WEBBASE

Daniel Olmedilla

Abstract

The Stanford WebBase project is investigating various issues in crawling, storage, indexing, and querying of large collections of Web pages. The project builds on the previous Google activity that was part of the DLI1 initiative. The DLI2 WebBase project aims to build the necessary infrastructure to facilitate the development and testing of new algorithms for clustering, searching, mining, and classification of Web content.

1 Overview

The Stanford WebBase project is investigating various issues in crawling, storage, indexing, and querying of large collections of Web pages. The project builds on the previous Google activity that was part of the DLI1 initiative. The DLI2 WebBase project aims to build the necessary infrastructure to facilitate the development and testing of new algorithms for clustering, searching, mining, and classification of Web content.

1.1 Goals

The WebBase project has the following goals:

Provide a storage infrastructure for Web-like content

Store a sizeable portion of the Web

Enable researchers to easily build indexes of page features across large sets of pages

Distribute Webbase content via multicast channels

Support structure and content-based querying over the stored collection

1.2 Challenges

The huge size of the information space (well over two billion pages according to recent estimates) and the need to work with limited resources (disk space, memory, time, bandwidth, etc.) are significant global challenges that must be confronted by each component of WebBase. In addition, the project is looking at the following issues:

- Continuous update and maintenance
  - Smart crawling to maximize freshness and relevance
  - Crawler parallelism
  - Crawling robustness (interruptible crawls)
- Efficient and scalable indexing
  - Web-scale indexes over textual content
  - Indexes over hyperlinked graph structure
- Easy access to stored collection
  - Collection access API
  - Efficient query-based retrieval
  - API for building new indexes
  - Convenient wide-area distribution of entire content

1.3 Architecture

The WebBase project employs the architecture shown in Figure 1. The important components of this architecture are described below.

- **Crawler:** "Smart" crawling technology is used to crawl 'valuable' sites more frequently or more deeply. The measure of 'value' is, of course, itself an important research topic. Smart crawling is also used to estimate the rate of change of web pages and adjust the crawling algorithm to maximize the *freshness* of the pages in the repository.
- **Page repository:** A scalable distributed repository is used to store the crawled collection of Web pages. Strategies for physical organization of pages on the storage devices, distribution of pages across machines, and mechanisms to integrate freshly crawled pages, are important issues in the design of this repository. The repository supports both random and

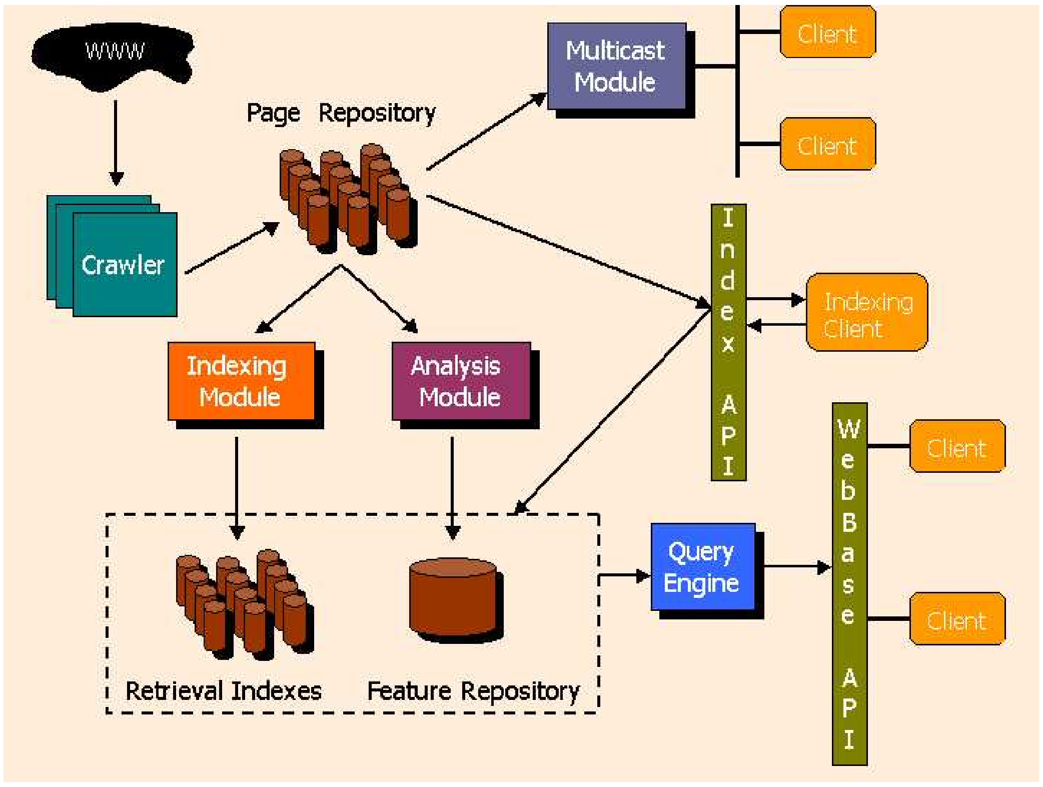

Figure 1: Webbase Architecture

stream-based access modes. Random access allows individual pages to be retrieved based on an internal page identifier. Stream-based access allows all or a significant subset of pages to be retrieved as a stream.

