cuando los objetos sean locales y en el mismo proceso.

Algunas de las formas más modernas y populares de comunicar redes con sistemas heterogéneos suelen ser cuellos de botella en sistemas de Recuperación de Información. Por ejemplo, OmniPaper [Technologies05a, Technologies05c, Technologies05b], probablemente el Sistema de Recuperación de Información Distribuido más ambicioso (de código cerrado y comercial) gasta del 40 % al 54 % del tiempo total de resolución de las consultas en transportar y analizar mensajes de SOAP [Technologies05a].

Los requerimientos para la interconexión de nuestro Sistema de Recuperación de Información (portabilidad, independencia de arquitectura, eficiencia, pervasividad, facilidad de programación e integración en el ambiente de desarrollo) no son cubiertos en su totalidad por los ORBs presentados. Debido a esto, se decidió crear e implementar una nueva arquitectura que llenase los requerimientos de la recuperación de información distribuida y pervasiva.

3.2. Arquitectura de SPyRO

SPyRO es un *ORB*, creado en Python con la intención de comunicar eficientemente y fácilmente procesos que existan en la misma o diferentes computadoras. Es transparente al programador que utiliza Python, ya que maneja la comunicación de igual manera que si se manejaran objetos locales. En el caso más extremo (uso a nivel experto, es decir utilizando todas sus funcionalidades) es un ORB translúcido. Sin embargo, SPyRO no posee un IDL por lo que las aplicaciones deben ser escritas directamente en lenguaje Python, simplificando el proceso de desarrollo.

En el nivel de implementación, se escogió Python [Foundation05] porque sus características de introspección y dinámismo permiten una fácil y rápida implementación del

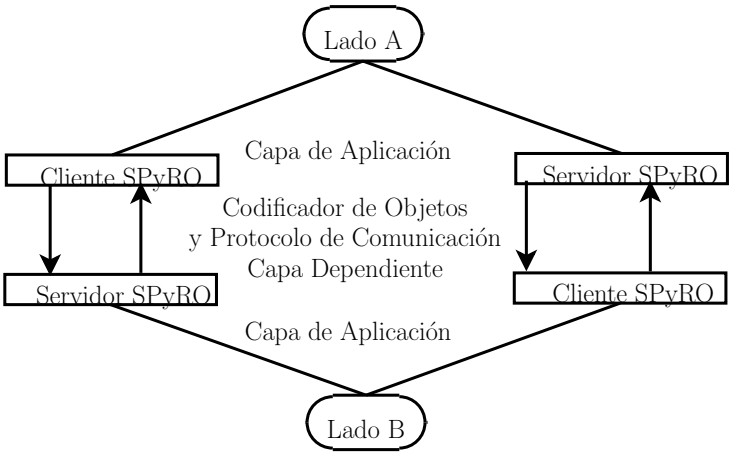


Figura 3.1: Diagrama simplificado de la arquitectura de SPyRO

ORB. Además, Python es un lenguaje muy popular y está disponible para una amplia gama de arquitecturas y sistemas operativos. Básicamente Python, al igual que Java [Sun], cuenta con una máquina virtual que interpreta *bytecode* [2] independiente de la computadora que lo ejecute y únicamente dependiente del lenguaje de programación. SPyRO permite construir aplicaciones distribuidas de una manera muy simple y rápida.

Una conexión de SPyRO se crea mediante dos nodos o *partes*. Cada nodo en una conexión puede ser cliente, servidor o ambos roles a un mismo tiempo. En la Figura 3.1 se muestra como ambas partes de la conexión (Lado A y Lado B) funcionan como servidores y como clientes; también se aprecian los componentes básicos de la arquitectura de SPyRO, así como las tres capas de alto nivel con las que esta formado el protocolo. La capa de aplicación es el API utilizado para programar, si se utiliza la forma transparente de SPyRO, las capas son transparentes al programador. La etapa de codificación y decodificación de objetos se encarga de interpretar los mensajes, dandoles un significado. Esta etapa es realizada en las entidades *Cliente SPyRO* y *Servidor SPyRO* de la Figura 3.1. El protocolo HTTP[Consortium] es el utilizado para el envío y recepción de mensajes.

La parte cliente siempre inicia las conexiones haciendo peticiones que son respondidas por la parte servidora. Si ambas partes de la conexión se mantienen tanto como servidor y cliente a la vez, entonces ambas pueden iniciar una comunicación. En la parte del cliente,

[2] El *bytecode* es un código intermedio, más abstracto que el código máquina

SPyRO tiene un protocolo síncrono, heredado de la capa de transporte HTTP. Sincronía en el protocolo implica que una petición debe ser resuelta antes de realizar otra petición. En programas multitarea esto es una limitante ya que hacer varias peticiones por un mismo canal de comunicación puede corromper la comunicación.

Desde otra perspectiva, la sincronía ayuda a implementar o cambiar de manera transparente, objetos locales por clientes SPyRO, ya que el comportamiento de los objetos locales es también síncrono en las llamadas a métodos, asignación y obtención de atributos. El efecto de implementar la sincronía implica detener el flujo del programa hasta que la respuesta sea recibida por parte del servidor. Existen algunos protocolos asíncronos (e.g Twisted [Toolkit]). Estos son protocolos no transparentes y requieren una forma diferente de pensamiento por parte del programador, además de la creación de programas especialmente escritos para el protocolo.

Al costo de mantener varias conexiones, se provee una interfaz transparente para manejar objetos remotos en una aplicación multi-tarea. Esto sin ningún código o complejidad adicional de parte de los programadores. El número de canales puede ser modulado para mayor control de recursos. El número de hilos puede llegar al límite permisible (número para el cual está optimizado el sistema operativo) cada proceso y el sistema operativo en general.

Si existe un número tal de tareas que los límites del SO son alcanzados, es posible que el sistema y los programas basados en SPyRO se inutilicen. Debido a esto, la cantidad de recursos utilizados por SPyRO pueden ser configurados desde las aplicaciones, el SO debe proporcionar maneras para configurar estos parámetros. Por ejemplo Linux [Tolvards] proporciona una interfaz directa para cambiar parámetros del *kernel* mediante el sistema de archivos *proc* para incrementar el número de procesos y tareas.

Si se usa Python, el manejo de objetos es transparente, esto es, sin importar la localización real del objeto, el programador únicamente debe preocuparse por el desarrollo de la lógica de sus programas. El servidor de SPyRO hace un fuerte uso de *hilos* o *threads*, pero a diferencia del cliente, el servidor no necesita enmascarar la sincronia del protocolo, ya que su diseño permite aplicaciones multi-hilo.

Dado que SPyRO basa su capa de transporte en HTTP y tiene soporte para

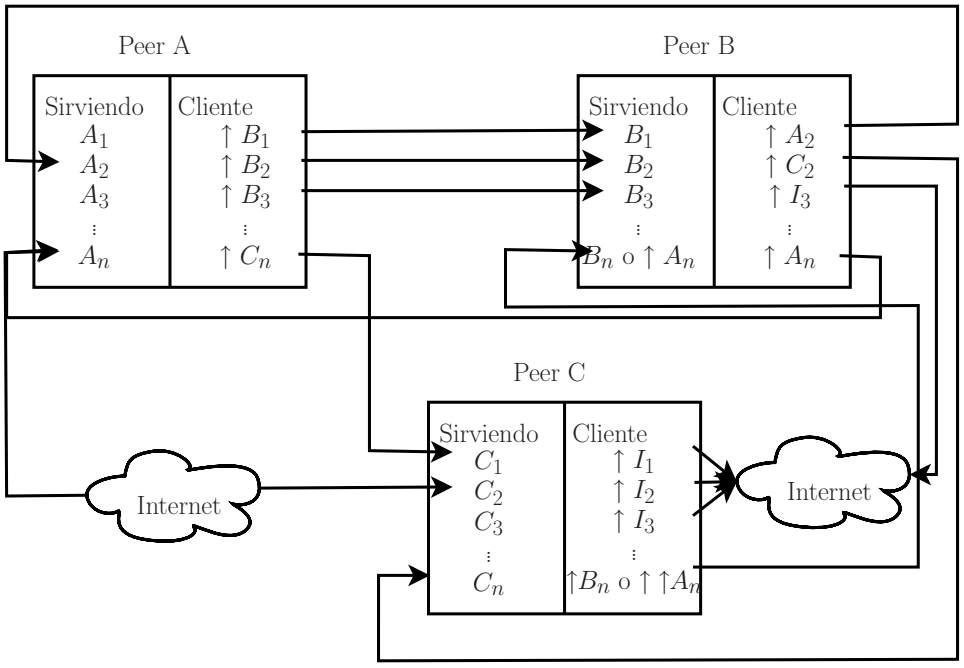


Figura 3.2: Objetos compartidos entre 3 nodos y la Internet

protocolos basados en XML, puede ser considerado un prestador de Servicios Web. Una ventaja adicional de usar el protocolo HTTP y su puerto tradicional (puerto 80) es que tradicionalmente ese puerto se encuentra abierto en la mayoría de los *Cortafuegos*.

SPyRO está formado por un servidor de objetos que permite compartir un conjunto de objetos registrados. Por lo tanto, debe existir un cliente realizando una petición para realizar una operación sobre algún objeto. Un ejemplo de la arquitectura se muestra en la Figura 3.2, donde muestran tres computadoras con SPyRO, donde los procesos de SPyRO tienen los roles de cliente y servidor. Los servidores están compartiendo objetos, que podrían ser tanto objetos locales como objetos remotos alojados en otros servidores. Los clientes tienen representaciones locales de objetos remotos, dichas representaciones son llamadas objeto *proxy*. En la figura 3.2, las referencias a objetos remotos son representadas con el símbolo $\uparrow$, seguido del nombre del objeto remoto. La Figura muestra conexiones en la Internet que pueden ser servidores o clientes de servicios web SPyRO.

Las señales de control en el protocolo son reducidas a una señal: la señal *bye*. La señal *bye*, realiza el cierre formal de una conexión entre un cliente y un servidor.

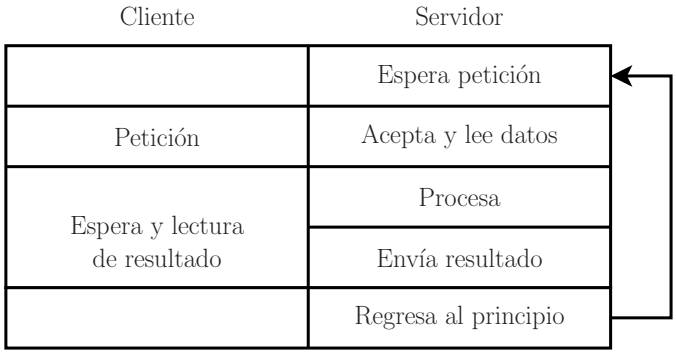
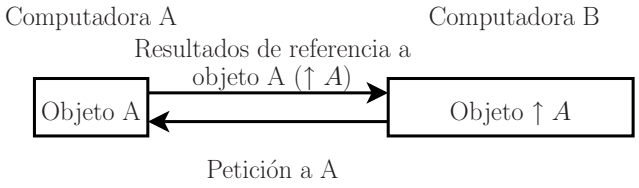


Figura 3.3: Intercambio de datos y el proceso de petición en SPyRO

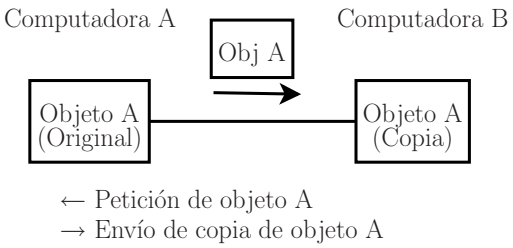
Figura 3.4: Pila del protocolo

El intercambio de datos de una petición de SPyRO obedece a una secuencia rígida (a consecuencia de la simplicidad del protocolo), como se muestra en la Figura 3.3, la secuencia mostrada está dividida en los roles de cliente y servidor, y sus interacciones. En el proceso mostrado se asume que ambos lados ya están conectados y al final de la petición permanecen conectado; cuando la conexión se establece, se pueden anteponer algunos pasos adicionales como autenticación (por HTTP [Consortium] o efectuado internamente), selección del formato de broker (la cual se verá en §3.4) o alguna configuración preliminar entre ambas partes de la conexión. Adicionalmente, cada objeto y operación puede ser protegida utilizando autenticación.

La pila del protocolo está basada en el cambio a la capa del formato del ORB, mejorando en el transporte y análisis de los datos. En el peor de los casos la mejoría no es posible, pero siempre mantiene la compatibilidad entre diferentes arquitecturas y plataformas.



(a) *Pasando objetos por referencia*



(b) *Pasando objetos por valor*

Figura 3.5: Formas de envío de objetos en SPyRO

3.3. Modos de Envío de Objetos

Un punto clave en el diseño de SPyRO es el Modo de Envío de Objetos (*MEO*). El MEO impacta directamente sobre la latencia del protocolo, y por lo tanto sobre la eficiencia de la conexión. El envío de objetos ocurre como paso de argumentos en las llamadas a métodos remotos, el valor en una asignación, el valor en la consulta de un atributo o el resultado al ejecutar un método.

Existen dos posibles modos de envío en SPyRO (ver Figura 3.5):

- Envío o paso de objetos por valor (*EV*).
- Envío o paso de objetos por referencia (*ER*).

Ambos MEO tienen ventajas y desventajas asociadas con su funcionalidad. Por lo tanto, se exponen ambas formas para justificar las decisiones de diseño.

3.3.1. Envío de Objetos por Referencia

El ER es el paso natural de un objeto, desgraciadamente el ER entre nodos remotos tiene un alto costo que debe ser ponderado. Primero, para mantener la compatibilidad con el transporte HTTP es necesario sostener dos conexiones, debido a que los clientes de una petición pueden enviar referencias. Por consecuencia, ambas partes de la conexión deben ser siempre clientes y servidores, manteniéndose siempre disponibles mientras dure la conexión.

Cuando ambas partes deben ser servidores, es necesario permitir conexiones en ambas direcciones y esto no siempre es posible. Existen Proveedores de Servicios de Internet (ISP por sus siglas en inglés) que protegen sus redes utilizando Cortafuegos o no disponen de suficientes direcciones IP para abastecer a sus clientes. Si el ISP no tiene contemplado permitir entradas a las computadoras dentro de su red interna desde el exterior y no provee configuraciones que permitan estos accesos, no es posible mantener ambas partes de la conexión como servidores. Aún si se utiliza el puerto 80, es posible que el ISP proporcione el servicio a través de un *NAT* que no resuelva las correspondencias de puertos. En algunos sistemas operativos [Tolvards, Microsystems, Unix, Microsoft] los puertos debajo del 1024 son propiedad de los administradores, por lo que no es posible ejecutar SPyRO en el puerto 80 por un usuario sin privilegios de administrador.

La Figura 3.5(a) muestra como el ER es realizado, se aprecian dos direcciones en la misma conexión. Si la computadora A hace peticiones sobre objetos registrados en la computadora B, la parte superior utiliza la computadora A como cliente y la B como servidor, mientras que la inferior usa la computadora B como cliente y la A como servido, permitiendo enviar argumentos de una petición por referencia sin la restricción de envío por referencia únicamente de los resultados.

Sin embargo, la desventaja más importante de *ER* es la alta latencia del protocolo: La latencia del protocolo es directamente proporcional al número de peticiones y respuestas entre los actores de la comunicación.

3.3.2. Envío de Objetos por Valor

El modo de paso de objetos llamado envío por valor tiene una característica crucial: el protocolo implica baja latencia. Dado que el objeto se envía solo una vez. Además los puertos de salida rara vez se encuentran cerrados en los NATs y Cortafuegos, por lo que no se afectan las conexiones. Una gran desventaja de este MEO es que los objetos remotos son duplicados creando copias locales. Éste comportamiento lleva a copias incoherentes de los objetos (i.e. un objeto A en un lado puede cambiar sin reflejar los cambios en el objeto A del otro lado de la conexión 3.5(b)).

Otra desventaja del EV es que es necesario conocer los tipos y clases de los objetos en el lado receptor, mientras que en ER solo es necesario conocer la interfaz del objeto.

Para aumentar el desempeño de SPyRO se decidió utilizar EV en el modo transparente. ER puede ser utilizado en un modo translúcido de SPyRO. El modo translúcido tiene acceso tanto a ER como EV, dando completo control al programado. Existe un tercer modo de envío de objetos, basado en EV y ER, es llamado modo híbrido y está especialmente diseñado para el modo transparente en Python.

Modo híbrido de Envío de Objetos en el Modo Transparente

En el modo transparente existe una forma de envío de objetos enfocada a utilizar las ventajas de ER y EV. Desgraciadamente, algunas de las desventajas también son heredadas. Ésta forma es llamada modo híbrido de envío de objetos.

El modo híbrido consiste en realizar EV en los valores regresados de los métodos y la recuperación de atributos, evitando aumentar la latencia de manera desproporcionada (sería necesario realizar peticiones por referencia por siempre o hasta cumplir una condición de paro, como un tipo básico). El ER es utilizado en el paso de argumentos a llamadas de métodos, pero los tipos básicos son enviados por valor (i.e. enteros, flotantes, enteros largos, cadenas, el valor *ninguno* o *None*, booleanos, y las tuplas). La asignación de atributos es realizada siguiendo el mismo patrón.

Una optimización adicional para mejorar la latencia del protocolo en el modo híbrido y transparente es utilizar un cache de atributos ejecutables. Gracias a éste cache,

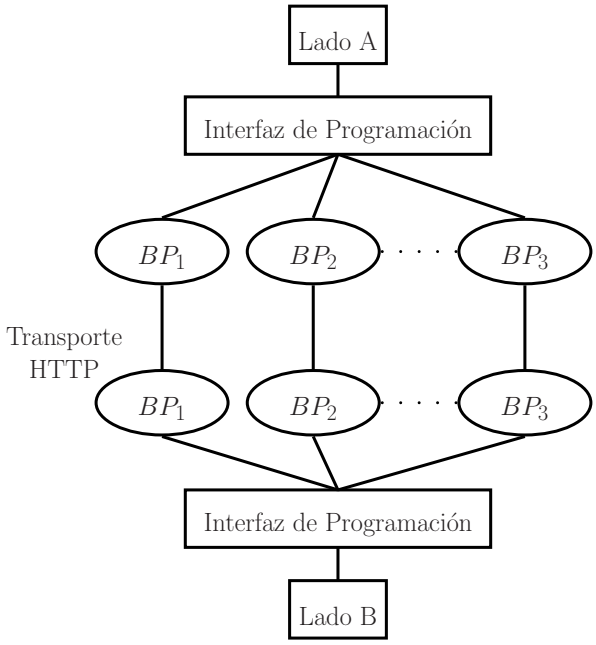


Figura 3.6: El formato de códificación repercute directamente sobre el desempeño (codificación, análisis y transporte)

la latencia durante el llamado de métodos se reduce aproximadamente en un 50 %, debido al proceso de ejecución en Python.

3.4. Conexión con SPyRO

SPyRO es un proveedor y cliente de Servicios Web, por lo que usa el protocolo HTTP de transporte para intercambiar datos entre las partes. La diferencia con ORBs tradicionales es que SPyRO cambia el formato de codificación de mensajes tratando de alcanzar una conexión eficiente e interpretación de los mensajes cuando es posible. No se debe olvidar la posibilidad de interconexión con otros proveedores y clientes diferentes a SPyRO, por lo tanto, es necesario soportar varios niveles de comunicación. SPyRO puede ser llamado un prestador y cliente de Servicios Web *poliglota*: es capaz de intercomunicarse con diferentes proveedores y clientes de servicios web que conozcan alguno de los formatos soportados, ver tabla 3.1. Ésta característica de SPyRO se muestra con mayor detenimiento en la Figura 3.6, donde muestran dos nodos que se comunican mediante SPyRO. En la Figura

es posible ver una gran cantidad de caminos (formatos) por los cuales la comunicación se puede efectuar, mientras que se programa de una sola forma (esto último únicamente es posible, por el momento, cuando se programa en Python y soporte para SPyRO, ya sea en modo transparente o translúcido).

Tabla 3.1: Propiedades de los formatos

Formato	Solo Python	Usa XML	Estándar W3C [Members05]	Orientado a Objetos
MARSHAL	Sí	No	No	No
PICKLE0	Sí	No	No	Sí
PICKLE1	Sí	No	No	Sí
PICKLE2	Sí	No	No	Sí
XMLGENERIC	Sí	Sí	No	No
WDDX	No	Sí	No	No
XMLRPC	No	Sí	No	No
SOAP	No	Sí	Sí	Sí

Actualmente se soportan los formatos SOAP, XMLRPC, XML (Genérico y WDDX), tres niveles de Pickle y Marshal de Python.

En formatos en los cuales la orientación a objetos no es soportada, se emula mediante parámetros adicionales a las funciones; sin embargo, se tiene la limitante al enviar objetos, ya que generalmente no se soporta.

En la mayoría de los casos, la portabilidad se mantiene en los formatos más ineficientes (SOAP, XMLRPC, WDDX y XML en general), mientras que los formatos más eficientes solo están disponibles para grupos específicos nodos (e.g. Pickle y Marshal [Foundation05]). La heurística utilizada para mantener la compatibilidad con mínimos sacrificios en la eficiencia es conectar cada par de nodos en la red utilizando el protocolo más eficiente que sea conocido por el par de nodos. Entonces, para seleccionar el formato de broker C, es necesario escogerlo desde una lista de formatos con su prioridad asociada. La forma de hacerlo es sencilla, como se muestra en la siguiente fórmula:

$$C = \max_{priority} \{F(A) \cap F(B)\}$$

Donde F es la función para obtener el conjunto de formatos de cada nodo. A y B son los nodos que serán conectados. La Figura 3.6 muestra está propiedad de SPyRO. La lista de formatos y su prioridad está asociada a cada BP_i en la Figura 3.6. La Figura 3.7

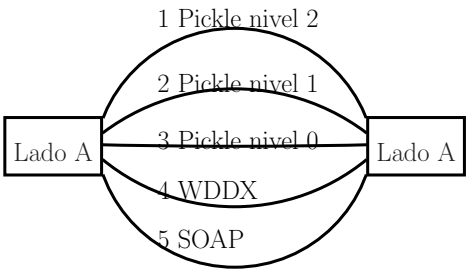


Figura 3.7: Costo de los protocolos

muestra un ejemplo de la prioridad de los protocolos. La prioridad se muestra a la izquierda del nombre del protocolo.

Otra forma de conectar partes es solicitar un formato especifico, ésta es la manera en la cual se conectan los nodos que no usan SPyRO.

El servidor HTTP donde SPyRO este alojado puede seguir realizando otros servicios a la vez, tales como servir páginas estáticas o páginas dinámicas creadas con CGIs [D. Robinson04], PHP [Achour05], PyEW (ver sección 6.1.2), etcétera.

Los servicios de SPyRO se denotan utilizando URIs, específicamente URLs los cuales son de la forma *protocolo://direccion-anfitrión:puerto/ruta?argumentos*. El protocolo HTTP se utiliza en la parte de especificación de protocolo del URL, la dirección del servidor del objeto se utiliza para denotar el anfitrión en el URL. La ruta obedece a la localización donde se encuentre registrado el servidor SPyRO en el sitio Web. El protocolo para el mediador de objetos se especifica después del símbolo `?` en la URL. Por ejemplo para nombrar al servidor `umich.mx` en la ruta `spyro` con el protocolo de codificación de objetos `SOAP` es necesario el URI `http://umich.mx/spyro?SOAP`, si el servidor HTTP escuchara en el puerto 8080 la URL es `http://umuch.mx:8080/spyro?SOAP`.