- **Indexing and Analysis Modules:** The indexing module uses a stream of pages from the repository to build basic retrieval indexes over the content and structure of the crawled collection. Indexes on the content are akin to standard IR-style text indexes, whereas indexes on the structure represent the hyperlinked Web graph. Efficient construction of such indexes over Web-scale collections poses interesting challenges.The analysis module populates the feature repository by using a collection of *feature extraction engines*. Each feature extraction engine computes some property of a page (example: reading level, genre), builds an index over that property, and makes that index available to the rest of WebBase. These indexes will be kept up to date as the archive is refreshed through new crawls. The set of feature extraction engines will grow, as new algorithms are devised to describe and characterize Web pages.

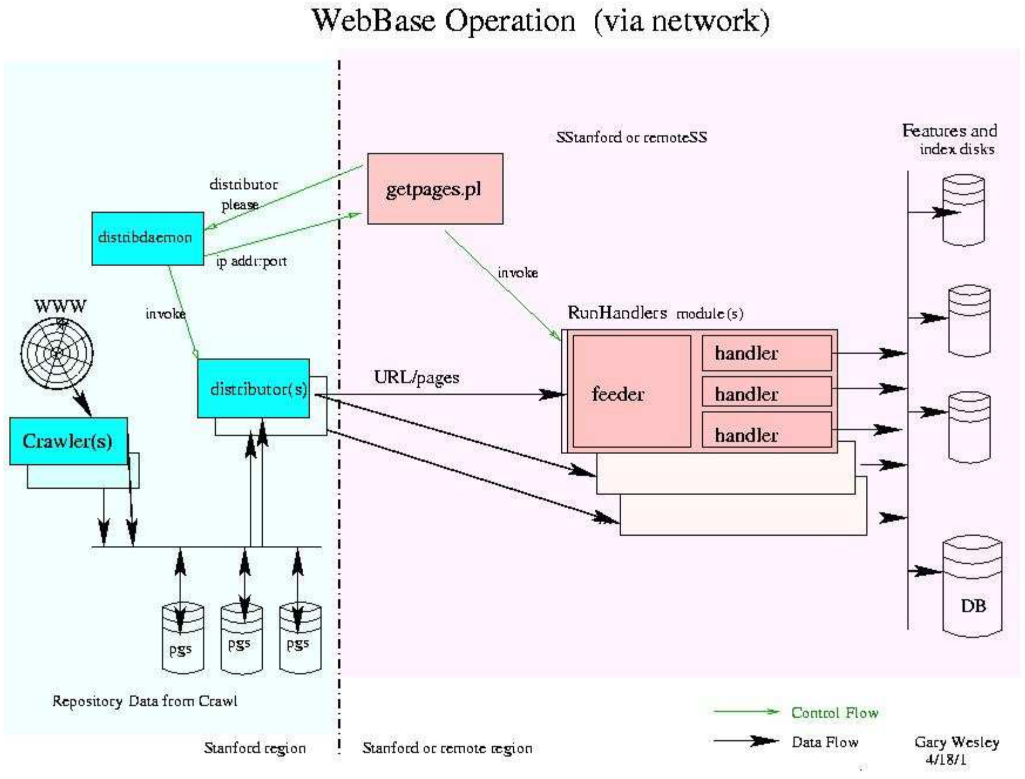

Figure 2: WebBase index factory diagram

In addition, indexes can also be built off-site by indexing clients that implement the "Index API". These indexing clients receive a page stream, use that to build the index files, and integrate the index into WebBase using the "Index API."

- **Multicast module:** The distribution machinery implemented by the multicast module will allow researchers everywhere to take advantage of our collected data. Rather than forcing all index creation and data analysis to run at the site where the data is located, our wide-area data distribution facility will multicast the WebBase content through multicast channels. Channels may vary in bandwidth and content. Subscribers specify the parameters of their channel, including the subset of pages that are of interest to them.

- **Query Engine:** Query-based access to the pages and the computed features (from the feature repository) is provided via the WebBase query engine. Unlike the traditional keyword-based queries supported by existing search engines, queries to the WebBase query engine can involve predicates on both the content and link structure of the Web pages.