3.5. Costo Mínimo Global en la Conexión

Con el afán de incrementar el desempeño (específicamente disminuir el tiempo desperdiciado en la computación distribuida por el proceso de comunicación, codificación y decodificación de los datos) se utilizan diferentes protocolos o formatos para codificar los

mensajes. Sin importar que existan varios protocolos, en la capa de programación se encuentra la misma interfaz, provocando que el programador no se preocupe por los cambios en la capa de codificación, decodificación y envió de mensajes. En el proceso de escoger el mejor formato se utilizan heurísticas simples tratando de obtener comunicaciones optimizadas para los recursos de red prevalecientes en el momento de realizar la conexión. Estas características multi-formato van encaminadas a la abolición, cuando es posible, del alto costo de analizar XML (SOAP y XMLRPC), pero manteniendo formatos de alto nivel para asegurar la interconexión en una red heterogénea. Por lo tanto en protocolos de alto nivel (basados en XML u otro formato en texto) se preserva la capacidad de interconexión mientras que en protocolos de bajo nivel (Pickle [Foundation05, vanRossum] y otros protocolos binarios) se incrementa el desempeño en la comunicación. En otras palabras, SPyRO obtiene el incremento del desempeño de comunicación utilizando los recursos homogéneos entre cada par de nodos en una red heterogénea.

A continuación se prueba que la heurística es correcta, y que además, la utilización de ésta lleva no solo a la mayor eficiencia de comunicación entre nodos, si no también a la mayor eficiencia global en la red SPyRO.

Sea la *conexión mínima local* la mejor conexión posible entre dos nodos en una red. La mejor conexión posible es aquella que mejora la transmisión, codificación y decodificación, con respecto a todas las posibles conexiones. El costo de un formato está representado por un número asociado al protocolo, tal como se muestra en la Figura 3.7.

Definición 3.5.1 *Sea una* conexión mínima local *la que sea ofrecida por el formato con costo mínimo entre un conjunto de posibles formatos de conexión.*

Definición 3.5.2 *Sea una* conexión mínima global *la configuración de conexiones locales que optimice el desempeño de la red completa. Una conexión óptima global es la configuración de conexiones locales que resulte en la suma mínima de los costos asociados a los formatos utilizados.*

Teorema 3.5.1 *Utilizar la información local de prioridades en una red con múltiples formatos de conexión entre cada par de nodos, su costo es mínimo globalmente si es mínimo para cada par de nodos que realizan una conexión.*

Prueba al Teorema de Costo Mínimo Global de Conexiones 3.5.1. Este teorema puede ser probado por contradicción. Supóngase que existe una red en la cual existe al menos una conexión que no es la mínima local, por lo tanto existe una conexión con un costo menor que la utilizada. Dado que existe un costo menor, la suma total de las prioridades de las conexiones locales no puede ser mínima.

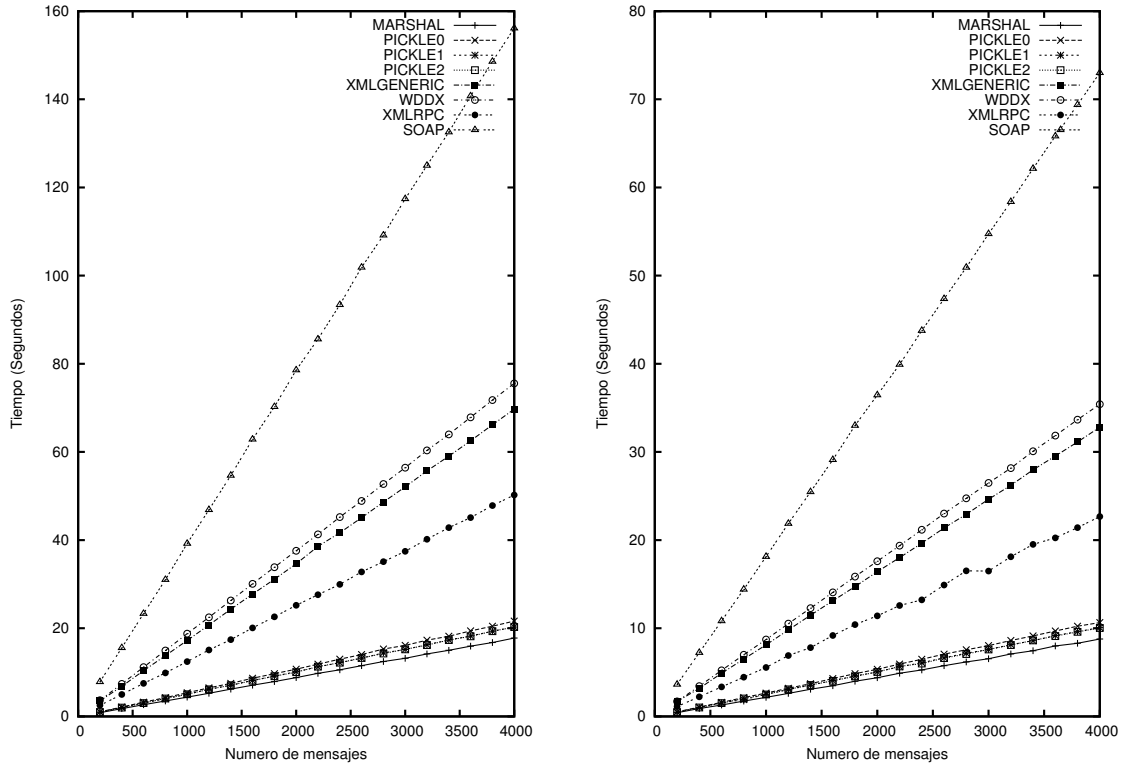
3.6. Compartiendo Objetos con SPyRO

Para utilizar un objeto es necesario registrarlo como *compartido* en el lado servidor y realizar una petición en el lado cliente. Existen tres diferentes formas de compartir objetos:

- **Registro explicito.** Es necesario registrar un objeto ligandolo a un identificador. El identificador debe ser único en el servidor y además debe ser un tipo básico, preferiblemente una cadena de carácteres en forma de URI [Berners-Lee05].
- **Atributos de objetos registrados.** Los atributos de los objetos registrados son compartidos implícitamente accediéndolos mediante el operador "." (punto). Por ejemplo, si un objeto o tiene como atributos a, b es posible acceder a estos atributos realizando peticiones como $o.a$ u $o.b$. Por consecuencia, si los atributos a, b contienen los atributos c, d es posible accederlos de la misma forma $o.a.c$ y $o.a.d$. Gracias a esta forma de acceder a los atributos es posible optimizar el acceso en el modo translúcido de SPyRO bajando la latencia de acceso, ya que no es necesario obtener o para acceder a a y luego hacer petición para c o d. Adicionalmente, es una forma de obtener ER.
- **Registro de resultados.** El registro de resultados es usado como una herramienta más en el ER. El resultado de la ejecución de un método es registrado como un objeto compartido, el identificador de registro se hace accesible en el momento del registro de resultado.

3.7. Resultados Experimentales

A lo largo del Capítulo se mostró el diseño de SPyRO. Además se probó la validez de las heurísticas utilizadas para optimizar las conexiones en una red SPyRO. A continua-



(a) ***Tiempo real de los mensajes*** **(b)** ***Tiempo de usuario de los mensajes***

Figura 3.8: Medición de los tiempos de comunicación con diferentes formatos de serialización

ción, se muestran los resultados de experimentos que muestran la efectividad de los métodos propuestos para acelerar conexiones. Los resultados muestran mejorías de hasta 1000 % de eficiencia ganada.

En una computadora Intel(R) Pentium(R) 4 a 2.4GHz con 256 de RAM, se probó una configuración simple de SPyRO: Ocho clientes con tiempo exclusivo de proceso, conectados a un servidor único. Cada cliente maneja un formato de ORB diferente tal como se muestra en la Figura 3.8. El experimento consiste en intercambiar mensajes con el propósito de demostrar que la carga añadida por la comunicación entre computadoras es un factor que puede ser reducido drásticamente utilizando las heurísticas de SPyRO. Los mensajes son los mismos para todas las pruebas.

La Figura 3.8(b) muestra el tiempo usado para codificar, decodificar e interpretar

un mensaje en una parte de la comunicación, especificamente el cliente. La Gráfica 3.8(a) muestra el tiempo real gastado para comunicar cliente-servidor. Este tiempo muestra no solo el tiempo usado para codificar, decodificar e interpretar los mensajes, sino también el necesario para realizar el intercambio entre procesos o computadoras.

Las relaciones entre el formato más rápido (*MARSHAL*) se muestran en la tabla 3.2, donde se observa como entre el formato MARSHAL (dependiente de Python) y SOAP (formato completamente portable) existe un relación que supera al 1000 %. Estos resultados, soportan la teoría de que la comunicación en una red heterogénea debe ser realizada utilizando las características homogéneas de la red.

Tabla 3.2: Resultados de la Comparación de Formatos

Formato	Relación
MARSHAL	1.0000
PICKLE0	1.1658
PICKLE1	1.0848
PICKLE2	1.0859
XMLGENERIC	3.8638
WDDX	4.2691
XMLRPC	3.9435
SOAP	10.6077

3.8. Sumario

Durante el presente capítulo se mostró un nuevo ORB, SPyRO, el cual es capaz de utilizar varios formatos para los mensajes y soportar servicios web de manera natural. También se mostraron los modos de operación de SPyRO, es decir, el modo translúcido y el transparente. Cada uno de estos modos aporta una característica importante a SPyRO, tal como código móvil en el modo transparente y control sobre los modos de envío en el modo translúcido. La tabla 3.3 muestra una comparación sobre los ORBs analizados en éste trabajo de tesis mostrando a SPyRO superior en los campos de interés listados en la tabla.

Adicionalmente se mostró la efectividad de las heurísticas usadas en SPyRO mediante los experimentos mostrados en la sección 3.7, obteniendo mejoras de más del 1000 %.

Tabla 3.3: Comparación de ORBs

ORB	Soporta Múltiples Formatos	Servicios Web	Basado en Estándares de Internet	Orientado a Objetos	Apto para Computo Pervasivo	Modo Translúcido en Python	Modo Transparente en Python	Control sobre los modos de envío
XMLRPC	No	Sí	Sí	No	Sí [1]	Sí	Sí	No
SOAP	Sí [2]	Sí	Sí	Sí 1	Sí	Sí	No	?
CORBA	Sí	No	Sí	Sí	Sí [3]	Sí	Sí	Sí
DCOM	No	No	Sí	Sí	Sí [3]	No	No	Sí
Ice	No	No	No	Sí	Sí	?	?	?
Twisted	Sí	No	Sí	Sí	Sí	Sí	No	No
PyRO	No	No	No	Sí	Sí	No	Sí	No
PIRO	No	No	No	Sí	Sí	No	Sí	No
SPyRO	Sí	Sí [4]	Sí [4]	Sí	Sí	Sí	Sí	Sí

1. Su base en el lenguaje XML lo hace portable pero lento debido al complejo análisis de los mensajes resultantes.
2. Soporta XML con diferentes esquemas.
3. De manera incompleta, ya que es muy complejo para la especificación completa.
4. Puede variar dependiendo del formato, para más información ver la tabla de formatos 3.1.

Capítulo 4

Adquisición de los Datos

El proceso llamado *adquisición de datos* consiste en obtener los datos o documentos para realizar las funciones del sistema. El módulo de adquisición de datos creado en éste trabajo soporta dos formas básicas de adquisición de datos: el método *pull* y el método *push*. Ambos métodos son utilizados en la Web para la recolección masiva de datos.

El método *pull* está ligado principalmente con las *arañas recolectoras de datos* y *cosechadores*. El método *push* con los procesos controlados por los usuarios, como peticiones en formas HTML el método POST de HTML.

La recolección y adquisición de datos mediante un proceso automático es un área de investigación en la Web, ya que la cantidad existente de de información sobrepasa las capacidades de los sistemas actuales de adquisición de información. Debido a esto, el obtener solo la información pertinente a un tema especifico, identificar la duplicidad de los datos recolectados, mantenimiento de la colección, capacidad de extracción y adquisición de los datos son áreas de investigación en continua evolución.

La tasa de crecimiento y cambio de contenido de la Web requiere soluciones donde el escalamiento es fundamental, sin ésta característica un sistema es simplemente inservible en Internet. Debido a esto, y a las características intrínsecas de la recolección de datos en la Web, los sistemas de recopilación de datos se han convertido en su mayoría en sistemas distribuidos, altamente eficientes, cuya eficiencia va ligada directamente al número de nodos que forman parte del sistema distribuido. La especialización de la recolección de datos

sobre un tópicos específicos de información también permite ahorrar ancho de banda y obtener un mayor número de documentos relevantes para la especialización de la colección en menor tiempo. De igual manera, el enfoque tomado en la arquitectura propuesta por éste trabajo es el de obtener los beneficios del cómputo distribuido, permitiendo a las aplicaciones generadas, ser lo suficientemente extensibles para soportar colecciones de grandes cantidades de documentos.

El presente Capítulo es presentado de la siguiente forma: La sección 4.1 analiza y presenta un breve estudio del trabajo relacionado con la adquisición de datos en la Web. En la sección 4.2 se presenta la arquitectura diseñada para el cosechador de datos. La sección 4.3 está dedicada a la arquitectura de la araña web. El repositorio de documentos, el método *push* y las operaciones sobre los datos recolectados son presentados en la sección 4.4. La información necesaria para el manejo de la información recolectada (*metadatos*) son explicados en la sección 4.5. Para finalizar, la sección 4.6 muestra un breve sumario del Capítulo.

4.1. Trabajo Relacionado

El método *pull* es utilizado para la obtención de datos en un sistema, recogiendo los datos de su fuente original. Este es el método principal de recolección de datos para la mayoría de las arquitecturas existentes: Google [Brin98], Citeseer [Bollacker98], Glimpse [Manber94], WebBase [Cho04], OmniPaper [Technologies05a], entre muchos otros [Chakrabarti99, Aggarwal01, Zhuang05, Bergmark02, Hafri04, Melnik01]. Los agentes encargados de recolectar datos por medio del método *pull* son llamados generalmente *web crawlers*, *arañas web*, o *robots web*. Una araña web recorre Internet gracias a un conjunto inicial de URIs [Berners-Lee05], llamados semillas, de los cuales se obtiene el contenido de las páginas recolectadas extrayendo nuevos URIs para ser visitados. Existe un tipo diferente de recolector *pull* llamado cosechador [Bowman95, Florescu98, Hardy96]. Un cosechador únicamente recolecta archivos, los cuales se encuentran en una *cola* de cosechamiento. La cola de cosechamiento puede ser una cola de prioridad o cola *FIFO*.

El método *push* es utilizado para la obtención de datos donde se desea que el crea-

dor de datos tenga control sobre el tiempo en el cual sus datos deben ser recibidos. En push, las fuentes de datos envían directamente los datos al recolector de información. Por ejemplo, el autor de un artículo puede publicarlo en cuanto termine de escribirlo y no debe esperar a que un agente recolector, como Citeseer [Bollacker98] o ScholarGoogle [Google-Inc], obtenga la información. OmniPaper [Technologies05a] planea implementar el método push en su versión final, mientras que Google recientemente ha implementado herramientas que permiten emular push sin modificar su arquitectura interna. En el pasado, el método push era poco utilizado en Sistemas de Recuperación de Información, pero en la actualizada ha ganado mayor popularidad.

La recolección por medio de cosechamiento y *crawling* focalizado (comenzando por [Bowman95, Chakrabarti99]), lleva al ahorro de ancho de banda, tiempo, y recursos computacionales. Mucho trabajo en el campo de crawling focalizado se ha producido, por ejemplo [Hafri04, Zhuang05, Chakrabarti99, Chakrabarti02, Bollacker98, Pandey05, Bergmark02].

El repositorio de documentos es fundamental para el escalamiento de un sistema de adquisición de datos. Los más notables repositorios han sido creados durante la historia del desarrollo de Google [Brin98], su posterior comercialización y mejoramiento de su sistema de archivos [Ghemawat03] y la rama académica WebBase [Cho04].

El módulo de adquisición de datos creado a partir de este trabajo, cuenta con un cosechador de documentos (i.e. descarga archivos explícitamente requeridos por el sistema), un crawler y un repositorio de documentos. El crawler no soporta crawling enfocado, pero la arquitectura y la implementación soportan la conexión de lógica que permite realizarlo. También soporta el adquisición con el método push, por medio del repositorio de documentos.

4.2. Arquitectura del Cosechador

La arquitectura de un cosechador individual se muestra en la Figura 4.1, donde se muestra los componentes abstractos, la interconexión de sus partes y la forma de conectar con sus posibles clientes.

Un cosechador puede ser visto como una tupla de 6 elementos (L, P, T, C, F, R)

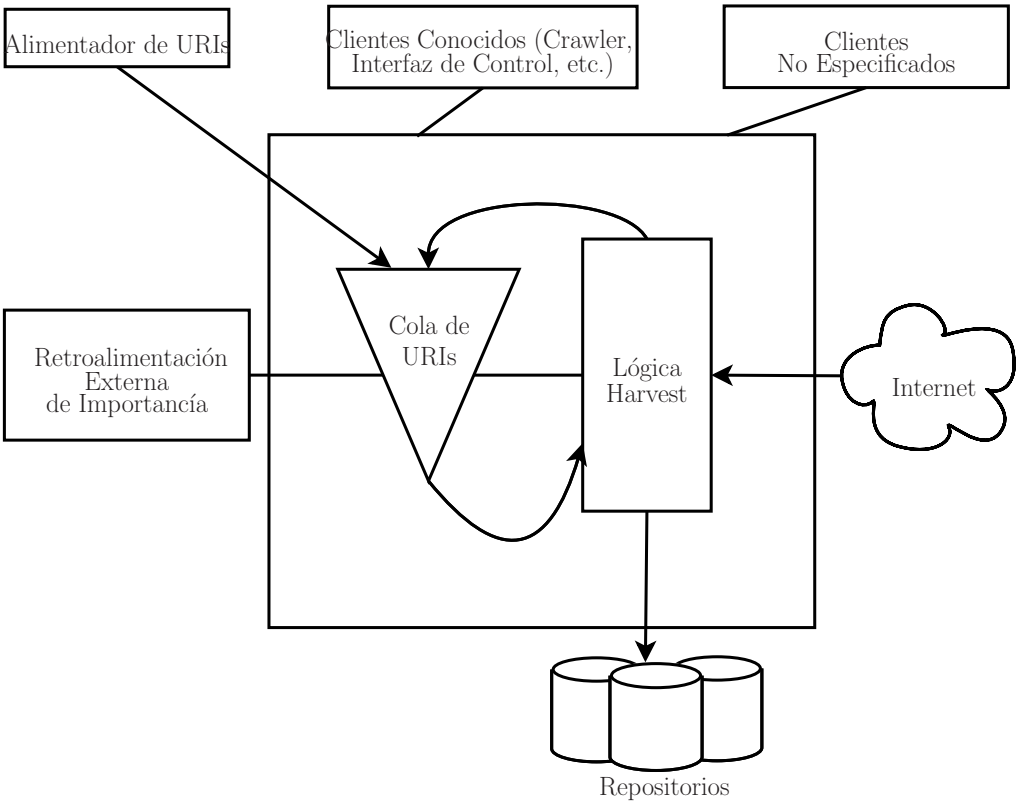


Figura 4.1: Arquitectura interna de un cosechador y sus clientes

donde L es la lógica con la cual el cosechador se maneja representada como un conjunto de reglas, procedimientos y funciones, P es la tupla de los parámetros para la lógica L, T es la cola de URIs a cosechar, pudiendo ser una cola secuencial o de prioridad (cuya prioridad puede ser dinámica o estática). C es el conjunto de clientes que de alguna u otra manera interactúan, controlan o modifican alguno de los elementos de la tupla. F es la fuente de archivos sobre la que se alimenta (típicamente Internet) y R es el repositorio donde los archivos son guardados después de ser descargados.

La lógica L es descrita como un conjunto de reglas, procedimientos y funciones que unidos permiten al cosechador tomar la decisión de realizar la cosecha de un URI, un conjunto de URIs o de un sitio completo. Además, una serie de parámetros son utilizados para refinar y ajustar la lógica del programa, denotados como P. Estos parámetros, y el paso de argumentos adicionales a la lógica, permiten controlar el comportamiento del cosechador. Los parámetros incluyen información sobre cuando se debe comenzar a cosechar, cada cuando se debe cosechar, número de hilos o descargas simultaneas, información sobre conexiones, entre otros.

La cola T es la parte medular del cosechador, la cola mantiene una lista de URIs bajo un orden particular, de manera que la lista solo puede ser accedida en una forma en particular dada por una prioridad especificada. La prioridad puede estar especificada por la lógica L del cosechador, una entidad externa E tal que $E \in C$ o una combinación de ambas. La prioridad determina el orden en el cual las URIs serán descargadas. Una prioridad secuencial está determinada por el instante de tiempo en el cual la URI fue introducida en T y la cola se convierte en una estructura FIFO si la prioridad esta dada a lo primero que entró .

El conjunto de clientes conectados al cosechador es llamado C. Los clientes pueden tomar roles muy variados como obtener URIs, modificar la pila o controlar la ejecución de acciones. Es difícil enumerar los posibles Clientes del cosechador, ya que se proporciona una API muy sencilla y Servicios Web [Kreger01] que factilitan la programación de extensiones y clientes, además del soporte de código móvil (ver Capítulo 3).

En la Figura 4.2 distinguimos la Internet como fuente de archivos para el cosechador, los cosechadores y una serie de clientes de los cosechadores. Los principales clientes de

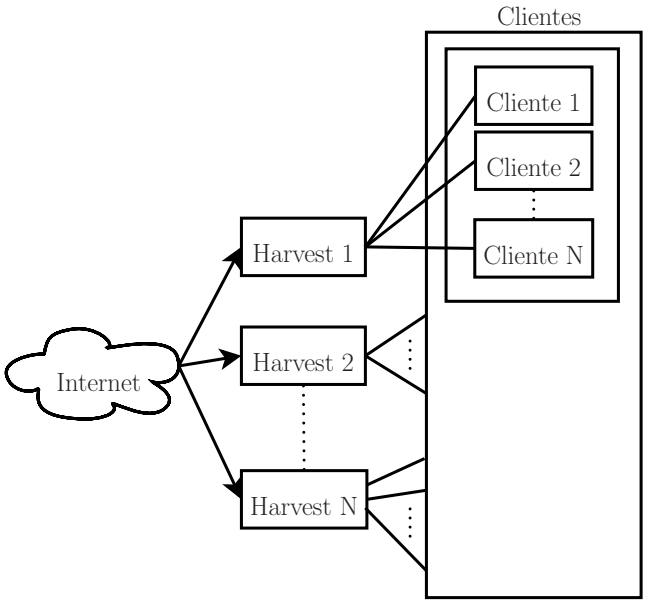


Figura 4.2: Arquitectura abstracta de un conjunto de cosechadores alimentándose de la Internet y sirviendo a una serie de clientes

los cosechadores son:

Interfaz de control. La interfaz de control es capaz de modificar la cola de URIs añadiendo, borrando o modificando URIs de sitios a cosechar. También puede modificar la prioridad de modificación. Además es capaz de cambiar parámetros de funcionamiento o dar instrucciones para comenzar cosechas o sincronizar la cola al disco.

La interfaz de control puede ser vía Web, línea de comandos o una aplicación visual en algún lenguaje de programación con capacidad de comunicación vía servicios web.

Crawler. Un crawler desde nuestro punto de vista es un extractor de posibles saltos o ligas *alcanzables* desde un determinado URI, determinando mediante un conjunto de reglas que archivos deben ser descargados. Estas reglas determinan la eficacia de un crawler y dependiendo de su eficacia es posible obtener un alto rendimiento optimizando recursos. Por ejemplo, un crawler puede ser capaz de obtener las archivos de un mismo sitio, sin salirse de él, sin importar los tipos de archivos recolectados o puede obtener únicamente las ligas relacionadas a un tópico específico o ponderar qué tan importante es obtener una liga en comparación a otras.

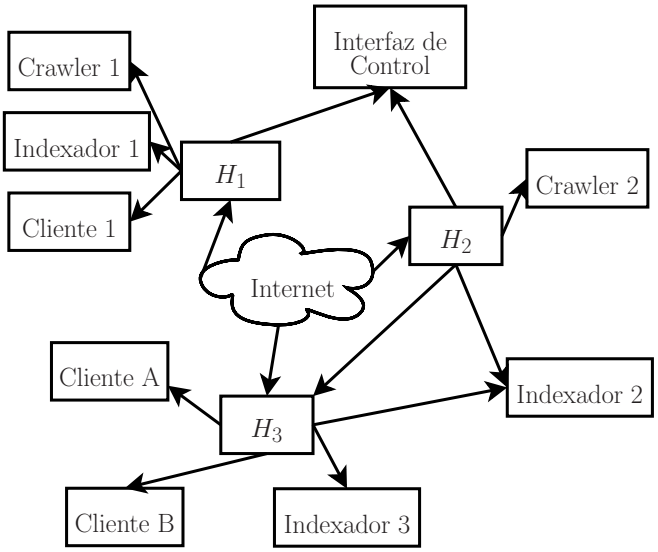


Figura 4.3: Ejemplo de conexión de un conjunto de Cosechadores H_i con sus clientes

El indexador. El indexador puede recibir la orden de indexar cuando el cosechador a descargado documentos. Útil para mantener índices *siempre al día*.

El cosechador por tanto puede tener una serie de clientes que modifiquen dramáticamente su desempeño y funcionalidad, siendo el caso más simple donde un cosechador trabaja por si solo. Un caso más complejo se presenta cuando más de un cliente controla al cosechador. Un conjunto de cosechadores y un conjunto de clientes interconectados como un ejemplo hipotético de las capacidades del cosechador se muestra en Figura 4.3, interconexiones a segundo nivel (no incluyendo a un cosechador) no son mostradas.

El sistema de recolección de datos es capaz de operar en un sistema distribuido, dividiendo la carga entre varios nodos en la red. La carga puede ser:

- Ancho de banda necesario para realizar las descargas y búsquedas.
- Tiempo de proceso de texto.
- Espacio de almacenamiento en disco.
- Espacio para realización de tareas en memoria primaria [Cho04, Brin98, Barroso03].
- Costo de hardware.

Existen varios criterios de división de la carga, por ejemplo *localización geográfica*, *aleatorias*, o por *contenido*. La localización geográfica puede ser utilizada para optimizar accesos por localización real o efectiva a los datos a recolectar, por ejemplo, es posible obtener un grafo ponderado a costos en tiempo o dinero debido a la transmisión, asignando recolectores a cada una de las localidades óptimas según el costo de adquisición, esto puede lograrse fácilmente mediante algoritmos de *caminos más cortos* [Cormen01]. Las políticas de balance aleatorio han probado ser de alto desempeño [Cho04, Brin98]. La política de balance aleatoria más popular es la división de la colección por medio de una función *hash* [Graham89] sobre algún identificador único de los documentos como el URI de los documentos [Cho04, Brin98]. La función *hash* devuelve un identificador numérico que puede ser procesado mediante una función deterministica para ser asignado a un conjunto particular de nodos. La función deterministica puede ser el *módulo*, y cada computadora puede tener asignado un número determinado de módulos. Como alternativa a una función deterministica de procesamiento también se puede crear una función *hash* con alto índice de *colisión* [Graham89], y a y la tabla hash es análoga al conjunto de computadoras.

También es posible distribuir de una manera totalmente un documento a algún nodo, sin importar ningún parametro. Ésta configuración presenta tolerancia a fallas [Kan01, Ripeanu01], a costa de poca eficiencia en consultas [Androutsellis-Theotokis04, Bawa03, Klampanos04, Baquero03].

La selección del conjunto de adquisidores de datos también realizarse utilizando información acerca del documento (apuntadores hacia el documento o meta-información). Esto con la finalidad de "adivinar" los tópicos del documento y agrupar los documentos relacionados. Si la segmentación de los documentos depende del tópico [Bowman95], aumenta la cohesión entre documentos de un tema determinado. Esta solución aumenta la calidad de los resultados y la velocidad de respuesta [Bawa03, Khoussainov04].

4.3. Arquitectura de la Araña Web

Un crawler puede ser definido como un cliente C_C de un cosechador tal qué procesa un documento asociado a un URI u_a y obtiene un conjunto de URIs U_C tal que $\forall_{u \in U_C}(u_a, u)$.

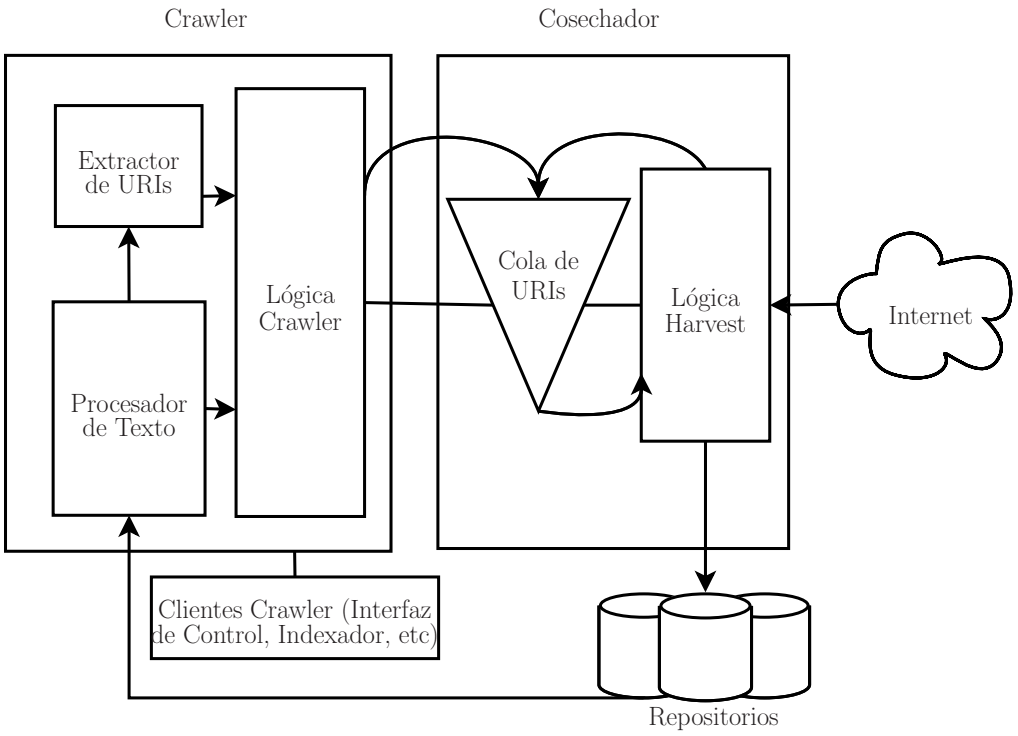


Figura 4.4: Arquitectura Abstracta del Crawler

La obtención de U_C se realiza mediante un procesador de texto con un conjunto de reglas gramáticales que permiten extraer el texto y el conjunto U_C. La eficiencia de un crawler esta ligada a su procesador de texto y a su capacidad para obtener y discernir U_C, así como obtener U_R, donde $U_R \subseteq U_C$. U_R representa el conjunto de URIs relevantes al enfoque del crawler. Adicionalmente, la forma para distinguir entre un documento más importante que otro para modificar la prioridad de un URI en la cola del cosechador es un tema muy importante para las máquinas de búsqueda en la web [Pandey05, Cho04, Brin98].

Se puede definir un crawler como el mostrado en la Figura 4.4 como la tupla (T, P, E, L, H, C), donde T es el extractor de texto, transformador de un formato dado a una normalización de un formato conocido. P es un conjunto de parámetros utilizados por la lógica del programa y contiene los parámetros para cada sitio, E es el extractor de URIs. L es la lógica del crawler cuya función principal es decidir que es recolectado primero y controlar el cosechador por medio de la conexión al cosechador H. C es un conjunto de clientes

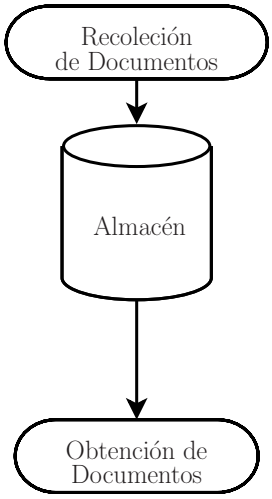


Figura 4.5: Posición lógica del almacén de documentos en el Sistema de Recuperación de Información

que pueden modificar o ajustar cualquiera de los elementos de la tupla, enriqueciendo el funcionamiento del Crawler. Por ejemplo, un dispositivo de resolución de consultas puede ser consultado para ponderar la importancia de un texto descriptor del documento conforme a la colección pueden reconocer, un *categorizador* puede calcular que tan relacionado se encuentra el texto con las categorías existentes.

La lógica del crawler incluye la forma en la cual serán visitados los URIs, tomando en cuenta el grafo formado por (V, E) donde V es el conjunto de URIs y E el conjunto de ligas entre URIs. Entonces la visita debe hacerse en BFS y no en DFS. BFS proporciona un mayor control sobre la cantidad de memoria consumida, paradójicamente a la visita tradicional de BFS que utiliza una gran cantidad de memoria. La principal causa de usar BFS es que conceptualmente mantiene primero en visitas los URIs más relacionados con el URI semilla. Un caso de especial interés es la *Web Profunda*, que son páginas web que se generan dinámicamente y son muy extensas o infinitas, para estos casos las condiciones de paro son complejas o restrictivas.

4.4. Repositorio de Documentos

El repositorio es el almacén donde los documentos son guardados (Figura 4.5). Además de asegurar la persistencia de los datos a lo largo del tiempo, es un conector natural entre módulos que acceden directamente a los documentos.

Google [Ghemawat03] y WebBase [Cho04] están optimizados para el acceso *ráfaga*. El acceso ráfaga es utilizado para el envío y acceso eficiente de grandes bloques de información. Google y WebBase también usan acceso aleatorio, en menor medida.

A diferencia de Google, el repositorio de documentos creado en éste trabajo no se encuentra optimizado para acceso ráfaga por bloques, únicamente por archivo completo, esto lleva a cierto tipo de pago en eficiencia, a costa de simplicidad en la implementación. En un futuro, se planea optimizar el repositorio para accesos ráfaga por bloques. La compresión de los archivos es realizada utilizando gzip, basado en LZ77 [Jacob Ziv77], gracias a los trabajos y consideraciones encontradas por Ghemawat et al [Ghemawat03].

El sistema de archivos de Google y WebBase contiene un índice centralizado que aumenta el desempeño, a costa de un punto de falla (el punto de falla se encuentra respaldado y con altas medidas de seguridad). La decisión en ambos sistemas de archivos está sostenida en que el conjunto de computadoras se encuentra en un ambiente bien controlado.

A diferencia de los sistemas de archivos de Google y WebBase, nuestro repositorio de documentos está integrado a los demás módulos utilizando Servicios Web [Kreger01] mediante SPyRO (ver Capítulo 3). La íntima integración con estos servicios nos proporciona ventajas sobre sistemas de archivos anteriores, como las siguientes:

- Repositorio distribuido.
- Interfaz para el método *push*.
- Implementación sencilla de interfaces de control.
- Conexión a cosechadores remotos.
- En general, la conexión de cualquier tipo de cliente para controlar o modificar el repositorio.

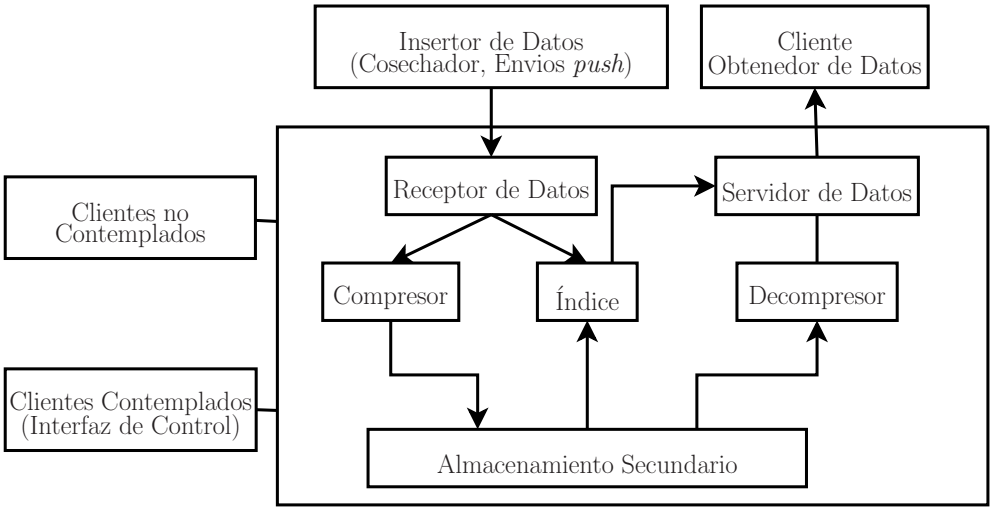


Figura 4.6: Arquitectura del Repositorio

La arquitectura del repositorio se muestra en la Figura 4.6. La Figura muestra la interconexión de los componentes del repositorio, así como la interacción entre ellos. El *insertor de datos* es un cliente especializado como el Cosechador o los envíos de documentos por medio del método *push*, donde la principal acción del cliente es insertar documentos al repositorio. Los documentos son recogidos por el *receptor de datos*. Éste se encarga de guardar el documento en memoria secundaria si es necesario.

El comportamiento es ilustrado a mayor detalle en el Algoritmo 1.

Un *cliente obtenedor* de datos crea peticiones para recuperar archivos del repositorio. Los archivos pueden ser entregados en forma comprimida o descomprimida. El proceso se muestra en el Algoritmo 2.

4.5. Metadatos de los Documentos

Los metadatos son datos que describen muchas de las características administrativas de los documentos. Existe un conjunto de metadatos asociados a cada documento. Los metadatos son guardados como parte del índice de entradas en el repositorio, para el acceso inmediato después de consultar el índice. El índice es implementado mediante una base de datos de Berkeley, basada en B-Tree [Kim01, Comer79, Batory81, Johnson93]. La

Algoritmo 1 Inserción de documentos al repositorio

1. **Entradas**.

   Documento: Datos del documento.

   Metadatos: Información descriptiva e informativa del documento.

   URI: Nombre identificador del archivo en el repositorio.

2. **Salidas**. Ninguna salida.

3. Se busca *URI* dentro del índice para comprobar si el documento existe anteriormente:

   a) Si existe y el documento en el repositorio es más viejo, se remplaza el documento existente y los metadatos.

   b) Si no existe se inserta URI en el índice, Se guarda *Documento* en el almacenamiento secundario y se obtienen los metadatos.

Algoritmo 2 Obtención de documentos del repositorio

1. **Entradas**.

   URI. Nombre identificador del archivo en el repositorio.

2. **Salidas**.

   Flujo de archivo.

3. Se revisa el índice para comprobar si el documento existe:

   a) Si existe, se recupera el documento, se descomprime y se envía el archivo a través de un flujo de datos.

   b) Si no existe, se regresa una excepción (posiblemente remota).

Tabla 4.1: Conjunto de Metadatos

Metadatos	**Tipo**
Fecha de inserción	Fecha
Fecha de última revisión	Fecha
Número de actualizaciones	Fecha
Número de Errores 404	Número Entero
Número de Errores Totales	Número Entero
Posición en Disco	Índice Numérico
Longitud	Entero Numérico
Tipo MIME	Cadena MIME

llave del índice es el URI del documento y el valor de la base de datos es una estructura con el conjunto de metadatos.

El conjunto de metadatos utilizados se muestra en la Tabla 4.1. A continuación se describe la finalidad de cada campo:

Fecha de inserción. Índica cuando fue la primera inserción.

Fecha de última revisión. La fecha y tiempo en la cual ocurrió correctamente la última actualización.

Número de actualizaciones. Índica cuantas actualizaciones ha sufrido el documento desde la fecha de inserción.

Número de Errores 404. Número de errores indicando la no existencia del documento.

Número de Errores Totales. Número de errores genéricos o no fatales en la recuperación del documento.

Posición en Disco. Posición o índice del documento en el archivo de repositorio, redundante al índice primer índice.

Longitud. La longitud total del documento.

Tipo MIME. Qué tipo de documento era originalmente.

4.6. Sumario

A lo largo de éste Capítulo se discutió el módulo de adquisición de datos. Se dio una introducción a los sistemas existentes y sus módulos de adquisición de datos. Por la forma en la cual obtienen datos, los métodos se dividen en *push* (envío dirigido por el usuario) y *pull* (recolección por medio de agentes ya sea por localización directa o búsqueda-navegación). El módulo de adquisición de datos de ésta tesis, cuenta con un cosechador y un crawler que usan el método *pull*, mientras que el repositorio de documentos permite la adquisición por medio de *push*, el cual es un método cada vez más popular en la adquisición de datos por medio de máquinas de búsqueda en la Internet. El método ha ido cobrando fuerza debido que proporciona una herramienta guiada por los usuarios creando un SRI con versiones actualizadas de los documentos. La razón de la existencia de dos tipos diferentes de adquisidores pull es que ambos realizan dos tipos diferentes de adquisición: el cosechador es usado para la recolección de datos en localidades fijas, mientras que el crawler navega a través de hipervinculos buscando documentos de interés.

Se mostró la arquitectura creada con el fin de alcanzar los objetivos propuestos, la cual consiste en una serie de conectores e interfaces que permiten la conexión "en vivo" de componentes. Estos componentes pueden ser internos o externos, también se pueden ver como controladores del comportamiento de los adquisidores o clientes.

El Capítulo finaliza con una lista de los metadatos usados para identificar y caracterizar los documentos, ya que esta información es de vital importancia para alguien que desee extender el sistema.

Es importante destacar que los componentes están basados en código móvil y Servicios Web, por lo que usuarios externos pueden hacer programas que usen el sistema sin hacer uso de interfaces de usuario, esto es, permitiendo la automatización de procesos. Además, también es posible distribuir cada uno de los módulos a través de una red heterogénea de computadoras y son capaces de funcionar de manera transparentes. También es necesario enfatizar que los módulos distribuidos soportan segmentación y son capaces de seguir prestando servicios. Ligada a la segmentación, existe una posible perdida de datos, que puede ser minimizada a través de factores de replicación de contenido.

Capítulo 5

Indexamiento y Consulta

El indexamiento de documentos es una práctica que data desde los inicios de la escritura, donde la necesidad de encontrar pasajes en un documento propició el surgimiento de métodos que facilitarán el acceso a ciertas partes identificadas en el texto. En la antigüedad, varios años eran necesarios para completar el indexamiento de un libro extenso, como la biblia o el Quijote. Sin embargo, en la actualidad los sistemas indexadores pueden procesar miles de documentos por segundo gracias a las técnicas modernas de recuperación de información. Un estudio profundo sobre los comienzos de la Recuperación de Información se encuentra disponible gracias a la Sociedad Americana de Indexadores [ofIndexers05].

En términos generales, el indexamiento de un documento consiste en la identificación de los términos del documento y la posterior construcción de una tabla, llamada índice, que indique las posiciones en el documento donde aparecen los términos indexados.

Por otra parte, el indexado de una colección de documentos consiste en indexar el conjunto de documentos. Esto permite localizar documentos y pasajes en los documentos, donde ocurran los términos indexados.

El proceso de construcción de los índices es de gran interés para las empresas dedicadas a estos servicios, ya que su efectividad y eficiencia está ligada directamente con la capacidad de procesamiento de información, así como con la cantidad de información coleccionada. De igual forma, la capacidad de respuesta en las consultas determina el número de clientes que pueden ser atendidos, por consecuencia es un factor prioritario para las

empresas. La relevancia de los resultados entregados por una consulta es un tema muy importante de investigación ya que implica la efectividad del Sistema de Recuperación de Información.

Muchas de las compañías que prestan servicios a través de Internet tienen como producto principal (o al menos prioritario) el indexamiento y consulta de información. Por ejemplo *Google*[1], *Yahoo!*[2], *Altavista*[3], *Ebay*[4], *Amazon*[5], entre muchas más.

El trabajo relacionado a éste módulo está íntimamente ligado al presentado en el Capítulo 1, debido a esto, el presente Capítulo ómite el trabajo relacionado. Además de las consideraciones y principios expuestos en los Capítulos 1 y 2.

El presente Capítulo está dedicado al módulo de indexamiento y consulta. La sección 5.1 se enfoca en la arquitectura del módulo, que incluye el creador de índices y el resolvedor de consultas. La sección 5.2, muestra el proceso de indexamiento y de resolución de consultas llevado a cabo. La sección 5.3 está dedicada a presentar las consideraciones llevadas a cabo en la implementación del módulo de indexamiento y consulta. Las características del módulo se explican en la sección 5.4, y para finalizar, la sección 5.5 presenta un resumen del Capítulo.

5.1. Arquitectura

Un indexador puede ser visto como un tupla de 6 elementos (I, Q, M, E, C, F), donde I es la técnica de creación de índices utilizada, Q es el resolvedor de consultas. M es el módelo de Recuperación de Información utilizado, mientras que E es el conjunto de modificadores adicionales al módelo. C es el conjunto de clientes que modifican, interactúan o controlan el comportamiento del indexador. Por último, F es la fuente de alimentación

[1] Google Inc. es una empresa dedicada a encontrar recursos en la *Word Wide Web*. Se encuentra dentro de la WWW en http://www.google.com

[2] Yahoo! (Yet Another Hierarchically Organized Oracle) en sus inicios es un enorme índice de sitios en la web, organizado por temas. En la actualidad es un sitio que presta servicios de correo electrónico, máquina de búsqueda, temarios, juegos, tiendas, y varios servicios adicionales

[3] Altavista fue una de las primeras máquinas de búsqueda en Internet y a la fecha sigue siendo una de las más importantes compañías dedicadas a la Recuperación de Información en Internet. Altavista se encuentra disponible en http://www.altavista.com

[4] http://www.ebay.com

[5] http://www.amazon.com

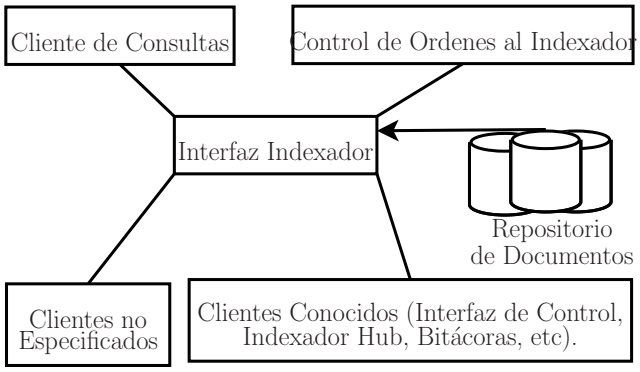


Figura 5.1: Interfaz de un Indexador

de documentos.

La técnica I es utilizada para la creación de los índices de documentos, y representa un factor muy importante en el desempeño del indexamiento. Algunas técnicas de indexamiento son la creación de índices en memoría principal, índices segmentados, uso de tablas hash, tries, btrees, etcétera. Para mayor información una serie de excelentes referencias son [Baeza-Yates99, Turtle92, Hiemstra00, Witten99]. El resolvedor de consultas Q debe comprender los formatos de los índices generados por I y encontrar documentos valorados mediante el modelo de RI M con las mejoras E.

El conjunto de clientes C conectados al indexador pueden tener varios roles, como clasificadores, interfaces de consulta, interfaces de configuración, filtros u otros indexadores, y muchos otros posibles clientes debido a las posibles extensiones utilizando los servicios web y el API provisto.

El módulo del indexador está formado por una *interfaz genérica* (Figura 5.1) y dos especializaciones de la interfaz: *Indexador Hub* (Figura 5.2), *Indexador Final* (Figura 5.3).