2 Requirements

Linux Operating System (Suse Linux 7.3 distribution recommended) with the next packages

- perl (e.g. v5.6.1-41): Perl Interpreter
- perl-DBI (e.g. v1.19-17): Perl Database Interface
- perl-DBD-Pg (e.g. v1.01-1)
- perl-libwww-perl (e.g. v5.53-17): modules Providing API to WWW
- perl-SQL-Statement (e.g. v0.1017-84): SQL parsing and processing engine
- database system (postgreSQL recommended)
  - postgresql (e.g. v7.1.3-11): PostgreSQL - the Database
  - postgresql-server (e.g. v7.1.3-11): the programs needed to create and run a PostgreSQL server
  - postgresql-perl (e.g. v7.1.3-11): development module needed for Perl code to access a PostgreSQL DB
  - postgresql-libs (e.g. v7.1.3-11): the shared libraries required for any PostgreSQL clients
  - PgAccess (e.g. v0.98.8)(optional): visual database administrator
  - postgresql_autodoc (e.g. v1.00)(optional): automatic database diagram generation
- PHP4-enabled Web Server (v4.1.2 at least recommended to avoid a php security issue)
- In order to compile sourcecode:
  - gcc (earlier than version 3.0. version 2.9x recommended)
  - gmake (e.g. v3.79.1)
  - w3c-libwww (e.g. v5.4.0): the client code needs files libwwwcore.* and libwwwutils.* (http://www.w3.org/Library/)

3 Installing and running Webbase

Be sure you have the packages mentioned above before you start this section.

3.1 Crawler

3.1.1 Crawler quickstart

For more information on this portion, see crawler/README:

1. Edit the HTTP/1.0 headers in crawler/crawl_server/fetch.cc:367 so inquiries and complaints go to the correct email address (the address of the crawler operator). The crawler calls itself "Pita", and declares that it runs on behalf of "webmaster@pita.stanford.edu". These are specified in the HTTP/1.0 request headers sent in crawler/crawl_server/fetch.cc:367. The HTTP spec says that robots (such as this one) should use their From: header to identify the user who is running the robot. The From: line should be a machine readable email address (RFC 822/RFC 1123 compliant). As a courtesy to Web servers that don't log From: headers, the email address is also duplicated as a comment in the User-agent: header line. The comment (the email address) must be surrounded by parentheses '(' and ')' to conform to HTTP spec.

2. In this (WebBase/) directory, './configure' and 'make crawler'.
   For more information on this portion, see crawler/crawl_binaries/README, starting at "Instructions to run the crawler":

3. chdir crawler/crawl_binaries/

4. Edit crawl_config and dir.config.* (which tells each crawler process where to write its output). Create also the directories specified in dir.config.* for crawler output.

5. mkdir log/ (if it doesn't already exist)

   There is no bundled information for this portion:

6. If this is the first time you will run the crawler, create the database and the tables you will need.

   (a) Login as a database user (normally postgres user is used).

   (b) Creating a database cluster: initdb -D [destination] (e.g. /usr/local/pgsql/data)

   (c) Start the database server: postmaster -i -D [destination] ¿ [logFile] 2¿&1 &

(d) Create the dataase: createdb [databaseName] (e.g. webbase)

(e) Access to the database (optional): psql [databaseName]

(f) Create the basic tables (the file ~/crawler/crawl-binaries/crawl_table_setup.sql could be used for it): psql [databaseName] -e -f [file]

-
  ```
  CREATE TABLE hosts(
  hid BIGINT PRIMARY KEY,
  hostname character varying(1024) unique not null,
  priority real not null,
  minpages integer,
  maxpages integer not null,
  mindepth integer,
  maxdepth integer not null,
  imagesp boolean,
  pdfsp boolean,
  audiop boolean,
  minpause integer,
  rooturls text);
  ```

-
  ```
  CREATE TABLE aliases (
  hid bigint not null,
  alias character varying (1024) not null);
  ```

-
  ```
  CREATE INDEX aliases_hid_skey on aliases (hid);
  ```

-
  ```
  CREATE TABLE ipaddrs(
  hid bigint not null,
  ipaddr inet not null);
  ```

-
  ```
  CREATE INDEX ipaddrs_hid_skey on ipaddrs (hid);
  ```

-
  ```
  CREATE TABLE uncrawled (
  hid BIGINT PRIMARY KEY,
  ipaddr inet NOT NULL);
  ```

-
  ```
  CREATE INDEX uncrawled_hid_skey on uncrawled (hid);
  ```

-
  ```
  CREATE TABLE crawling (
  hid BIGINT PRIMARY KEY,
  id integer NOT NULL, ipaddr inet NOT NULL);
  ```

-
  ```
  CREATE INDEX crawling_ipaddr_skey on crawling(ipaddr);
  ```

-
  ```
  CREATE TABLE crawled (
  hid bigint primary key,
  id integer,
  status varchar(1024));
  ```