La interfaz de *indexador genérica* describe la manera en que un indexador se comporta e interacciona con su entorno. La Figura 5.1 muestra la interfaz de un indexador. El *cliente de consultas* es un cliente que realiza peticiones de consultas y la interfaz de indexador responde con los resultados relacionados a las consultas. El *control de ordenes al indexador*, indica al indexador las tareas que debe realizar (e.g. indexar una serie de documentos del *repositorio de documentos* o indexar por el método *push*). Éste último, también es parte del

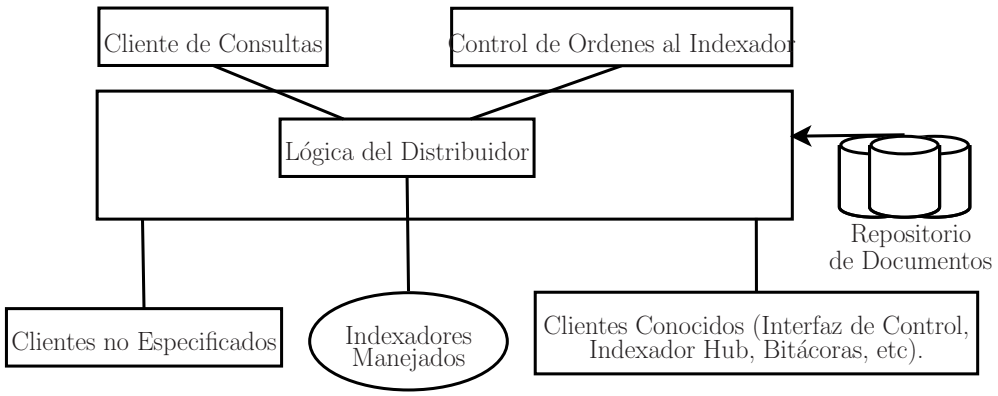


Figura 5.2: Arquitectura del Indexador Hub

conjunto de clientes aunque por su importancía y papel dentro de la arquitectura, tiene una atención especial.

Los *clientes* o submódulos se conectan para mejorar, incrementar o modificar las características y comportamientos del indexador. En la Figura 5.1 se distinguen dos tipos de clientes:

- **Clientes conocidos.** Estos clientes tienen un propósito específico y bien conocido. Ejemplos de clientes conocidos son interfaces de control, interfaces de consulta, indexadores, generadores de bitácoras, web crawlers, cosechadores, etc.
- **Clientes no especificados.** Son clientes que por el momento no han sido creados, o su interacción no ha sido definida.

La Figura 5.2 muestra la especialización de la interfaz para un *indexador hub*. Un indexador hub es capaz de mostrar un conjunto de indexadores como uno solo. El indexador hub recibe peticiones de consultas y las resuelve mediante un conjunto de *indexadores manejados*, los resultados son unidos y devueltos al cliente que realiza consultas. Así mismo, es capaz de recibir y distribuir hacia sus *indexadores controlados* las peticiones de indexamiento y control recibidas del *control de ordenes al indexador*. La *lógica del distribuidor* es la parte esencial del indexador hub, ya que determina la eficiencia de las consultas y la calidad de los resultados. Los clientes conocidos y no especificados se comportan de igual forma que en la interfaz de indexador generalizada.

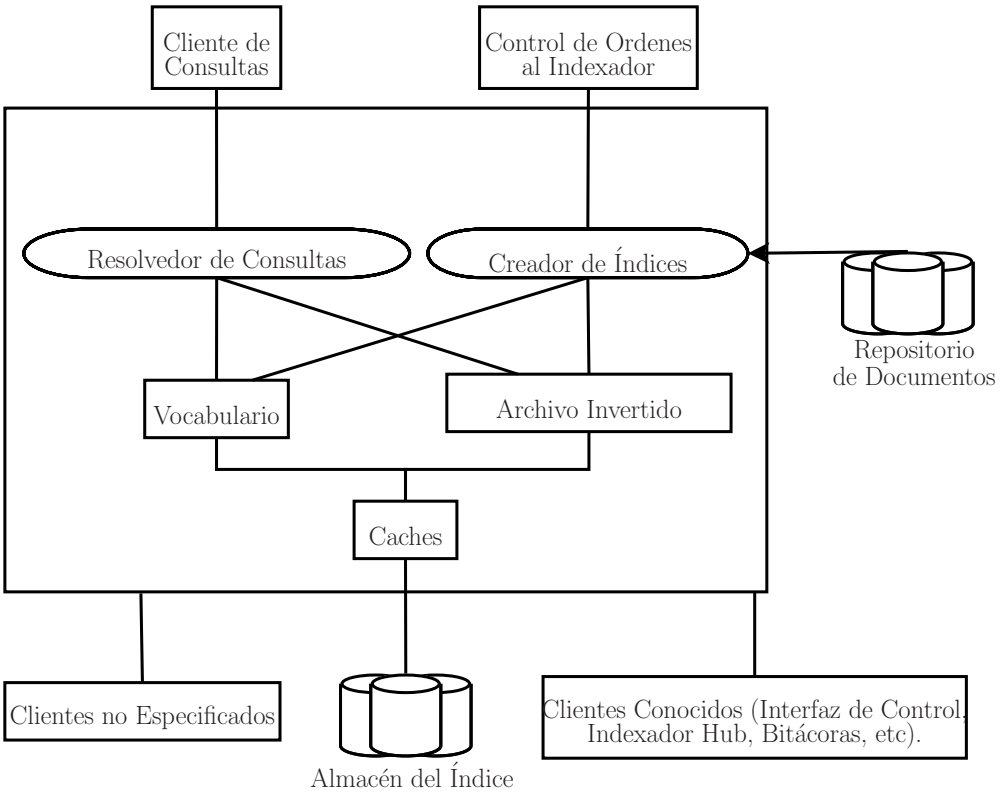


Figura 5.3: Arquitectura del Indexador Final

Un *indexador final* se muestra en la Figura 5.3. El indexador final tiene como propósito ser una rama terminal en un grafo de indexadores, es decir, un indexador que indexe los documentos por sí mismo. Se encarga de resolver consultas, creación de archivos invertidos, etcétera. En la Figura 5.3 se muestra un *resolvedor de consultas* que se encarga de realizar las búsquedas en el índice y los archivos invertidos, y determinar k documentos relevantes para la consulta dada relativos a una función de similitud particular. El *creador de índices* recibe peticiones de indexamiento de documentos. Los documentos pueden ser obtenidos a través del *repositorio de documentos* o pueden estar incluidos en la misma petición. El creador de índices modifica el vocabulario, los archivos invertidos, y en general las estadísticas que cambien por la inserción de documentos nuevos.

Tanto el *vocabulario* como los *archivos invertidos* pueden estar basados en caches de discos acelerando el acceso al almacén del índice. En general los accesos a disco deben estar acelerados por medio de caches de alto desempeño.

5.2. Proceso de Indexamiento y Consulta

La Figura 5.4 ilustra el proceso de indexamiento. Un cliente crea una petición de indexamiento. El indexador toma el documento, lo analiza para obtener sus propiedades dependientes del formato y estadísticas. Si es necesario, se actualiza el vocabulario y las estadísticas globales de la colección. A continuación se añade el nuevo documento a los archivos invertidos.

El resolvedor de consultas utiliza la misma información y co-existe en el mismo espacio de memoria que el indexador. La Figura 5.5 ilustra los pasos del resolvedor de consultas. El resolvedor de consultas procesa la consulta, realiza la búsqueda de los documentos que bajo una medida de similitud resulten de importancia (i.e. aplicando esquemas $TF \times IDF$) para la búsqueda. Por último, se ordenan los resultados por importancia y se devuelven k documentos, comenzando desde el n-ésimo elemento.

La comunicación entre indexadores es realizada utilizando SPyRO y HTTP. SPyRO es utilizada para el manejo y acceso de objetos remotos y HTTP para el intercambio masivo de datos.

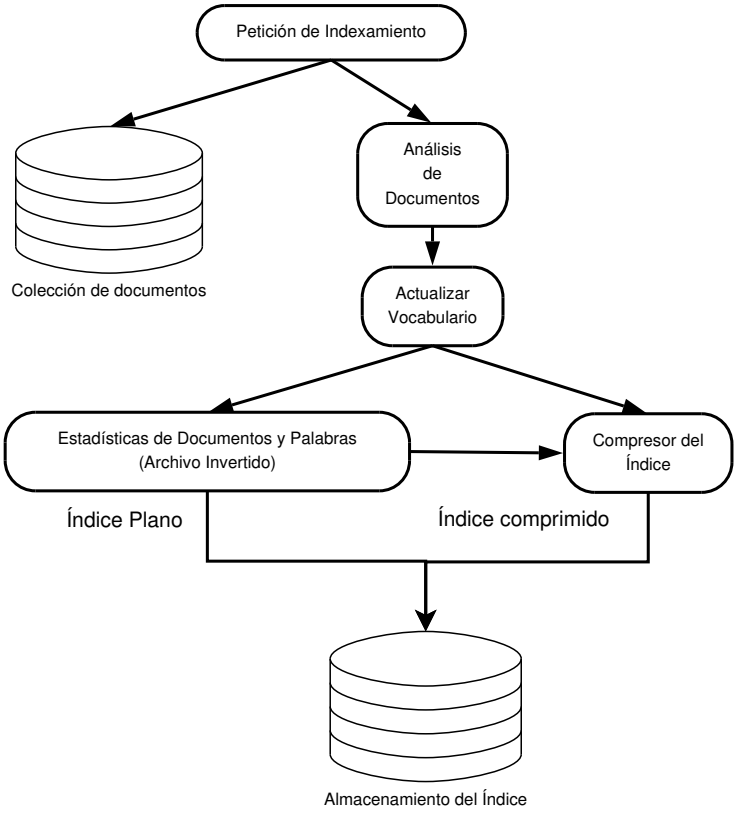


Figura 5.4: Proceso del indexador

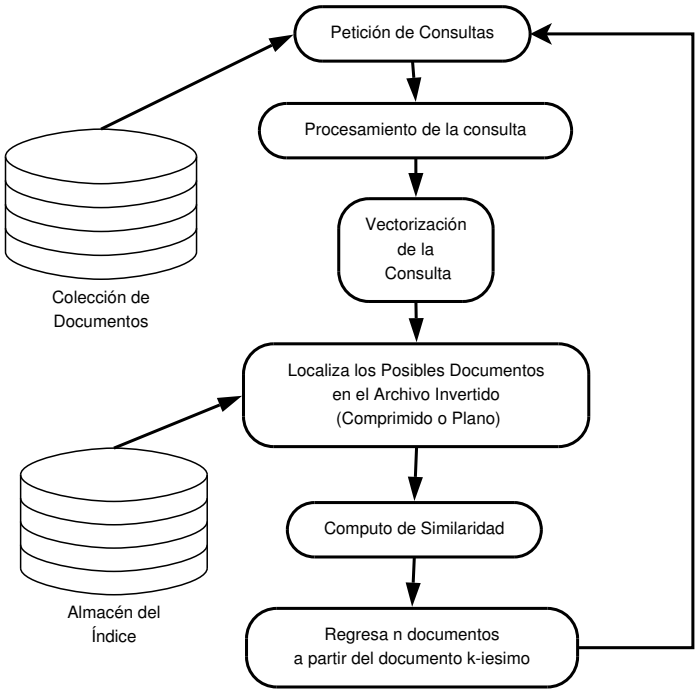


Figura 5.5: Proceso del resolvedor de consultas

5.3. Consideraciones de Implementación

El módulo de indexamiento consta de un indexador de texto completo y un resolvedor de consultas. Ambos son subsistemas de propósito general, modulares y distribuidos. Tanto el resolvedor de consultas como el indexador comparten el mismo subsistema. Ambos pueden coexistir dentro del mismo proceso en una computadora. Ésta cualidad permite índices y archivos invertidos *siempre actualizados*. En contraste, el resolvedor de consultas es espacial y temporalmente poco eficiente comparado con técnicas de compresión de índices [Witten99], como los arreglos de sufijos y compresión por diferencias relativas. Para sistemas con cargas altas de peticiones o colecciones grandes es necesario escalar el sistema o utilizar compresión.

Para colecciones grandes o muy visitadas es posible utilizar compresión, para acelerar las respuestas. Mientras que para colecciones pequeñas (poco visitadas o pocos documentos) es factible utilizar índices *siempre al día*. El submódulo resolvedor de consultas usa el mismo API que el submódulo indexador.

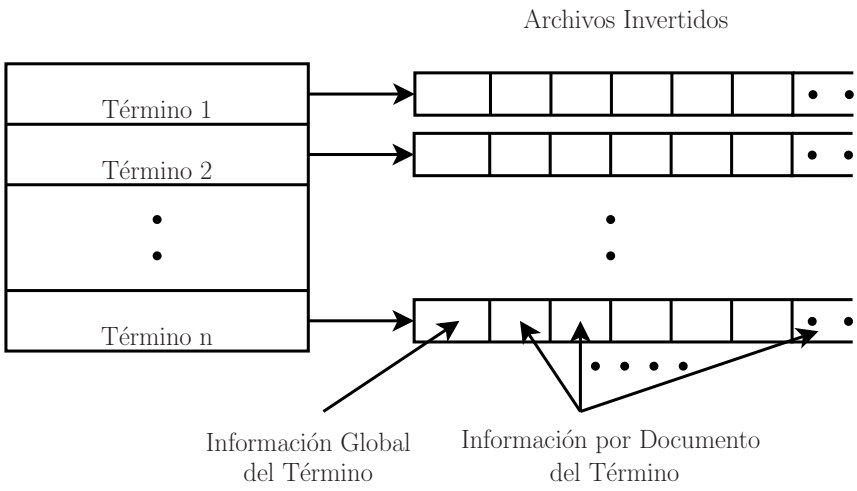


Figura 5.6: Archivo Invertido

Gracias a la modularidad del diseño, la arquitectura y la implementación soportan índices y archivos invertidos comprimidos, aunque no han sido implementados.

Un archivo invertido (ver Figura 5.6) se compone de la misma manera que una lista invertida, pero su implementación está optimizada para memoria secundaria.

En la implementación, el indexador de documentos construye un índice de texto completo estándar [Baeza-Yates99, Witten99] basado en el texto de los documentos proporcionados. Se utilizan archivos invertidos para la información de estadísticas de los documentos y palabras. El vocabulario utiliza un *trie* basado en arreglos [Baeza-Yates99] (ver Figura 5.7). El indexador ésta basado en tries para el manejo eficiente de las consultas, ya que la búsqueda de cadenas en un trie está acotada por $O(n)$, donde n es la longitud de la palabra a buscar. En la búsqueda por medio de tries no importa el tamaño de la colección o del vocabulario, por lo tanto un trie es ideal para búsquedas de texto en grandes colecciones y vocabularios grandes. También es posible efectuar búsquedas utilizando expresiones regulares por medio de un trie de manera eficiente[6]. Adicionalmente, existe una compresión natural de las palabras al utilizar un trie, es una característica notable cuando el vocabulario es muy denso.

[6]No todas las expresiones regulares pueden ser realizadas de manera rápida en un trie, pero a excepción de las expresiones $*xy$, el espacio de búsqueda siempre es reducido de manera eficiente. Si las búsquedas $*xy$ son necesarias, es posible indexar palabras invertidas con el uso de un vocabulario adicional.

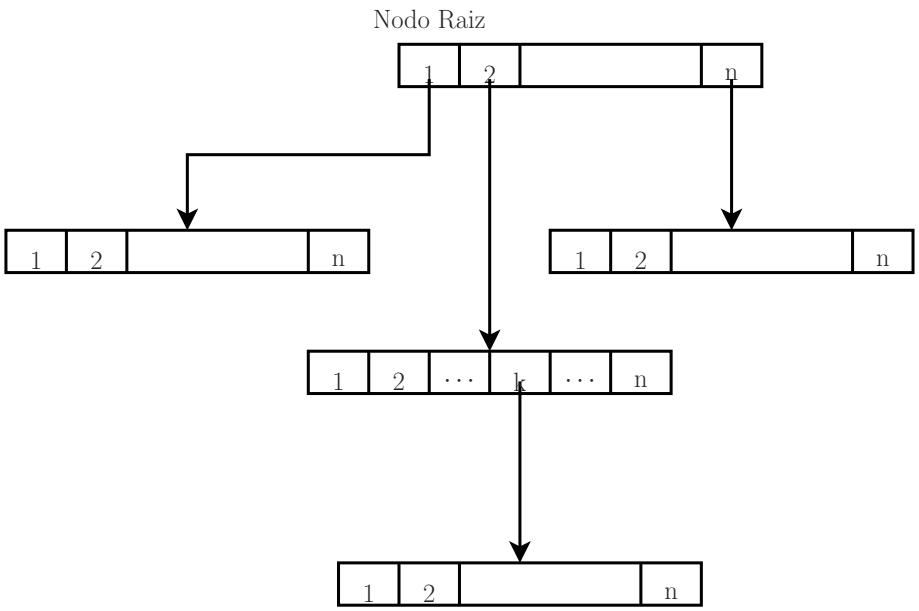


Figura 5.7: Trie basado en arreglos

Los nodos del trie contienen $1+|\Sigma|$ localidades. La primera localidad está destinada a un marcador de terminación de palabra y las siguientes $|\Sigma|$ localidades son utilizadas por localidades del alfabeto. En el caso de la implementación basada en arreglos es necesario reducir el número de nodos terminales para evitar un alto índice de vacuidad en los nodos. Las localidades destinadas al alfabeto están divididas en tres tipos:

- **Nodo Índice.** Un nodo que indica donde se encuentra el siguiente nodo.
- **Nodo Final.** Un nodo que indica la terminación de una palabra.
- **Nodo Vacío.** Un nodo que no es índice ni final.

Debido a que la implementación hace un alto uso de hilos, el trie se encuentra protegido para el acceso concurrente de hilos que desean consultar o modificar los nodos del trie mediante una matriz de exclusores mutuos por nivel del trie. Esto es, una matriz de $|\Sigma|$ localidades por k renglones, donde k es el nivel máximo a proteger. El bloqueo se realiza, por niveles y por localidades correspondientes a cada carácter del alfabeto, liberando cada exclusor mutuo en cuanto se sobrepasa el nivel. Gracias a éste esquema, los bloqueos

de niveles inferiores no bloquean el acceso a niveles superiores, dando mayor capacidad de concurrencia.

Para cada lista invertida se guarda la frecuencia global y el número de documentos con la palabra. Además cada entrada en la lista guarda una tupla de dos elementos: *identificador de documentos (docid)*, *frecuencia de la palabra en el documento (docfreq)*. En versiones antiguas del indexador la posición y la etiqueta (para documentos en formato XML [Yergeau04]) también guardados (opcional en la versión actual). Actualmente se usa un *docid* especial con dos elementos $(IDmayor, IDmenor)$. Cada documento es divido en pequeñas partes como es sugerido en [Bell95]. El $IDmayor$ esta dedicado para identificar a cada documento, en sí mismo. El $IDmenor$ es utilizado para identificar la partición del documento donde la palabra aparece. Bajo está disposición es posible controlar el número de bits utilizados para cada elemento del ID para representar las partes más importantes del documento (IDmenor), maximizando el número de documentos posibles. El identificador ID es propio de las colecciones locales, en la colección distribuida el identificador está dado por el URI [Berners-Lee05] del documento).

Los índices pueden crecer sin reindexar la colección entera, pero no puede ser reducido, esto es, el borrado de entradas no es posible. Para excluir que un documento aparezca como parte del conjunto de resultados, el documento debe ser marcado como borrado. Cuando el número de documentos borrados es grande (e.g. Cuando el 10 % de los documentos indexados son documentos marcados como borrados), es necesario reindexar la colección nuevamente.

5.4. Características del Modulo Indexador y Resolvedor de Consultas

El indexador creado es escalable, robusto y extensible. Cada una de éstas características convierten el sistema en un poderoso Sistema de Recuperación de Información Distribuido.

5.4.1. Escalable y Robusto

Un conjunto de computadoras puede ser utilizado para crear un conjunto de indexadores que interactúen entre sí. Si el conjunto de documentos sobrepasa los recursos de una computadora, la capacidad de indexamiento crece si se añaden más computadoras a la red de indexadores colaboradores, incrementando el desempeño global del sistema. El número de nodos aceptados es ilimitado, pero el desempeño de las operaciones en conjunto es relativa a la topología y a los algoritmos utilizados para distribuir las peticiones.

La interacción puede ser llevada a cabo para realizar balance de carga o colaboración. Los índices soportan distribución y replicación sobre redes heterogéneas. Existe un factor de replicación, configurado de manera local para cada índice obteniendo índices heterogéneos y tolerantes a fallos.

Dado que cada índice en la colección es independiente y funcional, si parte de la colección de índices colapsa el resto de la colección continua funcionando correctamente. Aún bajo condiciones de fallo, la colección entera puede sufrir pocas perdidas, debido a la replicación de los índices. La redundancia es ajustada utilizando un factor de replicación. Éste factor es dependiente de los requerimientos específicos de la colección de índices. Adicionalmente, escoger un subconjunto correcto de índices puede preservar la completitud en nuestros resultados. Este es un tema de investigación muy activo [Brin98, Cho04, Khoussainov04, Sun04, Si02, Danzig91, Craswell00, Clarke02, Clarke01, Crainiceanu04, Linga05]. El módulo resolvedor de consultas por su parte funciona de la siguiente manera para cuando el sistema es distribuido y una función de ponderación vectorial:

- Selección de los indexadores participantes en la consulta, dependiente del distribuidor (ver 6.1.1) utilizado).
- Cálculo de las estadísticas de los términos en cada una de las colecciones, se regresan al indexador hub.
- Unión y valoración de las propiedades en el indexador hub.
- Envío de nuevas estadísticas a los indexadores participantes en la resolución de la consulta, con el fin de reflejar un resultado “global”.

- Los indexadores ponderan según sus respectivas funciones y regresan una lista de documentos junto con su similitud a la consulta.
- Reunión en el indexador hub de todos los resultados, asi como su reordenamiento.

Mediante éste algoritmo es posible obtener buenos resultados globales debido a que las estadísticas son recalculadas para la colección distribuida. En pruebas realizadas bajo colecciones disjuntas, la precision y verificación ([Witten99]) se presentan como precisiones del 100 %, teniendo en cuenta como documento relevante los resultados obtenidos en una colección monolítica con los mísmos documentos. Sin embargo, el orden de recuperación es diferente.

5.4.2. Capacidad de Extensión

El indexador está escrito en C [Kernighan88] y Python [Foundation05]. El lenguaje C es perfecto para realizar las rutinas a bajo nivel, tal como las tareas de indexamiento, resolver consultas, manejo de memoria, acceso a disco, etc. En C se provee de un *API* para el acceso a bajo nivel de funciones, estructuras y algoritmos.

Python es utilizado como un manejador de protocolos, interfaz rápida de programación, y procesador de textos. Con Python se asegura un desarrollo rápido, mantenible y legible. Además, con Python es posible desarrollar módulos de enchufe dinámico (i.e. módulos que pueden ser enchufados y desenchufados sin necesidad de parar los servicios). El API de C y Python puede ser utilizado para crear nuevos módulos.

Es posible limitar la memoria disponible o dedicada al proceso de servicio de indexamiento y consulta. La configuración de memoria debe ser hecha en términos del propósito del nodo, pues el desempeño es directamente proporcional a la cantidad de memoria dedicada. La cantidad de memoria es configurada por el mediador (entre la memoria principal y secundaria) sobre el cual está basado el módulo de indexamiento a bajo nivel. Una cache acelera el proceso de lecturas y escrituras a memoria secundaria, mejorando el tiempo de indexamiento y consulta, por consecuencia caches de tamaño pequeño ocasionan un gran número de accesos a disco.

El indexador y el resolvedor de consultas hacen uso exhaustivo de hilos y memoria

compartida. Para reducir la sobrecarga debida a la creación y destrucción de los hilos, los hilos son guardados en un *banco de hilos*. Cada tarea realiza trabajos específicos y al completar alguno regresa preparado para realizar más trabajos. Así, es posible controlar la carga y evitar la saturación del sistema. En sistemas Unix es posible ajustar el número de tareas posibles usando el sistema de archivos *proc*.

El módulo indexador está construido sobre un sistema de caches, el cual está preparado para trabajo intenso de hilos. El sistema de caches es llamado **cache1**. Cache1 está diseñado especialmente para Sistemas de Recuperación de Información. Cache1 decrementa el número de acceso a la memoria secundaria cuando los bloques de datos son frecuentemente accedidos (usando las propiedades de la ley de Zipf[Baeza-Yates99] y el principio básico de temporalidad en el uso de los datos). Aún para accesos con distribución uniforme, cache1 proporciona un manejo de memoria simple, enmascarando mucha de la carga de programación en una implementación directa (además del aprovechamiento de la principio de temporalidad y localidad).

Cache1 proporciona características similares a la función estándar `mmap` de POSIX [Lewine94]. Cache1 proporciona más control y sin errores asíncronos, así como soporte para *archivos dispersos*. Cache1 puede ser compilado sobre `mmap`, así es posible utilizar los beneficios de `mmap` y mejorar sus características. Desgraciadamente, el acceso a archivos dispersos no es posible, y los errores asíncronos pueden hacerse presentes.

Cache1 usa recolección de basura con política de remplazo al último accedido, basado en conteo de referencias. La longitud de las páginas y bloques de datos pueden ser configurados para proveer acceso a localidades largas en archivos utilizando pocos bits. El máximo número de páginas o bloques de datos puede ser configurado para controlar el desempeño del sistema (i.e. Los servidores dedicados pueden tener muchos bloques en memoria para mejorar el desempeño, mientras que los servidores no dedicados pueden utilizar bancos pequeños de memoria, al costo de incrementar el número de accesos a disco).

A pesar de que cache1 es perfecta como interfaz de acceso a bajo nivel, cache1 no es una estructura de datos flexible. Ésta es la causa de que una estructura de más alto nivel llamada *alist*, haya sido desarrollada. Una alist es una lista de arreglos diseñada para almacenamiento secundario. Con *alist*-as es posible insertar y recuperar datos de longitud

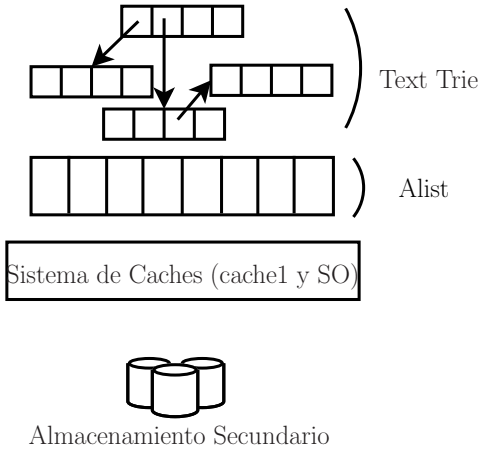


Figura 5.8: Jerarquía de Almacenamiento

variable. En estado de corrupción, alist provee mecanismos para recuperar información dañada y el API es más sencillo que el de cache1. Adicionalmente, *alist* cuenta con una interfaz adicional de acceso por medio de iteradores, que mejora el tiempo de respuesta de una alist. La Figura 5.8 muestra la jerarquia de acceso a los datos usada en el módulo de indexamiento.

La consulta en un indexador final está creada utilizando el modelo booleano y el modelo vectorial. Es posible implementar otros modelos de recuperación de información adicionando módulos que cumplan con la interfaz de *medición de similitud entre documentos* o *ranking*. El ordenamiento de los resultados se realiza mediante skiplists [MacDonald99]. Las skiplists permiten ordenamientos en un tiempo esperado de $O(n \log n)$ y tiempo lineal en visita de todos los elementos.

5.5. Sumario

A lo largo de éste Capítulo se discutió el módulo de indexamiento y consulta, los cuales son parte de la misma arquitectura y en la implementación comparte el mismo espacio de memoria, ya que se crean índices siempre al día.

Así mismo, se muestra la arquitectura creada para crear un SRI altamente distribuible, con capacidad para diferentes clientes e interfaces de usuario. La arquitectura creada

mantiene la posibilidad de enchufar componentes "en vivo", lo cual permite la conexión de clientes y controladores de manera dinámica posibilitando la programación de aplicaciones y la extensión del módulo, aún cuando este en funcionamiento.

La arquitectura esta formada por dos tipos de indexadores, que comparten una interfaz de indexador, y son los siguientes:

- Indexador Hub. Es un indexador que por sí mismo no conoce como indexar documentos o resolver consultas, pero es capaz de enrutar peticiones a otros indexadores que a su vez pueden enrutar nuevamente las peticiones o indexar. Los indexadores hub, son capaces de escoger un conjunto de indexadores a los cuales se les debe enrutar las peticiones basados en estadísticas de enrutamiento y conocimiento a priori de las colecciones. También es capaz de unir y re-ponderar los resultados obtenidos.

- Indexador Final. Es un indexador que realiza indexamiento y resolución de consultas, pero no enruta peticiones. También es el encargado de crear los archivos invertidos, mismos que deben ser optimizados para la resolución de consultas. Es capaz de resolver mediante un criterio de ponderación especificado o por omisión una consulta, misma que genera un conjunto de resultados los cuales son devueltos a la entidad que creo la consulta.

Es necesario enfatizar que los índices distribuidos soportan segmentación y son capaces de seguir prestando servicios de indexamiento y consulta, con la posibilidad de minimización de conjunto de resultados.

Capítulo 6

Aplicaciones de la Arquitectura

En los Capítulos anteriores de ésta tesis se definió la arquitectura necesaria para resolver el problema propuesto como parte central de la tesis. A lo largo de la tesis, también se mencionaron algunos detalles sobre la implementación de cada uno de los módulos de la arquitectura. Éste Capítulo muestra la unión de los módulos creados para crear un sistema completo de Recuperación de Información. El Sistema de Recuperación de Información creado es distribuido, extensible, modular, tolerante a fallas, tolerante a la segmentación de redes y altamente escalable.

La primera de las aplicaciones mostradas es la implementación de la arquitectura, la cual lleva el nombre de SPyDI (Simple Python Distributed Index), será descrita en la sección 6.1. De la implementación de SPyDI se desprenden dos aplicaciones de Recuperación de Información Distribuida: AFFAIRS (A Fully Federated Academic Information Retrieval System), explicada en la sección 6.2. La sección 6.3 está dedicada a U-Index (Universal Index), una máquina de indexamiento y consulta basada en el enriquecimiento de los documentos utilizando diferentes palabras clave y meta-datos. La sección 6.4 muestra un conjunto de aplicaciones basadas en los Servicios Web y objetos remotos mostrados en el Capítulo 3. Para finalizar, se presenta un breve resumen del Capítulo en la sección 6.5.

Las aplicaciones creadas tienen como propósito mostrar la versatilidad de SPyRO en el desarrollo de aplicaciones basadas en red, en cualquier área de investigación o desarrollo.

6.1. Implementación de la Arquitectura Propuesta para Recuperación de Información Distribuida

La implementación de la arquitectura es la base del paquete SPyDI. SPyDI está constituido por la unión del módulo de indexamiento-consulta y el módulo de adquisición de datos. La unión de los módulos es realizada utilizando SPyRO como mediador.

Para transformar SPyDI en una herramienta de desarrollo para la construcción de aplicaciones de Recuperación de Información y otras Tecnologías de Información. En el paquete se incluyen ambientes de desarrollo web y servidores autónomos para una independencia total. SPyDI es un paquete listo para producción, es autónomo, de pocas dependencias, y diseñado para funcionar desde computadoras de mano hasta grandes servidores corporativos. De igual forma, SPyDI está pensado para servir de plataforma de desarrollo para Sistemas de Recuperación de Información y otras tecnologías.

La implementación de los módulos ha sido discutida a lo largo de la tesis, por lo que se hará especial interés en las posibles uniones y topologías de interconexión entre módulos y submódulos, así como las aplicaciones y servidores que, aunque no forman parte de la arquitectura, fueron creadas con la finalidad de generar aplicaciones con pocas dependencias y autónomas.

Es posible utilizar los conocimientos de expertos para formar topologías de interconexión. Las mostradas a continuación están basadas en reconfiguración automática y destinadas a permitir el escalamiento, tolerancia a fallos y redundancia de los datos con pocos o ningún punto centralizado. Aunque la eficiencia de los protocolos también está asegurada, las mejores eficiencias son alcanzadas mediante configuraciones totalmente controladas como las de WebBase [Cho04] o Google [Ghemawat03, Brin98].

Con la ayuda de SPyRO y políticas de compartición de información, los indexadores permiten la creación de sistemas indexadores federados [Conrad97, Sheth90, Hasselbring00, Hsiao92, Khoussainov04]

6.1.1. Distribución de Adquisidores de Datos, Repositorios e Indexadores

La distribución sobre los adquisidores de datos, repositorios e indexadores es realizada mediante *distribuidores* de métodos que abstraen el proceso de distribución de operaciones. Un distribuidor es una interfaz que realiza la distribución mediante diversas técnicas, utilizando como base de la distribución un conjunto de llaves y las propiedades de una implementación en particular.

La interfaz consiste de los siguientes métodos:

- **Recuperar un resultado.** Ejecuta un método en un objeto registrado. El objeto debe estar ligado a un conjunto de llaves indicadas y regresa el resultado.
- **Ejecutar un método.** Ejecuta un método en un conjunto de objetos registrados, ligados a las llaves indicadas. Se regresa un conjunto de resultados.
- **Encuentra un nodo.** Encuentra y regresa un nodo registrado y asociado a un conjunto de llaves. El algoritmo es el mismo que el usado para *Recuperar un resultado*.
- **Registrar.** Registra un objeto y lo asocia con un conjunto de llaves.
- **Borrar de registro.** Desliga un objeto de las llaves a las cuales se encuentra asociado y lo borra de los registros.
- **Lista de objetos.** Regresa una lista de objetos asociados a un conjunto de llaves, utilizando el mismo algoritmo usado en *Ejecutar un método*.
- **Todos los objetos.** Regresa una lista de todos los objectos asociados con el distribuidor. No regresa los objetos alcanzables por los distribuidores registrados.
- **Cerrar.** Cierra formalmente un distribuidor.

Cada nodo en la red que desee participar en la distribución debe unirse a un distribuidor, y cada distribuidor debe conocer al menos otro distribuidor. De ésta forma, diferentes esquemas pueden ser utilizados para realizar la distribución ya sea basado en modelos P2P o módelos de distribución de carga. Por el momento, se encuentran implementados tres tipos de distribuidores:

- **No distribuidor.** No hay distribución, utilizado para servidores que no acepten distribución.
- **Distribuidor Aleatorio.** Realiza la distribución de manera aleatoria entre los elementos registrados.
- **Distribución de cobertura completa.** Realiza la distribución hacia todos los elementos registrados.
- **Por llave.** Basado en una implementación sencilla de tabla hash distribuida (DHT).

La distribución es realizada por una interfaz, por lo que no está limitada a distribuir únicamente por los métodos anteriores.

6.1.2. Aplicaciones Adicionales

Cualquier instalación de SPyDI es capaz de trabajar sin modificaciones o configuraciones a herramientas que no forman parte de SPyDI. Ésta simplicidad resulta en la rápida y fácil creación de nodos y sitios SPyDI.

La configuración por omisión de los módulos está orientada a crear, con el mínimo esfuerzo, servidores HTTP [Consortium] y SMTP [Postel82]. Aunque el uso de servidores autónomos facilita la instalación de SPyDI, existe un costo asociado al desempeño. Debido al diseño modular del sistema, es fácil remplazar los servidores por versiones optimizadas. Éste diseño fue creado debido a que muchas personas interesadas en el proyecto no poseen el el conocimiento técnico y/o el tiempo necesario. Las implementaciones comerciales de los servidores son generalmente fáciles de configurar, aunque de alto costo.

Servidor Autónomo HTTP

SPyDI utiliza una simple y a la vez poderosa interfaz web. La interfaz ha sido desarrollada como web debido a que las Interfaces de Usuario basadas en Web permiten un rápido desarrollo e independencia de la arquitectura y plataforma de los clientes.

El servidor está escrito de manera que se aproveche completamente la multitarea. El servidor es capaz de atender peticiones GET, HEAD y POST. Además, tiene la habilidad

de manejar o redirigir las peticiones hacia programas escritos en Python u otro lenguaje de programación conforme con soporte para CGIs [D. Robinson04]. El servidor HTTP maneja directorios virtuales manejados por funciones registradas en Python, parecido al comportamiento de mod_python [Trubetskoy03].

Páginas Web con Python Empotrado. Las interfaces proporcionadas en SPyDI están dirigidas tanto a la administración como a los usuarios regulares. La interfaz para la administración realiza tareas de administración variadas, tales como ajustar parámetros, modificar las entradas del cosechador, de los repositorios, etcétera. La interfaz de usuario permite realizar consultas, navegación, y peticiones *push*. En el desarrollo de estas interfaces es necesario un sistema de desarrollo de páginas dinámicas que permita el desarrollo rápido, integrado al sistema, soporte multilenguaje, de fácil manejo y poderoso a objetos remotos (ver Capítulo 3).

Para llenar estos requerimientos, se ha creado un lenguaje para la creación de sitios web dinámicos basado en python llamado PyEW (Python Embedded Webpages o Páginas Web Empotradas en Python). PyEW provee de una plataforma de desarrollo rápido de sitios web. PyEW trabaja bajo el servidor autónomo de SPyDI o bajo cualquier otro servidor que soporte programas CGI, como el servidor Web Apache [Server]. En apache, las páginas web dinámicas de PyEW pueden correr al menos en tres formas de ejecución: como CGI, FastCGI, y bajo mod_python. Corriendo bajo apache u otro servidor web, requiere esfuerzo adicional que correr bajo el servidor HTTP autónomo.

PyEW es similar en sintaxis empotrada y funcionalidad a PHP [Achour05] o ASP [Inc.05], como se muestra en la Figura 6.1. Existen otros proyectos de páginas empotradas en Python, tales como mod_python [Trubetskoy03], Webware [Esterbrook01] o Zope [Corporation]. Mod_python acelera la ejecución de programas CGI escritos en python en el servidor web Apache [Server]. Mod_python funciona únicamente en el servidor Web Apache y soporta páginas web empotradas. Mod_python contiene fuertes dependencias al servidor Apache, por lo que sólo puede ser utilizado con una instalación de Apache.

Webware es conjunto de componentes de software para desarrollar aplicaciones basadas en web orientadas a objetos. Desafortunadamente, es dependiente del servidor apa-

```
<?python
import SPyDI as idx

def paginacion(length, begin, count, create,
               max=12,anterior="Anterior",siguiente="Siguiente"):
    ...

def handle_query(query,begin,count,qtype):
    ...

query = get('query')
count = int(get('count',10))
begin = int(get('begin',0))
qtype = int(get('qtype',idx.QUERY_VECTOR))
results, approx, terms = handle_query(query,begin,count,qtype)

?>

<form action="%(dirScript)s/busca" method="POST">
<label for="query" accesskey="l">
<input name="query" type="text" size="50" value="%(query)s" /></label>
<label for="buscar" accesskey="b">
<input type="submit" name="buscar" value="Buscar" /></label>
</form>

<?python
echo(",  ".join(map(lambda term:
'%s ocurrencias de <a href="%s/busca?query=%s&buscar=Buscar">%s</a>'%(
     term[1],dirScript,term[0],term[0]), terms)))
?>
<p>
Resultados para: %(query)s
</p>

<table id="resultados" border="0"><tbody>
<?python
pos = begin
for res in results:
    url, sim = res
    pos+=1
    echo("""<tr><td>%d.-  </td><td>%d%%</td><td><a href="%s">%s</td></tr>
    <tr><td></td><td>Avance</td></tr>"""%(pos, int(sim*100), "file://"+url, url))
?>
</tbody></table>

<?python


paginacion(approx, begin, count,
           (lambda x: echo('<a href="%s">%s</a>  '%(callThisScript({
    'query':query, 'begin': x[0], 'count': x[1], 'qtype': qtype}), x[2]))), 15)

if len(results) > 5:
    ?>
<br /><br />

<form action="%(dirScript)s/busca" method="POST">
...
```

Figura 6.1: Ejemplo de una página PyEW

che.

Zope es un servidor de código abierto para construir sistemas de manejo de contenido, intranets, portales y aplicaciones personalizadas. Es realmente complejo, e inapropiado para cómputo pervasivo.

Para mejorar PyEW se han creado paquetes de soporte para manejo y registro de usuarios, envío de correo y caches de archivos PyEW. Además, en conjunto con SPyRO es posible alcanzar una plataforma distribuida de desarrollo web.

Un pequeño API es incluido, adicionalmente al conjunto de clases y funciones provisto por el API de una distribución del lenguaje Python. Éste API es orientado para crear un manejo amigable de las variables CGI y las tareas comunes realizadas con el CGI. Estos paquetes proveen de un conjunto de herramientas especializadas para realizar tareas en un ambiente común de desarrollo web.

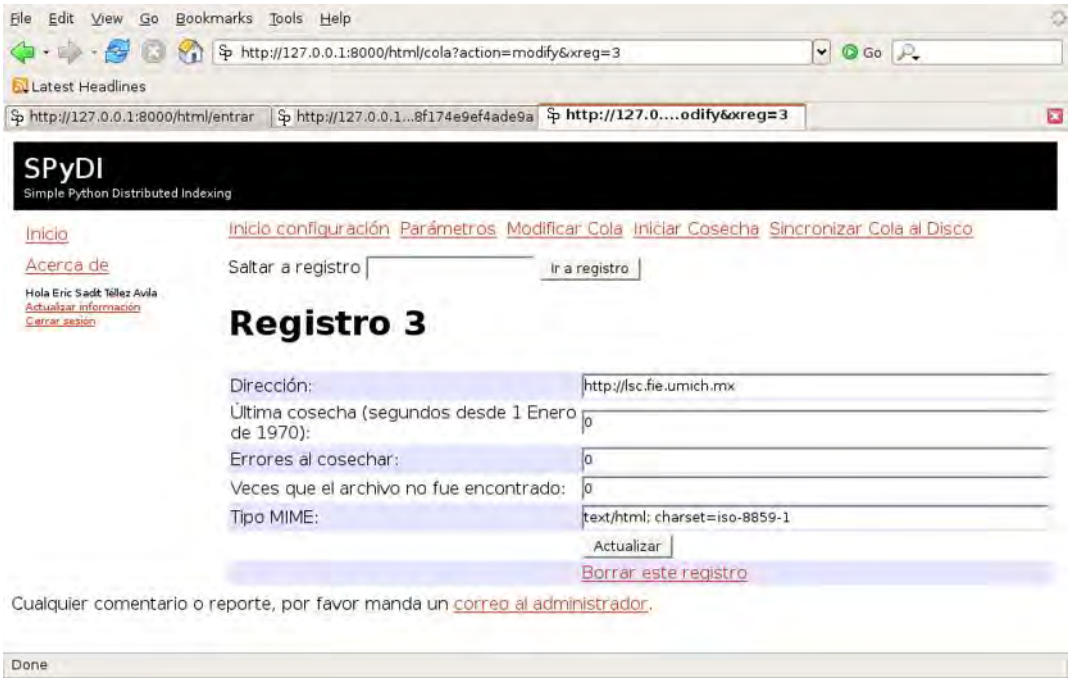
6.1.3. Interfaces de Usuario y Servicios Web

La interfaz de configuración está basada en Web y Servicios Web. La interfaz web está diseñada para que la experiencia del usuario sea placentera y de fácil comprensión, con abolición de ambigüedades y opciones de configuración minimas. Las interfaz de Servicios Web está diseñada para controlar a detalle los parametros y realizar todas las tareas posibles a un nivel de configuración profundo. La interfaz de servicios web no es mostrada en está tesis, pero se encuentra disponible en el sitio web de la SPyDI[1].

En la Figura 6.2 se muestran dos interfaces de la configuración de la cola del cosechador. En la interfaz de configuración de la cola, se permite la modificación de los registros, los cuales pueden ser borrados, editados o insertados. Los registros son las entidades que serán descargadas. La edición de los registros es posible mediante una lista que permite navegar por los registros o especificando el registro en particula que se desea. También es posible sincronizar la cola al disco y ordenar el inicio de la cosecha.

La búsqueda es realizada por medio de una interfaz muy simple, la cual está diseñada para minimizar confusiones 6.3(a). Los resultados son presentados en orden de relevancia para la consulta dada 6.3(b). La relevancia de cada resultado es mostrada como parte

[1] http://www.spyron.org



(a) ***Modificación de registros***

(b) ***Modificación de los parámetros del cosechador***

Figura 6.2: Interfaz para la configuración de la cola del cosechador

(a) ***Interfaz para realizar búsquedas***

(b) ***Presentación de resultados***

Figura 6.3: Interfaces de búsqueda de SPyDI

de la presentación, esto da al usuario una idea de la posibilidad que tiene el documento de ser de su importancia. También se muestra una breve introducción a los documentos, y se enfatizan los términos que fueron considerados para la búsqueda. Cada documento mantiene un apuntador al documento que se mantiene en el cache de SPyDI, el cual es un repositorio distribuido.

Además, se muestran una serie de bloques de resultados posibles números como páginas de resultados, localizado en el fondo de la página. En esta misma localidad, se muestra la posición que actualmente se encuentra viendo el usuario, así como apuntadores hacia adelante y atrás que facilitan la navegación por los resultados.

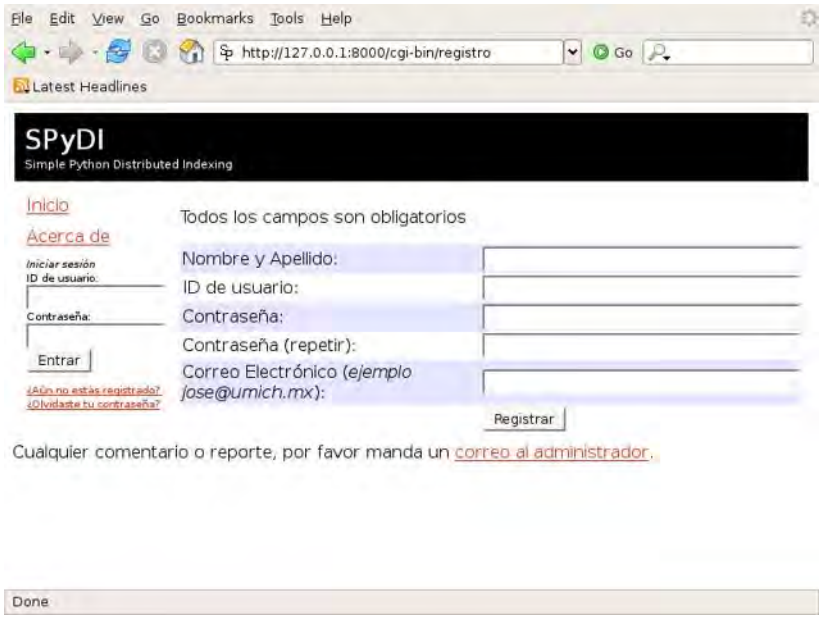
La interfaz para el registro de usuarios nuevos es simple (ver Figura 6.4), con pocos datos requeridos para los usuario para no incrementar la complejidad del registro. El registro revisa por errores típicos de llenado de información, y ayuda a corregirlos con mensajes explicitos en los errores, además los campos con errores son marcados con colores contrastantes para localizar rápidamente los errores.

Es preciso notar, que las interfaces web creadas tienen soporte multilenguaje, y que las mismas estructuras y programas son utilizados para mostrar resultados en varios idiomas. Por el momento, únicamente se soporta el idioma inglés y el castellano.

6.1.4. Estado Actual y Perspectivas a Futuro de SPyDI

SPyDI es capaz de crear SRI distribuidos. Los nodos pueden ser colaborativos o preparados para distribución de carga. Además se provee de una serie de herramientas desarrolladas para dar a SPyDI independencia y capacidad de desarrollo. Las topologías pueden ser creadas por expertos o llevadas a cabo por métodos automáticos de interconexión basados en tecnologías P2P. El indexador genérico (diseñado para la cohexistencia la máquina de resolución de consultas) permite el uso de *índices siempre al día* y ahorro de memoria primaria. La modularidad y generalidad de SPyDI, permite su especialización en diferentes aplicaciones basadas de recuperación de información. Además, el uso de objetos remotos permite la extensión de SPyDI sin necesidad de reiniciar servicios, así como expande la cantidad de módulos posibles.