- – PostgreSQL only:
  vacuum analyze

7. Load a database with Web sites to crawl, and edit crawl_buddy.pl:44-49 to point to the database (fill in the table "hosts"

   (a) Fill in the table "hosts"

   - Modify crawler/seed_list/load-aliases.pl:35 (database, host, user and password)
   - Create a file with the hosts we want to crawl. One each line and with a complete URL
   - Use extract-hosts.pl, resolve.pl, merge-aliases.pl, prune-aliases.pl and load-aliases.pl to load the table hosts:
     i. cat [hostsFile] — perl extract-hosts.pl — perl resolve.pl — perl merge-aliases.pl — perl prune-aliases.pl ¿ [table.sql]
     ii. If this is the first time modified load-aliases.pl in lines 38, 39 ("INSERT INTO Hosts(hid, hostname, priority, maxpages, maxdepth, minpause) VALUES (?, ?, ?, 20000, 10, [millisecondsBetweenRequests]", 500 or 1000 would be enough)
     iii. cat table.sql — perl load-aliases.pl

   (b) Fill in the table uncrawled from table hosts:

   - Delete previous content (if any and needed)
     DELETE FROM uncrawled ;
   - Insert new data from table "hosts" INSERT INTO uncrawled (hid, ipaddr) (SELECT h.hid, (SELECT ipaddr FROM ipaddrs WHERE hid = h.hid ORDER BY ipaddr limit 1) FROM hosts h WHERE h.hid NOT IN (SELECT hid FROM crawled UNION SELECT hid FROM crawling));

8. Modified crawl_buddy.pl with database, user and password in lines 45, 47 and 49

9. Use crawl_ctl.pl to start each Web crawler with a unique nonnegative ID
   $ ./crawl_ctl.pl start 0
   $ ./crawl_ctl.pl start 1
   :
   Run crawl_ctl.pl without arguments to see its options:

```
$ ./crawl_ctl.pl
Usage: crawlerctl.pl {start—pause—resume—status—end} crawler-id
```

There is no bundled information for this portion, which is optional:

1. Configure a PHP4-enabled Web server to run WebBase/web-scripts/crawl*.php. crawl{stat,cmd,log,lib}.php can be relocated, but must remain in the same path as each other.

2. Edit the default directory specified in crawlstat.php:14 to point to the path WebBase/crawler/crawl_binaries/ in step 3.

3. Point a Web browser to the Web server of step 7, so that the server executes WebBase/web-scripts/crawlstat.php. (For example, if crawl*.php were copied into the Web server's document root, get "http://crawlserver/crawlstat".) This Web resource provides basic Web crawler monitoring.

3.1.2 More information

WebBase Web crawler
Junghoo Cho, Pranav Kantawala
This file last updated 9 Aug 2002, Wang Lam .
This file describes how the crawler is folded into WebBase.

Notes to users Documentation about the crawler and its operation is in crawl_binaries/README. "Instructions to compile and build the crawler binaries" should be skipped; use WebBase/Makefile instead to generate them (approximately) as instructed (cd WebBase/ && make crawler). As noted in crawl_binaries/README, some warning messages may show up, but can be ignored. The crawler was ported to Red Hat Linux 7.2 (Berkeley DB 3.2), but will now also compile on Red Hat Linux 7.0 (Berkeley DB 3.1). Red Hat Linux 6.1 does not have Berkeley DB 3, so builds on that version will fail. Please build the entire crawler using only one version of g++ (for example, by not interrupting the build or switching versions of Red Hat Linux halfway). Unlike the rest of WebBase, mixing incompatible object-file formats here may cause link failures for code in this directory.

Please do not create blank lines in crawler/crawl_binaries/crawl_config.

Both crawler/*_server/parameter.cc seem to think of them as EOF markers.

This issue may now be fixed.

Directories The crawler is imported into the WebBase source tree as follows (relative to WebBase root):

crawler/crawl_server/
: source code for the crawl server, which fetches Web pages from the World Wide Web

crawler/crawl_binaries/
: configuration and input for a crawl, including seed Web sites to crawl; this directory doubles as an example run environment for the crawler

Notes to maintainers

site_server/Makefile chooses Berkeley DB 3.1 or Berkeley DB 3.2 depending on the version of Red Hat Linux (7.0 or 7.2). This run-time choice allows the code to link correctly on one more version of Red Hat Linux, but the test (check /etc/redhat-release for a particular version) is weak. Versions other than the two tested above should default to Berkeley DB 3.2.

WebBase/Makefile does something a bit brain-dead: It will invoke {crawl_server,site_server}/Mak (which builds in the source directory) and then copy the needed binaries into crawl_binaries/. This is unlike the rest of WebBase, which is built into a WebBase/obj-(version)/ and the WebBase/bin/ directory.