SPyDI carece de algunas características en el “estado del arte” en algunos de sus



(a) ***Registro de un usuario nuevo***

(b) ***Forma para recuperar una clave pérdida***

Figura 6.4: Interfaces de registro y recuperación de claves olvidadas

módulos. Es deseable el cifrado de datos en repositorios, especialmente en redes colaborativas de baja confianza. Se requieren especializaciones para alto rendimiento, varios niveles de colaboración, reconfiguración dinámica de topologías. Para mejorar el rendimiento de los módulos, es necesario realizar pruebas de desempeño en cada uno de los módulos y realizar las respectivas modificaciones de las deducciones de las pruebas.

Además, SPyDI carece de interfaces gráficas de configuración y control de los elementos distribuidos, y únicamente cubren los parametros básicos de funcionamiento. Por tanto, es necesario crear las interfaces necesarias para controlar SPyDI como sistema distribuido en un conjunto, así como para abarcar la mayoría de los parámetros de configuración. Por el momento, las interfaces de configuración se enfocan a los parametros básicos que controlan el funcionamiento en un nodo.

6.2. Sistema de Recuperación de Información Académica Federado: AFFAIRS

AFFAIRS (A Fully Federated Academic Information System) es un SRI especializado a datos académicos. Basado en SPyDI, AFFAIRS se beneficia directamente de la arquitectura presentada en ésta tesis heredando un sistema de recuperación distribuido, modular, y escalable. AFFAIRS está diseñado para ser alojado en servidores autónomos e independientes. Cada servidor puede ser la estación de trabajo de un autor o un servidor departamental, a nivel facultad, universidad o conjunto de universidades.

Debido a las estructuras de alto nivel que implican las entidades académicas (autores, artículos, proyectos, tesis, reportes, etcétera), la colección entera es modelada mediante ontologías [Khan00, Noy04]. El índice y los datos están distribuidos a lo largo de una red heterogénea de computadoras. Un usuario es capaz de navegar la ontología, consultar sobre el texto completo de los documentos, descargar información en el sistema, así como publicar información en el sistema.

El propósito de AFFAIRS es facilitar el proceso de manejo, distribución y desarrollo de productos académicos.

Los datos son replicados a lo largo de la red, pero un usuario del sistema percibe

los datos como pertenecientes a un servidor central (no existente). Cada cambio local crea un cambio global, sin esfuerzo. Marcas de tiempo son usadas para mantener la coherencia entre versiones en nodos diferentes.

AFFAIRS puede ser personalizado para servir a una comunidad especifica. La configuración en AFFAIRS permite un la recuperación del sistema aún bajo condiciones de falla.

Uno de los más importantes proyectos para manejar información académica es Citeseer [Lawrence99a, Lawrence99b, Bollacker98, Bollacker99]. La función de similitud utilizada por Citeseer está basada en la semántica implícita en las citas entre publicaciones, de una manera similar pero menos poderosa que el PageRank de Google [Brin98].

AFFAIRS trabaja sobre el mismo dominio que Citeseer, pero además de ser autónomo también es posible cambiar la colección de manera asistida, esto es, realizar modificaciones directas a la colección de manera asistida por el usuario dueño del documento. Adicionalmente, se soportan otras entidades académicas y manejo por ontologías, usadas para describir el sistema. Es posible descubrir información no programada en la colección. El modelo distribuido de AFFAIRS proporciona tolerancia a fallos y comunidades de usuarios AFFAIRS,

Además, aunque AFFAIRS fue creado para trabajar con información académica, el dominio puede ser cambiado de una manera muy sencilla cambiando las ontologías descriptivas y realizando pequeñas personalizaciones.

6.2.1. Sindicación y Agregación

La sindicación es método utilizado para compartir el contenido web de un sitio con otros sitios o clientes web, sin la necesidad de visitar los sitios de manera explicita.

La sindicación es realizada en diferentes formatos, los más populares en la actualidad son

- *RSS* [etalb] es un formato de sindicación basado en XML [Yergeau04] y RDF *RDF* [etal.a].
- *Atom* [Nottingham03] es un lenguaje de sindicación basado en versiones antiguas de

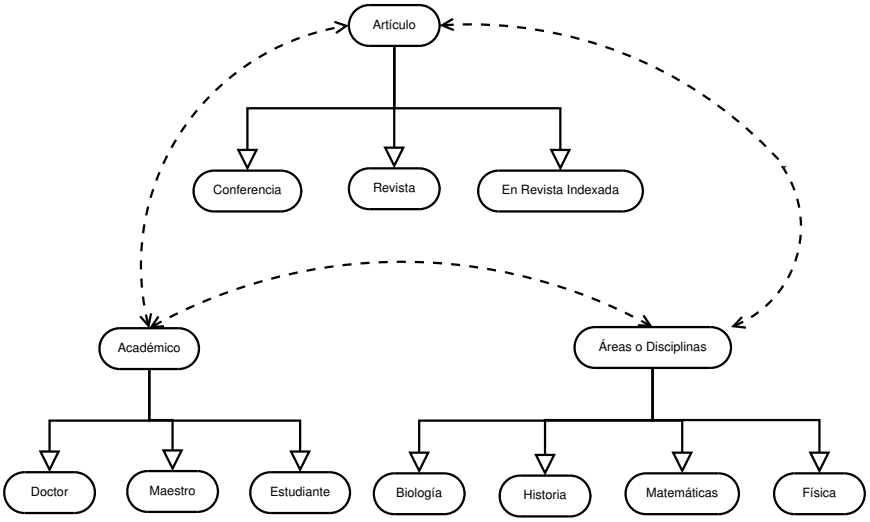


Figura 6.5: Entidades en una Ontología Simplificada

RSS, en términos generales, Atom es un lenguaje mucho más flexible que RSS (desde la versión 2.0 de RSS sus principales deficiencias fueron resueltas).

Usando la información almacenada y programas CGIs, es posible crear fuentes RSS o Atoms para proveer una manera simple de integración con *agregadores*.

Con AFFAIRS es posible realizar sindicación en varias formas. Es posible crear Servicios Web provistos por SPyRO para obtener sindicación de una manera elegante y versátil de objetos seleccionados o métodos ejecutados bajo un objeto o un conjunto de argumentos (siguiendo la arquitectura de REST propuesta por Fielding [Fielding02]).

La agregación también es soportada usando páginas web empotradas en Python. La agregación es el proceso de unir varias fuentes de sindicación en una página web. La agregación de fuentes RSS y Atom es fácil cuando se utilizan herramientas de Python para el procesamiento de RDF y XML. Otra manera de realizar agregación es usar directamente los Servicios Web provistos por SPyRO u otro proveedor de servicios web.

6.2.2. Manejador de Ontologías

Ésta es una adición a la arquitectura propuesta y está enfocada a dar una semántica de búsqueda y navegación a los datos [Noy04]. Y se incorpora directamente en los módu-

los de adquisición de datos, y consulta en los submódulos que manejan la lógica de los componentes.

El manejador de ontologías es usado para descubrir relaciones escondidas entre entidades académicas. Por ejemplo, una cita es descubierta interpretando relaciones mutuas entre un artículo que cita y otro que es citado. El manejador es capaz de deducir relaciones más complejas como las creadas por saltos múltiples entre entidades académicas, e.g. grupos de trabajo, autores de influencia sobre otro autor o tema (por citas de múltiples saltos), agrupamiento de publicaciones, agrupamientos de tópicos, etcétera. La Figura 6.5 muestra la ontología simplificada para información académica. Ésta ontología está especializada en varios sub-tipos de publicaciones como un libro, una conferencia, una revista, o un trabajo publicado bajo el *Science Citation Index* . Éste ejemplo utiliza únicamente pocas dependencias (líneas punteadas) porque se usa una ontología muy simple y general. Aún siendo una ontología muy simple, el número de dependencias implícitas es muy alta, ya que la dependencia en un elemento general hereda sus dependencias a los elementos más especializados. Ontologías más detalladas tienen dependencias a niveles especializados, que no se encuentran en elementos generales. Una ontologia completa será publicada cuando el módulo quede completado.

6.2.3. Sincronización de Repositorios

Para asegurar permanencia de los datos, aún en condiciones de fallo, los datos son replicados a través de una serie de nodos. La replicación de datos lleva a inconsistencias sobre versiones de los registros, éste es un problema desafiante, por lo qué en el presente nivel de desarrollo la solución es muy básica, aunque resuelve los problemas enfrentados en el escenario.

Muchas heurísticas son usadas para mantener sincronizados los repositorios. Básicamente, las violaciones a la sincronización de datos son causadas por *modificaciones* (inserción, borrado y edición de registros) y *perdidas de datos*.

Los registros tienen marcas de tiempo que indican la modificación y creación de los registros. El número de registros puede incrementar o disminuir. Las discriminaciones entre posibles eventos es una tarea complicada. Se distinguen los siguientes eventos para prevenir

perdidas de datos y asincronía entre nodos: *inserción de registros, actualización de registros, borrado autorizado de registros, borrado accidental de registros (perdida), corrupción de registros.*

Para resolver el problema se utilizan marcas de tiempo en cada registro para mantener bajo control los cambios. Las marcas de tiempo ayudan a manejar registros duplicados (dispares en tiempo) y unir conjuntos de registros con registros cuyas marcas de tiempo no sean coherentes entre sí para crear un conjunto de registros totalmente coherente.

El nuevo conjunto de registros debe ser replicado sobre la red, por lo que el proceso de replicación puede ser:

- En tiempo real. Las replicas son esparcidas en el momento que la actualización sucede.
- Bajo demanda. Las replicas son creadas cuando los nodos realizan peticiones sobre el conjunto de registros.
- Calendarizadas. Los registros son actualizados en horarios y fechas especificas.

El borrado de los registros es realizado creando registros especiales que marcan otros registros como borrados, además de no incluir el registro en los conjuntos resultantes. Marcar es una buena solución para el problema de borrado porque si un registro existe en algún conjunto de registros este debe ser incluido en un conjunto actualizado de registros. Para completar el borrado de registros las marcas de borrado son destruidas después de cierto tiempo, que prácticamente asegure el borrado completo del registro.

Los usuarios no tienen control sobre como sus datos son replicados a través de una red AFFAIRS, esto es para disminuir la cantidad de labores administrativas por parte de los usuarios. Cada nodo AFFAIRS tiene la política de confianza en sus compañeros de AFFAIRS, por lo que, los nodos en una red AFFAIRS deben ser *nodos selectos* ya que AFFAIRS no soporta ataques maliciosos dentro de la red. Contra modificaciones accidentales y corrupción, se utilizan firmas hash de cada registro para prevenir modificaciones accidentales y corrupción de los registros. Si no es posible confiar en un nodo, es mejor no utilizarlo dentro de la red AFFAIRS construida. De cualquier forma, es posible compartir servicios (consultas e indexamiento) dentro de AFFAIRS sin la necesidad de compartir o replicar datos.

Es importante enfatizar que para gozar de los servicios de AFFAIRS no es necesario ser un nodo AFFAIRS. Los servicios de usuario de AFFAIRS se encuentran en otro nivel de aplicación, por lo que únicamente es necesario registrarse en un servidor o comunidad AFFAIRS para disfrutar de todos los servicios a nivel de usuario provistos por AFFAIRS.

6.2.4. Estado Actual y Perspectivas a Futuro de AFFAIRS

La implementación de AFFAIRS aún no soporta deducciones basadas en la ontología, por lo que no es posible utilizar todo el poder de las ontologías por el momento. Además, el identificador de entidades en los textos académicos se encuentra en un estado muy inmaduro.

Aún con estas deficiencias, AFFAIRS es un servidor funcional, dedicado a la recuperación de información de tópico académico y es una especialización simple de la implementación de la arquitectura distribuida presentada en ésta tesis.

Además, estudios de usabilidad y accesibilidad deben ser realizados con el afán de detectar y mejorar la experiencia del usuario.

6.3. Índice Universal

El Índice Universal es una máquina de búsqueda basada en SPyDI. En adición a las características generales de SPyDI, se realiza el enriquecimiento de los documentos utilizando un conjunto de palabras especiales llamadas *calificadores de documento*. Los calificadores de documento son interpretados como modificadores de los documentos, asociando propiedades especiales a estos mismos. Un uso particular de los calificadores es la restricción de acceso a documentos a nivel de usuario o grupo. También puede ser utilizado para añadir información acerca de formatos, categorías, fechas, etcétera.

La idea central es recuperar documentos basados en dos operaciones diferentes de similitud: similitud de calificadores y similitud de contenido contra consulta. La similitud de calificadores es realizada por operaciones booleanas y mediante funciones más complejas como el *modelo vectorial* [Baeza-Yates99].

La segunda parte, la similitud en el contenido del documento contra la consulta

es independiente de la similitud de calificadores. Es posible utilizar similitudes basadas en $TF \times IDF$, modelos probabilísticos [Baeza-Yates99], o cualquier modelo de Recuperación de Información.

Gracias a que es una especialización de SPyDI, U-Index puede ser accedido utilizando Servicios Web. También se ha creado una página web con la mayoría de los servicios disponibles. Además, el módulo de indexamiento y consulta puede estar distribuido en un conjunto de computadoras heterogéneas.

Utilizando control de acceso, un usuario percibe un Indexador y un procesador de consultas personalizado. Cada usuario registrado puede enviar documentos e interactivamente mejorar o modificar los documentos utilizando calificadores. Cada usuario es capaz de producir jerarquías de documentos, permitiendo a otros usuarios acceder la jerarquía a cualquier nivel debajo de las jerarquías permitidas. La percepción bajo consultas es completada con índices personalizados, índices compartidos por grupos e índices protegidos (las consultas no pueden ser realizadas a ciertos documentos). Los usuarios pueden realizar anotaciones a documentos, recuperar grupos por calificadores, contenido o combinaciones de ambos.

Los formatos reconocidos para los documentos son archivos de texto plano, pdf, postscript, html, rtf, y word de microsoft office. Es posible extender a muchos otros utilizando la interfaz de plug-ins incluida en SPyDI.

6.3.1. Modelo Booleano de Calificación

Un documento es representado como $D = (K, T)$ donde K es un conjunto de términos especiales llamados "calificadores", "propiedades" o "anotaciones" del documento D. T es el conjunto de términos que contiene el documento. Sea n el número de calificadores en el sistema y m términos en la colección, los calificadores del documento i son representados como el vector de booleanos $K_i = (k_1^i, k_2^i, k_3^i, \ldots, k_n^i)$. El vector $T_i = (t_1^i, t_2^i, t_3^i, \ldots, t_m^i)$ describe los términos del documento i.

Entonces, $k_1^a \wedge k_1^b, k_2^a \wedge k_2^b, \ldots, k_n^a \wedge k_n^b$ recupera los documentos que contienen todos los calificadores y $k_1^a \vee k_1^b, k_2^a \vee k_2^b, \ldots, k_n^a \vee k_n^b$ obtiene los documentos con cualquiera de los calificadores. El modelo puede ser fácilmente extendido para soportar expresiones en *Forma*

Normal Conjuntiva o *Forma Normal Disyuntiva*.

La similitud entre dos documentos D y Q es:

$$sim(D,Q) = sim(T_D, T_Q) \times [\sum_{j=1}^{m} k_j^D k_j^Q \geq a] \tag{6.1}$$

Donde a es 1 para la operación *union* y a es $\sum_{i=1}^{n} k_i^Q$ para la *intersección*. $[A \geq B]$ regresa 1 si $A \geq B$ es verdadero o 0 si es falso.

El número de calificadores en la colección es representado por m. La función sim sobre dos documentos vectorizados T_D y T_Q es la función de similitud usada por modelos de RI sin calificadores.

El prototipo creado implementa una medida de similitud basada en $TF \times IDF$. La función de similitud es como sigue:

$$sim(D,Q) = \frac{\sum_{i=1}^{n} w_i^D w_i^Q}{\sqrt{\sum_{i=1}^{n} (w_i^D)^2} + \sqrt{\sum_{i=1}^{n} (w_i^Q)^2}} \times [\sum_{j=1}^{m} k_j^D k_j^Q \geq a] \tag{6.2}$$

Donde w_i^D es el *peso* del término i en el documento D.

D y Q son usados como T_D y T_Q sin distinción para simplificar la notación y enfatizar que D y Q son una clase más general que T_D y T_Q, respectivamente.

6.3.2. Modelo Vectorial de Calificadores

El modelo booleano es demasiado estricto en el filtrado de resultados, y no se comporta de manera adecuada para la recuperación por medio de calificadores de una forma inexacta o más relajada. La perspectiva booleana es similar al esquema utilizado por bases de datos relacionales.

Es posible utilizar una forma más rica de manejar los calificadores. Muchas aplicaciones se pueden beneficiar de la perspectiva relajada en el manejo de los calificadores.

Una medida apoyada sobre el producto punto nos permite relajar el filtrado de documentos. El vector de calificadores es puesto a 1 si el calificador existe en el documento y 0 de otra forma. Para una consulta más fina, es posible utilizar valores 0 a 1 para expresar la importancia del calificador en el documento.

El producto punto para la ponderación en calificadores se realiza de la siguiente manera:

$$sim(D,Q) = sim(T_D, T_Q) \times \frac{\sum_{i=1}^{m} k_i^D k_i^Q}{\sqrt{\sum_{i=1}^{m} (k_i^D)^2} + \sqrt{\sum_{i=1}^{m} (k_i^Q)^2}} \tag{6.3}$$

Específicamente para el prototipo, la función de similitud es como sigue:

$$sim(D,Q) = \frac{\sum_{i=1}^{n} w_i^D w_i^Q}{\sqrt{\sum_{i=1}^{n} (w_i^D)^2} + \sqrt{\sum_{i=1}^{m} (w_i^Q)^2}} \times \frac{\sum_{i=1}^{m} k_i^D k_i^Q}{\sqrt{\sum_{i=1}^{m} (k_i^D)^2} + \sqrt{\sum_{i=1}^{m} (k_i^Q)^2}} \tag{6.4}$$

En general, es posible utilizar cualquier modelo de RI para ponderar la consulta contra el contenido o los calificadores.

6.3.3. Representación de los Calificadores

El conjunto de calificadores posibles es ilimitado. Pensando en la clasificación, fácil identificación y auto-descripción de los calificadores, se propone la utilización de URIs [Berners-Lee05]. Por ejemplo, es posible representar un calificador de categoría de la siguiente manera:

```
categoría://nombre-categoría[/nombre-sub-categoría[/etcétera]]
```

Cualquier recurso puede ser identificado utilizando URIs. Otro ejemplo es la representación de fechas, por ejemplo usando un URIs especial de la forma `fecha://año/mes/día`.

6.3.4. Servicios Web en U-Index

El prototipo está creado utilizando soporte para SPyRO (ver Capítulo 3), por lo que es posible realizar acceso al sistema por medio de Servicios Seb. Gracias a que U-Index se encuentra sobre SPyDI, el módulo de adquisición de datos (ver Capítulo 4) se encuentra dentro de U-Index: los documentos pueden ser descargados por URIs o enviados directamente mediante el método *push*. Además es posible enviar documentos utilizando una interfaz web con procedimiento POST.

La interfaz por medio de Servicios Web permite el acceso desde aplicaciones remotas, por lo que cualquier programa con conexión a Internet es capaz de controlar el

prototipo. El controlador remoto puede añadir documentos a la colección, así como calificadores a los documentos. Las búsquedas pueden ser realizadas a través de la interfaz de Servicios Web, por lo que los resultados pueden ser presentados en una gran variedad de formatos e interfaces de usuario, dependientes de los clientes de los Servicios Web.

6.3.5. Interfaz Web de U-Index

La interfaz web permite probar el prototipo sin necesidad de utilizar la interfaz de Servicios Web directamente. Las operaciones posibles por medio de la interfaz web son las siguientes:

1. **Enviar archivos.** Se envían archivos para ser indexados por medio de una forma HTML.

2. **Cosechamiento de documentos** Se descargan e indexan documentos asociados a una lista de URIs proporcionadas.

3. **Inserción Directa**. La inserción directa de texto de texto utilizando una forma HTML.

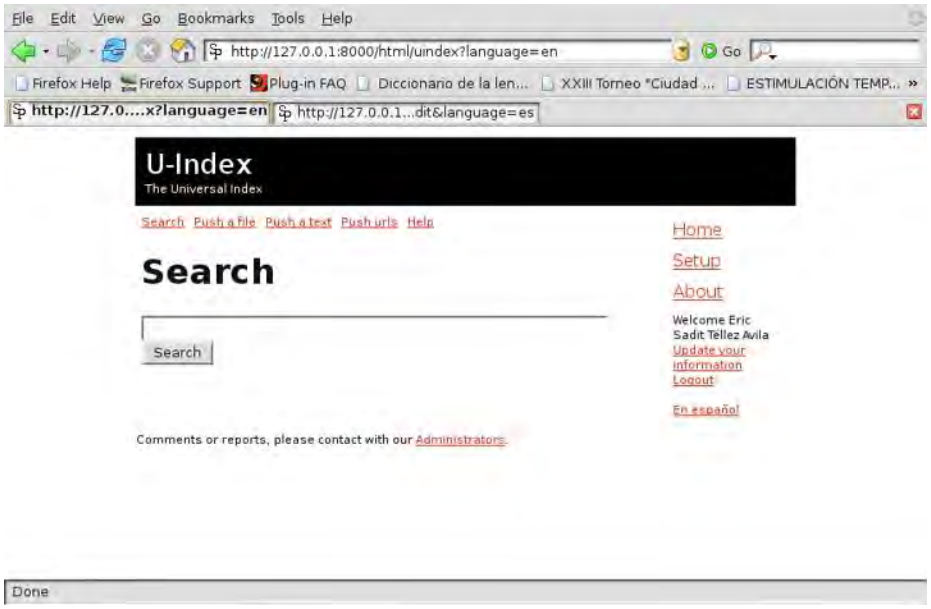
4. **Modificación de Calificadores**. Los modificadores de un documento son modificados.

La Figura 6.6 muestra la página web del prototipo, disponible en la dirección de página web **http://www.spyron.org**. La interfaz soporta multiples lenguajes y envío por varios tipos de fuentes, así como ayuda en línea 6.7.

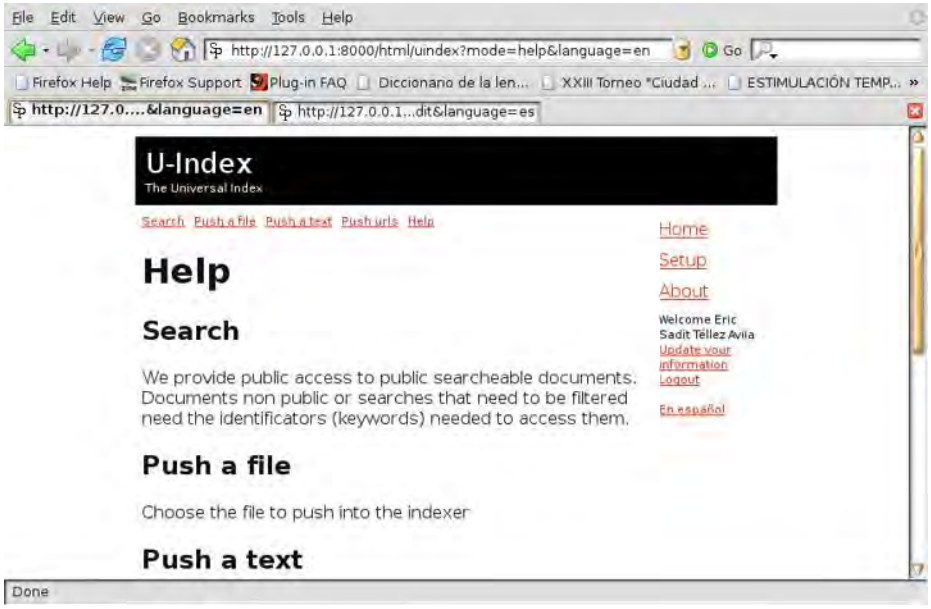
6.3.6. Estado Actual y Perspectivas a Futuro del Índice Universal

El modelo es capaz de crear índices personalizados, comunidades indexadas, y permitir consultas sobre documentos seleccionados. El modelo de calificadores no está dirigido únicamente al control de acceso, el filtro de resultados es posible en una forma rápida y elegante.

El prototipo soporta el modelo booleano y en parte el modelo vectorial de calificadores, soporta el envío de archivos al servidor y ordenes de descarga de URIs. El prototipo

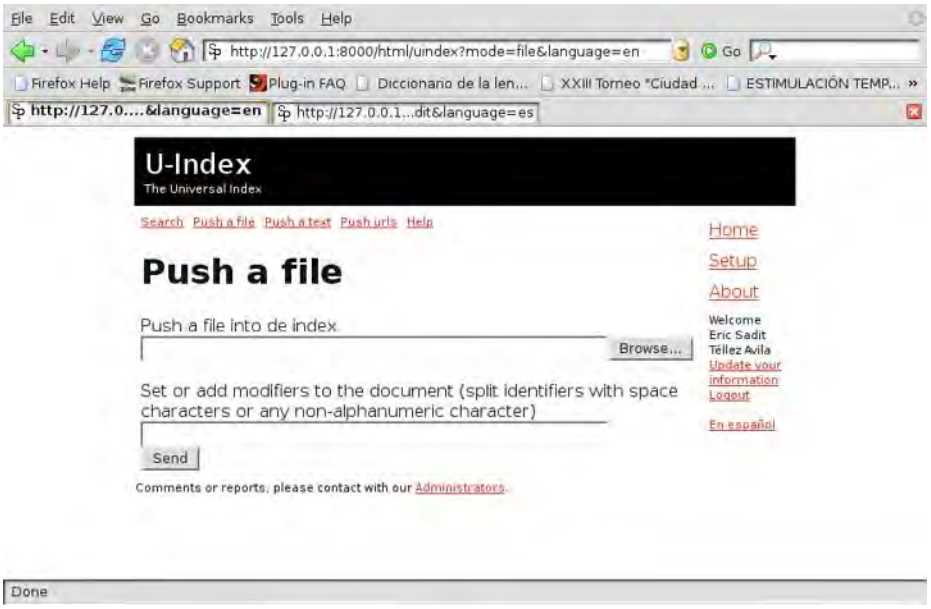


(a) ***Página de Búsqueda, los calificadores son parte de la consulta***

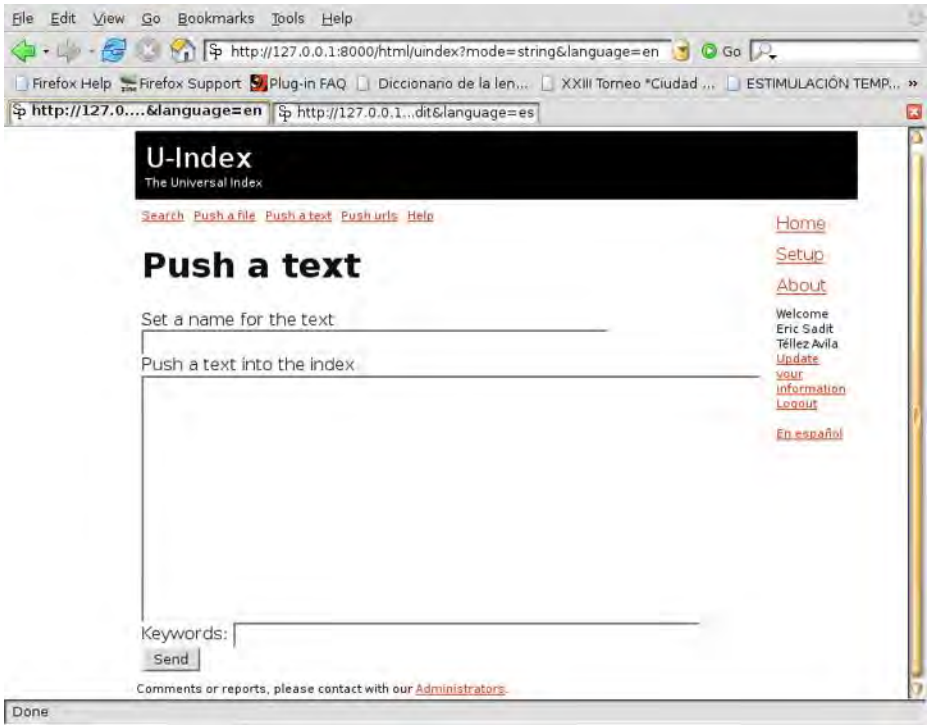


(b) ***Página de Ayuda***

Figura 6.6: Páginas de Búsqueda y Ayuda de la Interfaz del Índice Universal



(a) ***Envío de un archivo***



(b) ***Envío de un texto***

Figura 6.7: Páginas de envío de un archivo y de un texto. Ambas permiten adición de calificadores

tiene dos interfaces permitiendo el uso directo del prototipo usando la interfaz de usuario web o la programación de aplicaciones utilizando el API de Servicios Web.

Sin embargo, la interfaz web carece de muchos principios y guías de usabilidad, para resolver éste problema en un futuro se realizarán estudios de usabilidad para identificar con precisión las áreas de necesidad. Problemas como la fácil navegación, previsualización de los documentos en los resultados, e interfaz multilenguaje serán de especial interés en el estudio.

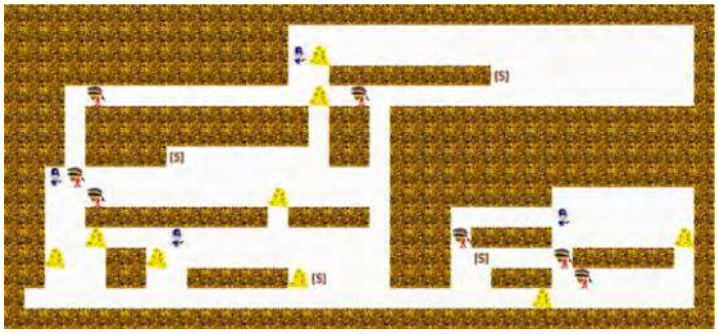
Las expresiones regulares no son implementadas en la presente versión del sistema. En posteriores implementaciones, las expresiones regulares serán incorporadas debido a que muchas aplicaciones y comportamientos son alcanzados utilizando expresiones regulares. La creación de índices de URIs sensibles a mayúsculas y minúsculas es deseable para una versión posterior.

En versiones posteriores, se tiene contemplada la interpretación de calificadores como funciones para incrementar el rango de funcionalidades del modelo. Por ejemplo funciones para realizar consultas por rango, negación, etcétera.

6.4. Aplicaciones Basadas en Objetos Remotos

SPyRO está creado para funcionar en aplicaciones de Recuperación de Información. El diseño de SPyRO esta enfocado en bajar las latencias de comunicación entre nodos y optimizar el desempeño utilizando varios formatos de códificación de mensajes. Éstas características son apreciadas en cualquier dominio, por lo que SPyRO es útil en múltiples aplicaciones, aún fuera de la Recuperación de Información.

Para ejemplificar las posibles aplicaciones se muestran algunas de las aplicaciones desarrolladas utilizando SPyRO como base de comunicación entre procesos remotos. Se muestran dos juegos, uno de ellos usa agentes e inteligencia artificial, es llamado “Policias y ladrones”. El segundo es un juego de destreza mental llamado “Rompevideos”. Para finalizar, se muestra el paquete *PassiveDirectory* de administración centralizada de redes.



(a) ***Interfaz gráfica***

(b) ***Interfaz texto***

Figura 6.8: Interfaces de Policias y Ladrones, la interfaz es independiente del código

6.4.1. Juego Policias y Ladrones

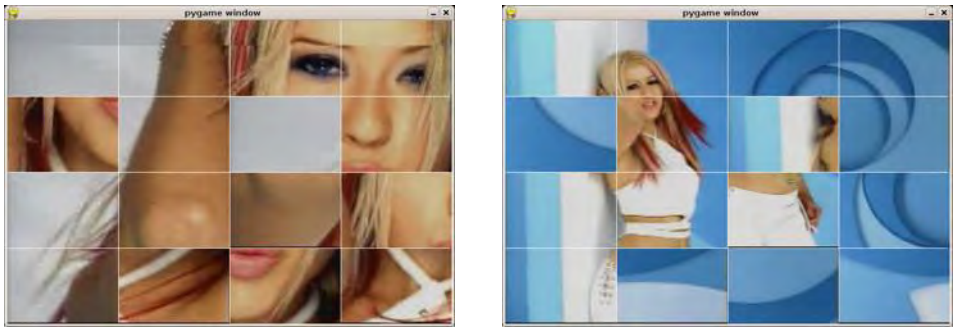
El juego exhibe dos tipos de contrincantes haciendo gala de su "Inteligencia" para conseguir sus objetivos. El juego es independiente de la interfaz, como se muestra en la Figura 6.8.

El juego se desarrolla en un mundo aleatoriamente generado mediante reglas que aseguran conectividada entre los elementos. Las reglas de movimiento y percepción son sencillas:

- Los jugadores solo pueden percibir lo que tienen enfrente, atrás y a los lados. La vista se ve interrumpida por paredes.
- Los jugadores no pueden distinguir la distancia de los objetos.
- Solo pueden hacer un movimiento a la vez.
- Los jugadores actúan de manera independiente, no hay conocimiento compartido entre ellos, ni conocimiento global.
- Cada personaje trata de cumplir sus metas.

Existen dos tipos de jugadores: policias y ladrones. Los policias rondan por el mundo hasta percibir un ladron, entonces cambian sus prioridades y tratan de apresarlos. Los ladrones buscan robar todo el dinero posible y escapar de los policias. Las fuentes se encuentran disponibles bajo la licencia GPL.

SPyRO es utilizado para demostrar la movilidad de código y permitir el uso de interfaces remotas de control. Los jugadores, por tanto, pueden ser locales o remotos. Los jugadores remotos son controlados utilizando objetos remotos de SPyRO, y lógica remota de acuerdo al estado local de los mundos. A su vez, las interfaces de visualización remotas pueden ser diferentes de las locales, esto se debe a que el mundo es un objeto exportable. SPyRO utiliza el modo híbrido de envío de eventos, debido a la necesidad de manipular estados sincronos entre el juego y jugadores.



(a) **(b)**

Figura 6.9: Rompecabezas

6.4.2. Juego Rompevideos

Rompevideos es esencialmente un rompecabezas de videos. El juego se muestra en la Figura 6.9.

El juego consiste en crear divisiones en pantalla, las divisiones representan piezas del rompecabezas. Las piezas se desordenan, y el objetivo del juego es re-ordenar la pantalla. La complicación del juego se debe a que las imágenes mostradas por los videos cambian a través del tiempo, por lo que una gran concentración y memoria es requerida para éste juego. La complejidad aumenta a medida que aumenta el número de particiones en la pantalla. El programa esta escrito en Python usando pygame y soporta videos en formato MPEG. El programa esta disponible bajo la licencia GPL.

El programa es ejecutado mediante la línea de comandos (una pequeña interfaz en *zenity* está disponible), dando como argumentos la lista de videos a reproducir.

SPyRO es utilizado para el juego en red. El juego en red puede ser colaborativo o competitivo. En el juego colaborativo dos o más jugadores unen esfuerzos para completar un único tablero de Rompevideos. Para el juego en competencia, varios jugadores comienzan con un tablero desordenado y tratan de resolver el tablero antes que cualquier otro jugador lo haga. El juego para un participante no necesita de SPyRO.

6.4.3. Administración de Redes

El programa maneja permisos de acceso a la Internet en una subred. El programa recibe un identificador de usuario, dirección IP de procedencia y de acuerdo a una base de datos de permisos, permite la entrada y salida a puertos seleccionados para la persona en la computadora anfitriona. El programa está destinado a controlar el acceso a Internet en redes accesibles físicamente (i.e. en conjunto con DHCPP [Bejar05] es capaz de mejorar la protección de las redes, acceso controlado en redes inalámbricas, etcétera).

6.5. Sumario

Éste Capítulo fue dedicado a las aplicaciones generadas a partir de ésta tesis, es decir, de la implementación de la Arquitectura de Recuperación de Información en Ambientes Distribuidos. La implementación de la arquitectura es la base del paquete SPyDI. SPyDI está constituido por la unión del módulo de indexamiento-consulta y el módulo de adquisición de datos. La unión de los módulos es realizada utilizando SPyRO como mediador. Para transformar SPyDI en una herramienta de desarrollo para la construcción de aplicaciones de Recuperación de Información y otras Tecnologías de Información. En el paquete se incluyen ambientes de desarrollo web y servidores autónomos para una independencia total. También se incluye un nuevo tipo de página web con python empotrado llamada página pyew, la cual tiene como finalidad acelerar el proceso de desarrollo e integración de SPyDI con los servidores web. SPyDI permite la creación de sitios y máquinas de búsqueda de una manera rápida y sencilla.

Sin embargo, SPyDI carece de cifrado de datos en repositorios y de submódulos de alto desempeño. También carece de varios niveles de colaboración y de reconfiguración dinámica de topologías.

A partir de SPyDI se crean dos nuevas aplicaciones “Fully Federated Academic Information retrieval System” (AFFAIRS) y “Universal Index”. AFFAIRS es un SRI especializado a datos académicos, está diseñado para ser alojado en servidores autónomos e independientes. Cada servidor puede ser la estación de trabajo de un investigador o un servidor departamental, a nivel facultad, universidad o conjunto de universidades. La im-

plementación de AFFAIRS aún no soporta deducciones basadas en ontologías. Además, el identificador de entidades en los textos académicos se encuentra en un estado muy inmaduro. No obstante, AFFAIRS es un sistema funcional en donde es posible recuperar información a partir de texto completo o relaciones a partir de citas.

Se muestra a U-Index, que en adición a las características generales de SPyDI, se realiza el enriquecimiento de los documentos utilizando un conjunto de palabras especiales llamadas *calificadores de documento*. Los calificadores de documento son interpretados como modificadores de los documentos, asociando propiedades especiales a estos mismos. La idea central es recuperar documentos basados en dos operaciones diferentes de similitud: similitud de calificadores y similitud de contenido contra consulta. La similitud de calificadores es realizada por operaciones booleanas y mediante funciones más complejas como el *modelo vectorial* [Baeza-Yates99], mientras que la similitud de contenido contra consulta puede ser cualquier tipo función de similitud. El prototipo soporta el modelo booleano y en parte el modelo vectorial de calificadores.

También se muestran una serie de aplicaciones creadas fuera del área de Recuperación de Información, que van del área de Juegos Computarizados al área de Seguridad Informática. Mostrando la versatilidad de las herramientas creadas a partir de ésta tesis.

Capítulo 7

Conclusiones y Trabajo a Futuro

Se diseño e implementó una plataforma estable, modular, de arquitectura y código abierto, enfocada al desarrollo e investigación de Sistemas de Recuperación de Información Distribuida. El prototipo implementado fue llamado SPyDI, el cual esta formado por un módulo de adquisición de datos, indexador, crawler y un conjunto de aplicaciones (servidor web, páginas web embebidas, cola de correo electrónico, etcétera) que convierten a SPyDI en una plataforma autonoma de desarrollo de Recuperación de Información y Web.

En el prototipo se incluye soporte para Recuperación de Información en redes P2P, la cual es una de las áreas de gran interés, debido a las peculiaridades resultantes de la fusión de ambas áreas.

Se ha creado un módulo de adquisición de datos que soporta su distribución a lo largo de una red. Junto al módulo, se propone un esquema de colaboración entre adquisidores de datos, así como clientes que permiten modificar y monitorizar el desempeño de los módulos. El adquisidor de datos soporta y utiliza hilos de manera agresiva (de 50 a 200 hilos concurrentes) para eliminar la mayoría de tiempos muertos y evitando de que los los discos duros y la red se conviertan en cuellos de botella.

Los repositorios diseñados son capaces de replicar los documentos para evitar la perdida de información. Los repositorios utilizan compresión de documentos, y eliminan los requerimientos de un bloque mínimo en el tamaño de archivo, ahorrando espacio. Además, los repositorios soportan colecciones dinámicas, soportan inserciones, borrado y recupera-

ción de documentos.

En el módulo de indexamiento y consulta se presento una arquitectura capaz de realizar indexamiento distribuida y resolución de consultas distribuidas (por balanceo de carga) o colaborativas. El comportamiento es dependiente de las topologías y políticas de resolución adoptadas, es necesario recalcar que es posible utilizar ambas políticas de manera híbrida. La recuperación cooperativa es realizada mediante relevancia de subcolecciones. El módulo está implementado utilizando estructuras de datos que soportan el indexamiento de colecciones dinámicas, y permite realizar consultas a a su vez que se realizan indexamientos. El índice se encuentra "siempre al día", por lo que un documento que se incluye en el índice es inmediatamente disponible para búsqueda.

El módulo de indexamiento y consulta hace uso excesivo de hilos, por lo que se propone el uso de una matriz de exclusores mutuos. Asegurando pocos interbloqueos para casos promedios. Las estructuras del índice invertido están construidas sobre el sistema de caches *cache1*, que también fue diseñado especialmente para la Recuperación de Información.

La arquitectura propuesta está preparada para ser distribuida a través de una red de computadoras. De manera que pueda ser utilizada en redes colaborativas obteniendo obteniendo redes sin servidores centrales o en redes de alto desempeño utilizando servidores centralizados como balance de carga. La arquitectura está diseñada especialmente para crear Sistemas de Recuperación de Información con grandes capacidades de escalamiento, tolerancia a fallos, colaboración entre nodos y utilización sobre redes heterogéneas, ayudando a reducir costos.

Los algoritmos de recuperación de información han sido adaptados para su utilización en redes distribuidas.

Se crearon varias especializaciones de la arquitectura implementada resultando en aplicaciones índependientes con las capacidades de SPyDI. Las aplicaciones incluyen a AFFAIRS (A Fully Federated Academic Information Retrieval System) y UIndex (Universal Index).

También se diseño e implementó una plataforma de objetos remotos de alto rendimiento y baja latencia llamada SPyRO. Además, SPyRO se integró de manera transparente dentro de Python, haciendo posible la comunicación entre computadoras remotas a través

de objetos remotos. Además, una nueva plataforma para cómputo distribuido es creada, ya que se permite la ejecución de métodos remotos. La integración por medio de diferentes modos de envío permite el control de la latencia del protocolo, así como la transparencia de SPyRO permite la creación de aplicaciones con código móvil. SPyRO es apto para la creación de aplicaciones en dispositivos pervasivos, así como aplicaciones en grandes servidores corporativos.

El diseño multiformato de SPyRO permite la comunicación entre clientes no-SPyRO, por lo que SPyRO es un ORB políglota. Además, gracias a ésta característica, es posible alcanzar estados de red de costo mínimo para los recursos de red disponibles, tal y como fue probado para el diseño de SPyRO.

SPyRO ha probado ser de gran eficacia tanto para las tareas de recuperación de información como para otras áreas de desarrollo, por ejemplo, la implementación (llamada SPyDI) esta completamente unida a través de los objetos remotos de SPyRO, esto permite una alta modularidad e independencia entre los módulos sin perder la alta integración. Puede parecer paradójico, pero es una realidad.

Además para mostrar las capacidades de SPyRO se han desarrollado algunas aplicaciones no relacionadas con la recuperación, las cuales se mostraron y consisten principalmente en juegos y herramientas de administración, sin limitarse a estos dominios.

SPyRO es de gran importancia para el desarrollo rápido de prototipos, así como para realizar comunicación eficiente entre objetos en aplicaciones remotas. SPyRO es útil en todo tipo de aplicaciones de red como agentes, aplicaciones cliente-servidor, P2P, distribuidas, federaciones, etcétera.

7.1. Trabajo a Futuro

Existe una gran cantidad de trabajo a futuro dentro de SPyDI y la arquitectura. En la arquitectura, es necesario realizar especificaciones más a detalle utilizando esquemas de ingeniería de software, permitiendo la revisión y critica de expertos en cada uno de los temas selectos que se incluyen en la arquitectura.

En la implementación de los módulos es necesario localizar puntos de alta concen-

tración de procesamiento para proceder a su optimización.

En SPyRO es necesario completar el módulo de integración a apache a bajo nivel utilizando mod_python. Los cortafuegos deben ser estudiados a mayor detalle, y tratar de traspasarlos para incrementar el número de posibles clientes en el sistema.

El módulo de adquisición de datos requiere de un crawler que permita adquisición focalizada de documentos y predicción de importancia de obtención para mejorar la lógica de control de la cola de los cosechadores.

Es necesario adaptar los repositorios para colecciones estáticas, ya que el uso de estás lleva a un alto desempeño. El uso de compresión en el repositorio con claves globales o por bloques incrementa el índice de compresión del repositorio, optimizando el espacio disponible en los repositorios. Esta demostrado que la replicación por bloques es aún más efectiva que la replicación por documento para aplicaciones de alto desempeño.

Los índexadores deben tener soporte para colecciones estáticas de documentos, permitiendo índices más pequeños debido a la posibilidad de utilización de índices comprimidos. La compresión de índices no está implementada, por lo que es necesario en versiones posteriores realizarla. Desgraciadamente, los métodos de compresión de índice implican colecciones estáticas y por lo tanto reconstrucción de al menos una gran parte del índice si la colección crece. A cambio se obtiene un mejor desempeño en tiempo de resolución de consultas y espacialmente. La resolución de consultas por proximidad podría implementarse para permitir búsqueda de frases de manera eficiente.

En cuanto a la forma de interconectar los nodos es posible mejorar el obteniendo topologías o interconexiones de manera dinámica utilizando información geográfica o de estados de red, además de los algoritmos de enrutamiento tradicionales.

La interfaz de SPyDI, así como de las demás aplicaciones creadas, debe ser sometida a estudios de usabilidad para encontrar los posibles errores en la interfaz y arreglarlos para incrementar la facilidad de uso de los sistemas maximizando la efectividad en su utilización con el mínimo esfuerzo por parte de los usuarios. Además las herramientas de configuración deben ser extendidas para soportar mayor usabilidad y rango de configuración.

La capacidad de monitorizar los nodos en un sistema puede ser añadido para

conocer los estados de cada computador y controlar cada uno de los nodos para propósitos de experimentación de nuevos algoritmos y topologías.

A pesar de que SPyRO y SPyDI han sido diseñados para funcionar en ambientes pervasivos, no hay suficientes experimentos para asegurar su correcto funcionamiento en muchas de las plataformas posibles.

Es necesario realizar pruebas de precisión en la información recuperada utilizando las colecciones TREC.

Referencias

[Achour05] Achour, M., Betz, F., Dovgal, A., Lopes, N., Olson, P., Richter, G., Seguy, D., y Vrana, J. *PHP Manual*. PHP Documentation Group, http://www.php.net/manual/, jun. 2005.

[Aggarwal01] Aggarwal, C. C., Al-Garawi, F., y Yu, P. S. Intelligent crawling on the world wide web with arbitrary predicates. *En WWW '01: Proceedings of the 10th international conference on World Wide Web*, págs. 96–105. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-348-0. doi: http://doi.acm.org/10.1145/371920.371955.

[Allman03] Allman, M. An evaluation of xml-rpc. *ACM performance evaluation review*, 2003.
URL `citeseer.ist.psu.edu/584180.html`

[Androutsellis-Theotokis04] Androutsellis-Theotokis, S. y Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1041680.1041681.

[Baeza-Yates99] Baeza-Yates, R. A. y Ribeiro-Neto, B. A. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. ISBN 0-201-39829-X.
URL `citeseer.ist.psu.edu/baeza-yates99modern.html`

[Baquero03] Baquero, C. y Lopes, N. Towards peer-to-peer content indexing. *SIGOPS Oper. Syst. Rev.*, 37(4):90–96, 2003. ISSN 0163-5980. doi:http://doi.acm.org/10.1145/958965.958974.

[Barroso03] Barroso, L. A., Dean, J., y Hözle, U. Web search for a planet: The google cluster architecture. *IEEE Micro*, págs. 22–29, March-April 2003.

[Batory81] Batory, D. S. B+ trees and indexed sequential files: a performance comparison. *En SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, págs. 30–39. ACM Press, New York, NY, USA, 1981. ISBN 0-89791-040-0. doi: http://doi.acm.org/10.1145/582318.582323.