Some source files in WebBase duplicate those in crawler/, and are compiled and linked independently. Please check to see whether changes to one source file need to be mirrored into the other copy.

Dependency information has been expunged from Makefiles because makedepend writes version-specific dependencies; the make depend targets have not been replaced. While editing the crawler code's dependencies, use make depend, but remove dependency information before checking in any revised Makefiles.

Do not drop project/stl (to use g++'s STL); this may expose some strange behavior. (For example, SiteServer::process_command tries to clone string msg into char* str, but g++'s string->string copy constructor does not necessarily allocate a new buffer, so modifications to str also clobber msg.)

3.1.3 ToDo list

Leyenda:

* active item

x no longer applies; "done" because siteserver/dnsdbserver replaced

- done

TODO list for crawler:

x siteserver is peppered with exit(1) statements (effectively, abort) in response to network failures. If dnsdbserver or a crawl_server is not listen(2)ing when siteserver tries to talk to it, siteserver will die.

x dnsdbserver may have a memory leak, causing it to segfault after some time.

* crawl_server does not suggest new sites to siteserver to crawl; building a new sitenames (domain-names-to-crawl) list requires scanning the old repository.

x siteserver does not save its state; if siteserver restarts, it goes through the sitenames list anew. siteserver also does not edit (append to) the sitenames list.

* siteserver does not implement a deny list (domain names never to crawl). To omit domain names, sitenames must be edited.

* crawl_server does not implement (the rarely-implemented) META-tag NOINDEX,NOFOLLOW robots exclusion in HTML files it sees. (crawl_server does implement robots.txt robot exclusion, as user-agent "Pita". This is slightly different from how it identifies itself currently, "Pita (email-address)" where an email-address is hard-coded at compile-time.)

* crawl_server does not recognize comments, and continues to parse for links inside them.

* crawl_server does not check whether URLs it sees are valid URIs (RFC 2396). In particular, it may allow \r or \n into URLs written into the repository, which may freak out programs expecting URLs to be one line.

* crawl_config cannot change substantially after siteserver is loaded. siteserver cannot reread crawl_config on demand. siteserver's data structures are fixed (in size) when siteserver starts up.

* crawl_server has a compiled MAX_NUM_QUEUES that limits the number of Web sites crawl_server can crawl concurrently. This constant cannot be exceeded, even if any configuration files request otherwise.

- crawl_server PAUSE may be "leaky:" If a CRAWL (new site) command arrives after a PAUSE, crawl_server will crawl anyway. A subsequent PAUSE command will pause any new sites without losing any old ones, but in the meanwhile, crawl_server can be induced into accumulating more sites to crawl concurrently than crawl_server/manager.h:MAX_NUM_QUEUES or crawl_config:[starter]:Threads allow. In this case, Threads would be ignored but crawl_server RESUME will drop any sites over MAX_NUM_QUEUES.

* crawl_server can have several pages queued for a Web site, simultaneously. The first is /robots.txt, and the second is /, but new pages may then be added to the queue before robots.txt is returned and processed.

* crawl_server tries only once to send its Web-site-completed message to its siteserver (or crawl_buddy.pl). If the send fails, e.g. because the Linux TCP/IP stack drops packets over localhost (!), the message is permanently lost.

3.2 Client

3.2.1 Getting Web Pages Out of WebBase

Status of This Document This Web page describes how to retrieve Web pages, individually or in bulk, from the StanfordWebBase, a World Wide Web repository built as part of the Stanford Digital Libraries Project by the Stanford University Database Group (InfoLab).
This document assumes familiarity with the Unix command line, including CVS. Otherwise, this document is self-contained.
This document is expected to be accurate for WebBase only up to 3 February 2001 except for RunHandlers, which is valid thereafter.
Because of widespread changes to WebBase code in progress, *instructions below only apply to CVS checkouts of WebBase before 12 Feb 2001, and its accompanying Web crawl on the machine Chub, except for RunHandlers which runs against the 2001 & later crawls.* Until the changes are finished, users with access to chub and who do not have an early checkout of WebBase may use the w̃lam/WebBase checkout (as shown in the examples below). If

you use w̃lam/WebBase, skip all software-build steps (instructions concerning make), because they have already been done. Instructions for the 2003 crawl will be added as soon as they are settled, but will probably be the same as the 2001 crawl. The 2002 distributions are hard coded to go to the 2001 crawl (eh on port 7003). We also have a crawl of .gov sites available on pita on port 7020(description of crawl). You will have to change your distribrequestor or getpages to access it. The next tarball will allow parameterized invocations of these perl scripts.

host	port	size	date	comments
Eh	7003	120Mpgs	1/2001	general crawl
Pita	7020	17Mpgs	12/2002	.gov crawl

Contents There are several ways to get Web pages out of WebBase, instructions for each of which is provided below.
The instructions assume Internet access to the machine hosting WebBase data and a CVS checkout of the WebBase code or an ftp get. Methods labelled "requires WebBase machine account" will not work without such machine access. Methods not so labelled will work for those without such access, assuming assistance from someone who can perform the steps requiring access to the WebBase machine.

- RunHandlers (recommended)
- simpleserver.pl
- webcat (requires WebBase machine account)
- webcast (requires WebBase machine account)
- server.pl
- server.C multicast-over-TCP (deprecated)
- SDLIP service is not yet available.
- IP multicast service is not yet available.

RunHandlers RunHandlers is supported on GNU/Linux and Solaris systems with GNU make (gmake), gcc (versions 2.9x), Perl 5.05+, and W3C's libwww.

1. Fetch the latest WebBase client source code from ftp://db.stanford.edu/pub/digital_library.

2. Unroll the source code. For example, GNU tar can do this with
   `> tar xfz webbase-client-????-??-??.tar.gz`

3. Follow the instructions in the source code's README.client.
   `> chdir dli2/src/WebBase/ && more README.client`

Below is an example (but obsolete) README.client. The instructions in the latest source code replace the instructions below, but if you don't already have the source, below is approximately what you would find.Build everything:
(Use a Linux box)

Quickstart:
Build everything: (Use a Linux or Solaris box)

Make sure the library path includes W3C's libwww, available at http://www.w3.org/Library/
In particular, this code needs files libwwwcore.* and libwwwutils.*.

Make sure environment variable WEBBASE points to WebBase: setenv WEBBASE xxx/WebBase

1. Run GNU make:
   WebBase/> ./configure
   WebBase/> make client

2. Test your build:

   (a) Turn on cat-handler, which simply outputs what it receives.
   In inputs/webbase.conf, set CAT_ON = 1

   (b) Try RunHandlers on a local example file:
   bin/RunHandlers inputs/webbase.conf \ "file://handlers/example-50-pages"
   [50 sample pages are printed]

3. Now try the network version:

Method 1 Run scripts/distribrequestor.pl to start a distributor on our server:
(either chmod +x scripts/distribrequestor.pl or invoke it with "perl")
WebBase/> scripts/distribrequestor.pl 0 [starts at offset 0]
client got back: 171.64.75.93:9007
WebBase/>
Save the port # and ip address: you'll need it. Now you can invoke RunHandlers with the above info:
WebBase/> bin/RunHandlers inputs/webbase.conf \ "net://171.64.75.93:9007/?numP

will print back 100 sample pages. All instances of RunHandlers connected to the above port will share the same pool of pages. To get an independent stream, rerun distribrequestor.pl to get a new port.

Method 2 You can also use our one-step script getPages.pl with no args (CHANGES ARE NEEDED INTO THE SCRIPT!!!!!), it will prompt you for the offset and num pages and invoke RunHandlers for you. As a rule of thumb, run at most 5 RunHandlers processes to conserve server/network resources.
Death threat:
If a distributor is inactive for 24hrs, it may be killed. To restart at the same point you must start a new distributor @ where it left off (+ 1 to prevent getting the previous page again)

To create a new handler:
You can use the other handlers in the distribution as templates. To add a new handler, add the following to the appropriate places:

1. #include "myhandler.h" into handlers/all_handlers.h

2. handler.push_back(new MyHandler()); into handlers/all_handlers.h (following the template of the handlers already there)

3. in Makefile, add entries for your segments to compile in the line: HANDLER_OBJS = jhandler.o [...]

4. (optional)in Makefile, customize your build if necessary by adding a line
jhandler_CXXFLAGS = -Iyour-include-dir –your-switches [...] (following the template of the handlers already there) We also have a one-button script called scripts/addHandler.pl that will prompt you for all your pieces and put them in place, without you having to do the above file surgery yourself.

To query local repository:

1. Create a new file with a list of all the files (complete path) which are part of the global repository

2. Change distribdaemon.pl:34 setting this file as input for repository

3. Change distribrequestor.pl:20 setting the host with the repository

4.

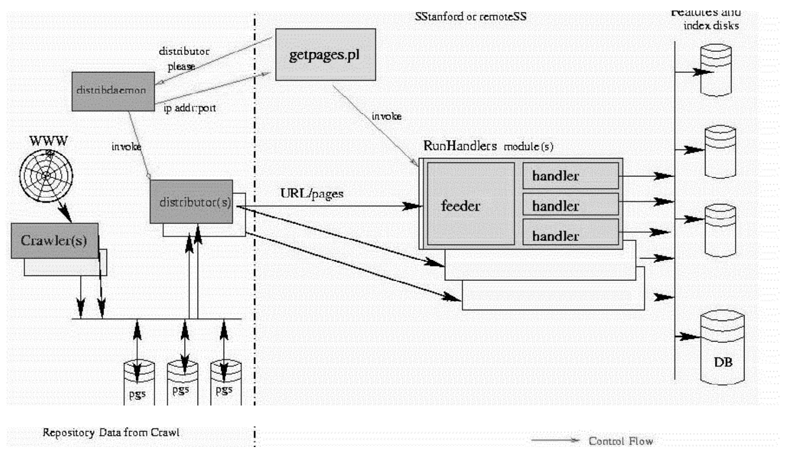

Figure 3: Runhanlers

4 Postgresql database

To create the database we must use
initdb -D /usr/local/pgsql/data -i
as a postgres user.