[Bawa03] Bawa, M., Manku, G. S., y Raghavan, P. Sets: search enhanced by topic segmentation. *En SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, págs. 306–313. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-646-3. doi: http://doi.acm.org/10.1145/860435.860491.

[Bejar05] Bejar, J. O. *Administración del Sistema DHCP*. Tesis Doctoral, Facultad de Ingeniería Eléctrica. UMSNH, jul. 2005.

[Bell95] Bell, T. C., Moffat, A., Witten, I. H., y Zobel, J. The mg retrieval system: compressing for space and speed. *Commun. ACM*, 38(4):41–42, 1995. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/205323.205327.

[Bergmark02] Bergmark, D., Lagoze, C., y Sbityakov, A. Focused crawls, tunneling, and digital libraries. *En ECDL '02: Proceedings of the 6th European Conference on Research and Advanced Techno-*

logy for Digital Libraries, págs. 91–106. Springer-Verlag, London, UK, 2002. ISBN 3-540-44178-6.

[Berners-Lee05] Berners-Lee, T., Fielding, R., y Masinter, L. Uniform resource identifier (uri): Generic syntax. rfc 3986. Available online at http://www.gbiv.com/protocols/uri/rfc/rfc3986.html, ene. 2005.

[Bollacker98] Bollacker, K. D., Lawrence, S., y Giles, C. L. Citeseer: an autonomous web agent for automatic retrieval and identification of interesting publications. *En AGENTS '98: Proceedings of the second international conference on Autonomous agents*, págs. 116–123. ACM Press, New York, NY, USA, 1998. ISBN 0-89791-983-1. doi:http://doi.acm.org/10.1145/280765.280786.

[Bollacker99] Bollacker, K. D., Lawrence, S., y Giles, C. L. A system for automatic personalized tracking of scientific literature on the web. *En DL '99: Proceedings of the fourth ACM conference on Digital libraries*, págs. 105–113. ACM Press, New York, NY, USA, 1999. ISBN 1-58113-145-3. doi: http://doi.acm.org/10.1145/313238.313270.

[Bowman95] Bowman, C., Danzig, P., Hardy, D., Manber, U., Schwartz, M., y Wessels, D. Harvest: A scalable, customizable discovery and access system. *En Proceedings of the Seventh International Web*. March 1995.

[Brin98] Brin, S. y Page, L. The anatomy of a large-scale hypertextual web search engine. *En Proceedings of the Seventh International Web*. 1998.
URL `http://gunther.smeal.psu.edu/brin98anatomy.html`

[Casanova02] Casanova, H. Distributed computing research issues in grid

computing. *SIGACT News*, 33(3):50–70, 2002. ISSN 0163-5700. doi:http://doi.acm.org/10.1145/582475.582486.

[Chakrabarti99] Chakrabarti, S., van den Berg, M., y Dom, B. Focused crawling: a new approach to topic-specific web resource discovery. *En Proceedings of the 8th International Conference on The World-Wide Web)*. 1999.

[Chakrabarti02] Chakrabarti, S., Punera, K., y Subramanyam, M. Accelerated focused crawling through online relevance feedback. *En WWW '02: Proceedings of the 11th international conference on World Wide Web*, págs. 148–159. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-449-5. doi: http://doi.acm.org/10.1145/511446.511466.

[Cho04] Cho, J., Garcia-Molina, H., Haveliwala, T., Lam, W., Paepcke, A., Raghavan, S., y Wesley, G. Stanford webbase components and applications. Inf. téc., Stanford University, jul. 2004.

[Clarke01] Clarke, I., Sandberg, O., Wiley, B., y Hong, T. W. Freenet: a distributed anonymous information storage and retrieval system. *En International workshop on Designing privacy enhancing technologies*, págs. 46–66. Springer-Verlag New York, Inc., New York, NY, USA, 2001. ISBN 3-540-41724-9.

[Clarke02] Clarke, I., Miller, S. G., Hong, T. W., Sandberg, O., y Wiley, B. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/4236.978368.

[Cohen03] Cohen, B. Incentives build robustness in bittorrent. *En http://www.bittorrent.com/bittorrentecon.pdf.* mayo 2003.

[Comer79] Comer, D. Ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137, 1979. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/356770.356776.

[Conrad97] Conrad, S., Eaglestone, B., Hasselbring, W., Roantgree, M., Saltor, F., Schönhoff, M., Strässler, M., y Vermeer, M. Research issues in federated database systems. *En Report of EFDBS '97 Workshop*, tomo 26. Barcelona, España, December 1997.

[Consortium] Consortium, W. W. W. W. Http - hypertext trasfer protocol; available online at http://www.w3c.org/protocols.

[Cormen01] Cormen, T. H., Leiserson, C., Rivest, R. L., y Stein, C. E. L. C. *Introduction to Algorithms*. McGraw-Hill, Inc., New York, NY, USA, second edition ed$^{\underline{\text{ón}}}$., 2001. ISBN 0070131430.

[Corporation] Corporation, Z. Zope, the open source web application server; zope homepage http://www.zope.org.

[Crainiceanu04] Crainiceanu, A., Linga, P., Machanavajjhala, A., Gehrke, J., y Shanmugasundaram, J. An indexing framework for peer-to-peer systems. *En SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, págs. 939–940. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007707.

[Craswell00] Craswell, N. E. *Methods for Distributed Information Retrieval*. Tesis Doctoral, The Australian National University, mayo 2000.

[Cutting05] Cutting, D. y para el Software Apache, F. Lucene. available online at http://lucene.apache.org, 1998-2005.

[D. Robinson04] D. Robinson, K. C. The common gateway inter-

face (cgi) version 1.1 rfc 3875; available online at http://www.faqs.org/rfcs/rfc3875.html, October 2004.

[Danzig91] Danzig, P. B., Ahn, J., Noll, J., y Obraczka, K. Distributed indexing: a scalable mechanism for distributed information retrieval. *En SIGIR '91*, págs. 220–229. ACM Press, New York, NY, USA, 1991. ISBN 0-89791-448-1. doi: http://doi.acm.org/10.1145/122860.122883.

[deJong05] de Jong, I. Python remote objects (pyro) webpage, http://www.sourceforge.net/projects/pyro, feb. 2005.

[Dumais88] Dumais, S. T., Furnas, G. W., Landauer, T. K., Deerwester, S., y Harshman, R. Using latent semantic analysis to improve access to textual information. *En Proceedings of the Conference on Human Factors in Computing Systems CHI'88*. 1988.
URL `citeseer.ist.psu.edu/dumais88using.html`

[eDevelopment Team05] e Development Team, S. Swish-e: Simple web indexing system for humans: Enhanced. available online at http://swish-e.org, 1996-2005.

[Esterbrook01] Esterbrook, C. Introduction to webware. *En Proceedings of the Ninth International Python Conference*. 2001.
URL `http://www.webwareforpython.org/Papers/IntroToWebware.html`

[etal.a] et al., D. B. Resource description framework (rdf); available online at http://www.w3.org/rdf.

[etalb] et al, G. B.-D. Rdf site summary (rss) 1.0; available online at http://purl.org/rss/1.0/spec.

[Fielding02] Fielding, R. T. y Taylor, R.Ñ. Principled design of the modern web architecture. *ACM Trans. In-*

ter. Tech., 2(2):115–150, 2002. ISSN 1533-5399. doi: http://doi.acm.org/10.1145/514183.514185.

[Florescu98] Florescu, D., Levy, A., y Mendelzon, A. Database techniques for the world-wide web: a survey. *SIGMOD Rec.*, 27(3):59–74, 1998. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/290593.290605.

[Foster98] Foster, I. y Karonis, N. T. A grid-enabled mpi: message passing in heterogeneous distributed computing systems. *En Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, págs. 1–11. IEEE Computer Society, Washington, DC, USA, 1998. ISBN 0-89791-984-X.

[Foundation05] Foundation, P. S. Python homepage http://www.python.org, 2005.

[Frattolillo05] Frattolillo, F. Running large-scale applications on cluster grids. *Int. J. High Perform. Comput. Appl.*, 19(2):157–172, 2005. ISSN 1094-3420. doi: http://dx.doi.org/10.1177/1094342005054261.

[Fuggetta98] Fuggetta, A., Picco, G. P., y Vigna, G. Undestanding code mobility. *IEEE Transactions on Software Engineerin*, 24(5), mayo 1998.

[Ghemawat03] Ghemawat, S., Gobioff, H., y Leung, S.-T. The google file system. *En SOSP '03*, págs. 29–43. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-757-5. doi: http://doi.acm.org/10.1145/945445.945450.

[Google-Inc] Google-Inc. Scholagoogle webpage, http://scholar.google.com.

[Graham89] Graham, R. L., Knuth, D. E., y Patashnik, O. *Concrete mathematics: a foundation for computer science.* Addison-Wesley

Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0-201-14236-9.

[Group04] Group, T. H. Ht://dig internet search engine. available online at http://www.htdig.org/, 1995–2004.

[Hafri04] Hafri, Y. y Djeraba, C. High performance crawling system. *En MIR '04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, págs. 299–306. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-940-3. doi:http://doi.acm.org/10.1145/1026711.1026760.

[Hardy96] Hardy, D. R. y Schwartz, M. F. Customized information extraction as a basis for resource discovery. *ACM Trans. Comput. Syst.*, 14(2):171–199, 1996. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/227695.227697.

[Hasselbring00] Hasselbring, W., van den Heuvel, W.-J., Houben, G., Kutsche, R.-D., Rieger, B., Roantree, M., y Subieta, K. Research and practice in federated information systems. *En Report of EFIS '2000 International Workshop*, tomo 29. Dublin, jun. 2000.

[Hiemstra00] Hiemstra, D. *Using Language Models For Information Retrieval.* Tesis Doctoral, Centre for Telematics and Information Retrieval, Universidad Twente, 2000.

[Hsiao92] Hsiao, D. K. Federated databases and systems: Part i - a tutorial on their data sharing. *VLDB Journal*, 1:127–179, 1992.

[Inc.05] Inc., M. Asp web page http://www.asp.net, 2005.

[Jacob Ziv77] Jacob Ziv, A. L. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.
URL `citeseer.ist.psu.edu/ziv77universal.html`

[Johnson93] Johnson, T. y Sasha, D. The performance of current b-tree algorithms. *ACM Trans. Database Syst.*, 18(1):51–101, 1993. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/151284.151286.

[Kan01] Kan, G. *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly, 2001.

[Kazaa05] Kazaa. Kazaa home page http://www.kazaa.com, 2005.

[Kernighan88] Kernighan, B. W. y Ritchie, D. M. *The C programming language*. Prentice Hall, Englewoord Cliffs, N.J., 1988. ISBN 0-131-10370-9.

[Khan00] Khan, L. R. *Ontology-based Information Selection*. Tesis Doctoral, University of Southern California, August 2000.

[Khoussainov04] Khoussainov, R. y Kushmerick, N. Specialisation dynamics in federated web search. *En WIDM '04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, págs. 112–119. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-978-0. doi: http://doi.acm.org/10.1145/1031453.1031474.

[Kim01] Kim, S.-W. y Won, H.-S. Batch-construction of b+-trees. *En SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, págs. 231–235. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-287-5. doi: http://doi.acm.org/10.1145/372202.372329.

[Klampanos04] Klampanos, I. A. y Jose, J. M. An architecture for information retrieval over semi-collaborating peer-to-peer networks. *En SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, págs. 1078–1083. ACM Press,

New York, NY, USA, 2004. ISBN 1-58113-812-1. doi: http://doi.acm.org/10.1145/967900.968119.

[Koftikian01] Koftikian, J. *Project Work: Simple Object Access Protocol (SOAP)*. Studienarbeit, TU Hamburg-Harburg, abr. 2001. URL `http://www.sts.tu-harburg.de/papers/2001/Koft01`

[Kossmann00] Kossmann, D. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000. ISSN 0360-0300. doi:http://doi.acm.org/10.1145/371578.371598.

[Kreger01] Kreger, H. Web services conceptual architecture (wsca 1.0). Inf. téc., IBM Software Group, mayo 2001.

[Lawrence99a] Lawrence, S., Bollacker, K., y Giles, C. L. Distributed error correction. *En Fourth ACM Conference on Digital Libraries*. ACM Press, New York, NY, USA, 1999.

[Lawrence99b] Lawrence, S., Bollacker, K., y Giles, C. L. Indexing and retrieval of scientific literature. *En CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*, págs. 139–146. ACM Press, New York, NY, USA, 1999. ISBN 1-58113-146-1. doi: http://doi.acm.org/10.1145/319950.319970.

[Lewine94] Lewine, D. A. *POSIX programmer's guide: writing portable UNIX programs with the POSIX. 1 standard*. O'Reilly & Associates, Sebastopol, Calif., 1994. ISBN 0-937-17573-0.

[Linga05] Linga, P., Crainiceanu, A., Gehrke, J., y Shanmugasudaram, J. Guaranteeing correctness and availability in p2p range indices. *En SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM Press, New York, NY, USA, jun. 2005.

[MacDonald99] MacDonald, J. y Zhao, B. Y. T-treap and cache performance of indexing data structures. *Available online at* http://www.cs.berkeley.edu/ %7eravenben/research/CS252/252Paper.pdf, dic. 1999.

[Manber90] Manber, U. y Myers, G. Suffix arrays: a new method for on-line string searches. *En SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, págs. 319–327. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990. ISBN 0-89871-251-3.

[Manber94] Manber, U. y Wu, S. GLIMPSE: A tool to search through entire file systems. *En Proceedings of the USENIX Winter 1994 Technical Conference*, págs. 23–32. San Fransisco, CA, USA, 17–21 1994.
URL `citeseer.ist.psu.edu/manber94glimpse.html`

[Manber97] Manber, U. y Gopal, B. Webglimpse — combining browsing and searching. *En Proceedings of the USENIX Winter 1997 Technical Conference*. San Fransisco, CA, USA, ene. 1997.

[Maymounkov02] Maymounkov, P. y Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric, 2002.
URL `citeseer.ist.psu.edu/sex02sex.html`

[McNab95] McNab, D. Spiro - simple python interface to remote objects. http://www.freenet.org.nz/python/spiro/, 1995.

[Melnik01] Melnik, S., Raghavan, S., Yang, B., y Garcia-Molina, H. Building a distributed full-text index for the web. *ACM Transactions Information Systems*, 19(3):217–241, 2001. ISSN 1046-8188. doi:http://doi.acm.org/10.1145/502115.502116.

[Members05] Members, W. World wide web consortium. Available online at http://www.w3.org/, dic. 2005.

[Microsoft] Microsoft. Microsoft web page. http://www.microsoft.com.

[Microsystems] Microsystems, S. Sun microsystems web page. http://www.sun.com.

[Microsystems05] Microsystems, S. Rmi: Java remote method invocation; available online at java.sun.com/products/jdk/rmi/, 2005.

[Nottingham03] Nottingham, M. The atom syndication format 0.3 (pre-draft); available online at http://www.mnot.net/drafts/draft-nottingham-atom-format-02.html, dic. 2003.

[Noy04] Noy, N. F. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, 2004. ISSN 0163-5808. doi:http://doi.acm.org/10.1145/1041410.1041421.

[ofIndexers05] of Indexers, A. S. How information retrieval started, jul. 2005.

[Pandey05] Pandey, S. y Olston, C. User-centric web crawling. *En WWW '05: Proceedings of the 14th international conference on World Wide Web*, págs. 401–411. ACM Press, New York, NY, USA, 2005. ISBN 1-59593-046-9. doi: http://doi.acm.org/10.1145/1060745.1060805.

[Pedroni02] Pedroni, S. y Rappin, N. *Jython Essentials*. O'Reilly, mar. 2002.

[Postel82] Postel, J. B. Simple mail transfer protocol. rfc 821; available online at http://www.faqs.org/rfcs/rfc821.html, ago. 1982.

[Qiu04] Qiu, D. y Srikant, R. Modeling and performance analysis of bittorrent-like peer-to-peer networks. *En SIGCOMM '04: Proceedings of the 2004 conference on Appli-*

cations, technologies, architectures, and protocols for computer communications, págs. 367–378. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-862-8. doi: http://doi.acm.org/10.1145/1015467.1015508.

[Rasolofo01] Rasolofo, Y., Abbaci, F., y Savoy, J. Approaches to collection selection and results merging for distributed information retrieval. *En CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, págs. 191–198. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-436-3. doi:http://doi.acm.org/10.1145/502585.502618.

[Ripeanu01] Ripeanu, M. Peer-to-peer architecture case study: Gnutella network. *En First International Conference on Peer-to-Peer Computing (P2P'01)*, tomo 00, pág. 99. IEEE Computer Society, 2001.

[Server] Server, A. H. http://www.apache.org.

[Sheth90] Sheth, A. P. y Larson, J. A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990. ISSN 0360-0300. doi:http://doi.acm.org/10.1145/96602.96604.

[Si02] Si, L. y Callan, J. Using sampled data and regression to merge search engine results. *En SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, págs. 19–26. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-561-0. doi: http://doi.acm.org/10.1145/564376.564382.

[Snavely03] Snavely, A., Chun, G., Casanova, H., der Wijngaart, R. F. V., y Frumkin, M. A. Benchmarks for grid computing: a review

of ongoing efforts and future directions. *SIGMETRICS Perform. Eval. Rev.*, 30(4):27–32, 2003. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/773056.773062.

[Srinivasan] Srinivasan, R. Rpc: Remote procedure call protocol specification version 2. rfc 1831; available online http://www.faqs.org/rfcs/rfc1831.html.

[Suciu02] Suciu, D. Distributed query evaluation on semistructured data. *ACM Trans. Database Syst.*, 27(1):1–62, 2002. ISSN 0362-5915. doi:http://doi.acm.org/10.1145/507234.507235.

[Sun] Java. http://java.sun.com.

[Sun04] Sun, T., Zhao, W., y Zhao, Z. An architecture of pattern-oriented distributed meta-search engine. *En InfoSecu '04: Proceedings of the 3rd international conference on Information security*, págs. 171–174. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-955-1. doi: http://doi.acm.org/10.1145/1046290.1046325.

[Technologies05a] Technologies, I. S. Blueprint: a universal standard model for efficient information retrieval (omnipaper: Smart access to european newspapers). Inf. téc., Information Society Technologies, feb. 2005.

[Technologies05b] Technologies, I. S. Evaluation and demostration framework (omnipaper: Smart access to european newspapers). Inf. téc., Information Society Technologies, feb. 2005.

[Technologies05c] Technologies, I. S. User interface requirements (omnipaper: Smart access to european newspapers). Inf. téc., Information Society Technologies, feb. 2005.

[Tolvards] Tolvards, L. Linux. http://www.linux.org.

[Toolkit] Toolkit, T. S. T. S. http://twistedmatrix.com/products/spread.

[Trubetskoy03] Trubetskoy, G. Introduction to mod_python; available online at http://www.modpython.org, January 2003.

[Turtle92] Turtle, H. R. y Croft, W. B. A comparison of text retrieval models. *Comput. J.*, 35(3):279–290, 1992. ISSN 0010-4620. doi:http://dx.doi.org/10.1093/comjnl/35.3.279.

[Unix] Unix. Unix. http://www.unix.com.

[vanRossum] van Rossum, G. y Jr., F. L. D. pickle — python object serialization. Python Library Reference.

[Vinoski97] Vinoski, S. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.
URL `citeseer.ist.psu.edu/vinoski97corba.html`

[Witten99] Witten, I. H., Moffat, A., , y Bell, T. C. *Managing Gigabytes: Compressing and Indexing documents and images*. Morgan Kaufmann Publishing, second edition ed$^{\underline{\text{ón}}}$., 1999. ISBN 1558605703.

[Yergeau04] Yergeau, F., Cowan, J., Bray, T., Paoli, J., Sperberg-McQueen, C. M., y Maler, E. Extensible markup language (xml) 1.1, w3c recommendation. Available online at http://www.w3.org/TR/2004/REC-xml11-20040204/, feb. 2004.

[ZeroC05] ZeroC, I. Ice: The internet communication engine, 2005.

[Zhuang05] Zhuang, Z., Wagle, R., y Giles, C. L. What's there and what's not?: focused crawling for missing documents in digital libraries. *En JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, págs. 301–310. ACM

Press, New York, NY, USA, 2005. ISBN 1-58113-876-8. doi: http://doi.acm.org/10.1145/1065385.1065455.

Glosario

Agregador Un programa que reúne varias fuentes de sindicación para unirlas y presentarlas en una sola página o interfaz, 96

Ambiente Pervasivo Es referido pervasivo como aquellos dispositivos que se introducen dentro de la vida cotidiana de un individuo, tal como una PDA, teléfonos celulares, etcéterea, 3

API Interfaz de programación de aplicaciones, 79

Archivos Dispersos Un archivo disperso es un archivo en el cual es posible acceder a localidades en un archivo que aún no han sido creadas, y escribir en localidades no contiguas a previas escrituras, 80

Atom Formato popular para sindicación de contenido, trata de resolver las limitaciones de RSS en las versiones anteriores a la 2.0, 95

BFS Breath First Search, 60

Categorizador Sistema de Recuperación de Información que es capaz de agrupar por categorías una colección de documentos, 60

Cortafuegos Los cortafuegos (*firewalls*) son sistemas de cómputo dedicados a imponer políticas de acceso a redes, generalmente a nivel del Protocolo de Internet IP., 37

Crawler Un crawler o *araña web* es un sistema de recuperación de información especializado en la adquisición de datos mediante el método *pull*. Esencialmente un crawler recorre la Web a través de los hipervínculos obteniendo una serie de documentos relevantes a algún criterio, 5

Crawling Acción de recorrer la Web por medio una araña, recolectando documentos y extrayendo las ligas en las páginas Web, 53

DFS Depth First Search, 60

DHT Tablas Hash Distribuidas, 7

ER Envío de Objetos por Referencia, 39

EV Envío de Objetos por Valor, 39

FIFO Estructura de datos en donde lo primero que entra es lo primero que sale, el nombre esta determinado por sus siglas en inglés de *First In First Out*, 55

Hilo Un hilo o hebra (también es muy común llamarlos por su nombre en inglés: *thread*) son secuencias de instrucciones ejecutadas paralelamente con otras secuencias, 36

Hipertexto Es texto semi-estructurado con enlaces entre documentos. Los documentos enlazados pueden ser documentos de hipertexto o cualquier tipo de documento (audio, vídeo, datos en general, texto, etc.), 68

Kernel Núcleo de un Sistema Operativo, 36

Método pull El método *pull* es utilizado para la obtención de datos en un sistema, recogiendo los datos de su fuente original., 51

Método push El método *push* es utilizado para la obtención de datos en un sistema. Básicamente, las fuente que contienen los datos envían directamente los datos al sistema, es opuesto al método *pull*., 51

MEO Modo de Envío de Objetos, 39

NAT Es el acrónimo en el idioma inglés de Translación de Direcciones de Red. Es un estándar de Internet que le permite a una red local (LAN) usar un grupo de direcciones IP para el tráfico interno y otro grupo de direcciones para el tráfico externo., 40

ORB Object Request Broker, 34

P2P Los sistemas P2P, o *Peer-to-peer*, son sistemas distribuidos tolerantes a fallas y de gran escalabilidad. En un enfoque purista, una red P2P no necesita de servicios centralizados y los nodos no requieren información global completa. Todos los nodos en la red P2P tienen la misma importancia., 6

RDF Resource Description Framework, 95

RI Recuperación de Información, 11

RSS Formato popular de sindicación, a lo largo de sus diferentes versiones sus siglas han cambiado de significado, 95

Sistema de Archivos *proc* El sistema de Archivos *proc* es una interfaz de comunicación directa con el Kernel. Permite modificar parámetros internos del Sistema Operativo, 36

SRI Sistema de Recuperación de Información, 11

SRI Sistemas de Recuperación de Información, 1

URI Identificador Uniforme de Recursos, Uniform Resource Identificator, 55

WWW La *World Wide Web* (Telaraña mundial) o simplemente "la web" es un sistema de información basado en *hipertexto*. Los contenidos pueden tener cualquier formato, y son accesibles para los usuarios mediante programas especiales, siendo los más populares los navegadores de la Web., 68