Use the schema described in WebBase/crawler/crawl_binaries/crawl_table_setup.sql. (WebBase/crawler/seed_list is an example of how a fellow WebBase developer loaded data into a database.)

Hosts

A list of the sites you want to crawl

- **hid:** a unique integer for the site; lower-numbers are crawled first
- **hostname:** hostname of the site
- **priority:** currently unused; any number will do
- **maxpages:** maximum number of pages to retrieve from the site
- **maxdepth:** maximum number of links to follow from root page(s)

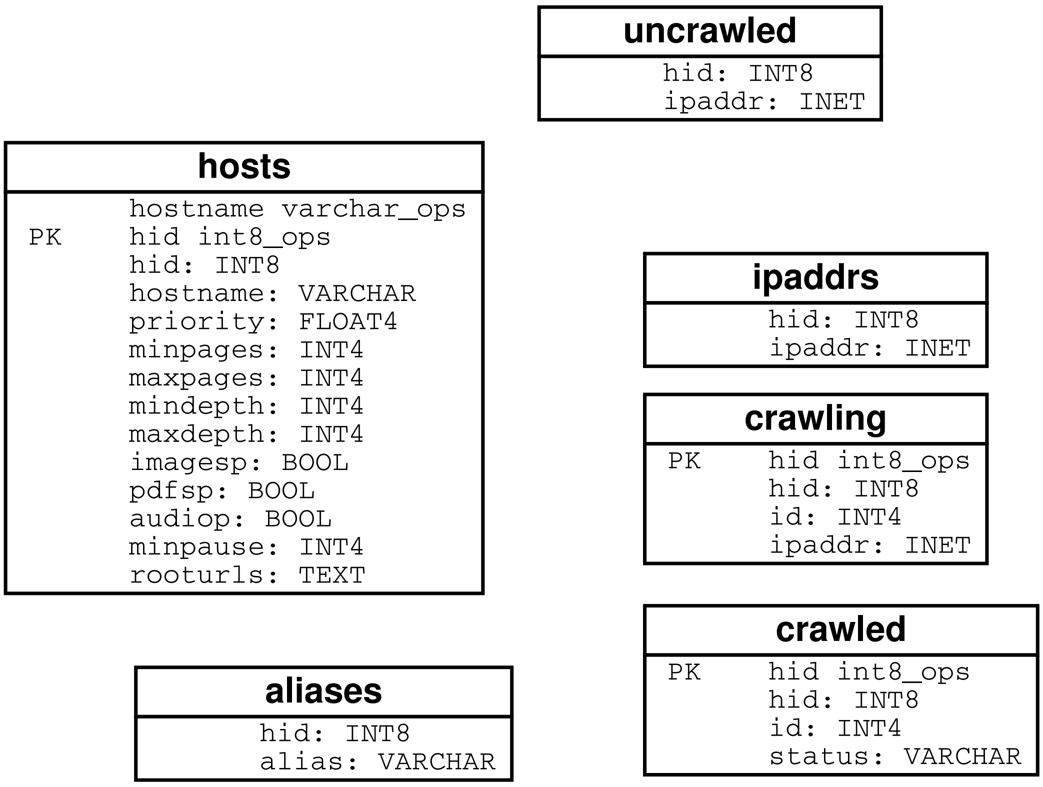

Figure 4: Webbase database diagram

- **{imagesp, pdfsp, audiop}:** currently unused
- **minpause:** milliseconds between requests to the site (e.g., 10000 for a ten-second pause between fetches); please keep this large to avoid beating up the Web sites you crawl
- **rooturls:** space-separated list of URLs where you want to start crawl for the site: Leave null to start at the root page. http://Hosts.hostname/ by default

IpAddres

IP addresses for the Web sites

- **hid:** corresponds to a Hosts.hid
- **ipaddr:** an IP address of the Web site Hosts.hostname

Aliases

Other hostnames for a Web site

- **hid:** corresponds to a Hosts.hid
- **alias:** a hostname that is the same site as Hosts.hostname

Each hid must join exactly one row in Hosts, at least one row in IPAddrs(so the crawler knows where to connect!), and any number of rows in Aliases (possibly zero)

Then, generate the tables uncrawled, crawling, and crawled as in WebBase/crawler/crawl_binaries/crawl_table_setup.sql (to populate from table hosts)

To start the database we must use
postmaster -D /usr/local/pgsql/data -i

5 Glossary

RunHandlers (formerly "process") an executable that indexes a stream, file or repository. Made up basically of a feeder and one or more handlers.

handler the interface that any index-building piece of code must implement. The interface's main (only) method will provide a page and associated metadata and the implementor of the method can do whatever he wants with it.

feeder the interface for receiving a page stream from any kind of source (directly from the repository, via Webcat, via network, etc.). The key method of the interface is "next" which advances the stream by one page. After calling next, various other methods can be used to get the associated metadata for the current page in the stream. Can also be used to build indexes if the index-building code is written to process page streams

distributor a program that disseminates pages to multiple clients over the network, supporting session ID's, etc. A generalization of what Distributor.cc in text-index/ does.

A DDL for database creation

– – Create tables for crawler operation
– Wang Lam ¡wlam@cs.stanford.edu¿
– created Aug 2001

```
– also see crawler/seed_list/load-aliases.pl
–
create table hosts(
    hid BIGINT PRIMARY KEY,
    hostname character varying(1024) unique not null,
    priority real not null,
    minpages integer,
    maxpages integer not null,
    mindepth integer,
    maxdepth integer not null,
    imagesp boolean,
    pdfsp boolean,
    audiop boolean,
    minpause integer,
    rooturls text);
– all tables should have hid -¿ hosts.hid referential integrity, and
– on delete cascade.
– host.hid should not be deletable if host is being crawled.

create table aliases (
    hid bigint not null,
    alias character varying (1024) not null);

create index aliases_hid_skey on aliases (hid);
create table ipaddrs(
    hid bigint not null,
    ipaddr inet not null);
create index ipaddrs_hid_skey on ipaddrs (hid);
– Deprecated (crawling is in hid order, not priority order):
– create index hosts_priority_skey on hosts(priority);
# to manually create and fill tables:
create table uncrawled (
    hid BIGINT PRIMARY KEY,
    id integer NOT NULL);
create table crawling (
    hid BIGINT PRIMARY KEY,
    id integer NOT NULL, ipaddr inet NOT NULL);
create index crawling_ipaddr_skey on crawling(ipaddr);
create table crawled (
    hid bigint primary key,
    id integer,
    status varchar(1024));
– to populate from table hosts:
– create table uncrawled (
–      hid, ipaddr) as
–      (select h.hid,
–      (select ipaddr from ipaddrs
–      where hid = h.hid order by ipaddr limit 1)
–      from hosts h);
– create index uncrawled_hid_skey on uncrawled(hid);
– create table crawling (hid BIGINT PRIMARY KEY, id integer NOT
NULL, ipaddr inet NOT NULL);
– create index crawling_ipaddr_skey on crawling(ipaddr);
– create table crawled (hid bigint primary key, id integer, status varchar(1024));
– PostgreSQL only:
vacuum analyze
```

B crawl_config file example

```
### Configuration parameters for siteserver module

[siteserver]
#hostname:port of siteserver
LocalAddress=pc150:20003,localhost:20004,localhost:20005
#log file that stores log of siteserver
LogFile=site.log
#file that stores names of sites to be crawled
SiteFile=sitenames
#Maximum number of pages to be downloaded from a site
MaxPage=3000
#Maximum link-depth at which to crawl in a site
MaxLinkDepth=10
#Interval in milli-seconds between crawler's consecutive requests to a site
RequestInterval=10000
### Configuration parameters for crawlserver module

[crawlserver]
#hostname:port of crawlers. There can be multiple crawlers.
#All entries should be separated by a comma
LocalAddress=pc150:20103,localhost:20104,localhost:20105
#file that stores location of repository
```

```
#dir.config.N refers to Nth crawler
#there should be N such files if N crawlers are running
#numbering starts at 0 and ends at N-1
ConfigFile=dir.config
# Extensions which are followed
ExtParse=htm,html,shtm,shtml,php,phtml,asp,no_ext
# Extensions which are stored (add mp3,gif,jpg,png ?)
ExtStore=htm,html,shtm,shtml,php,phtml,asp,txt,text,no_ext
# Set this to 1 to enable fetching the targets of tags
FollowIMG=0
### Configuration parameters for dnsdbserver module

[dnsdbserver]
#hostname:port of dnsdbserver
DnsAddress=pita:20100
#log file that stores log of dnsdbserver
LogFile=dnsdb.log
#name of the Database that stores mappings
#e.g. names will be dnsdb.name and dnsdb.ip
DbName=dnsdb
### Configuration parameters for starter module

[starter]
#Number of crawlers to be run
Crawlers=3
#Number of requests to send to each site during crawler bootstrap process
Threads=200
```