An Introduction to Heritrix

An open source archival quality web crawler

July 14, 2004

Internet Archive Web Team
Gordon Mohr
Michael Stack
Igor Ranitovic
Dan Avery
Michele Kimpton

Abstract

Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality webcrawler project. The Internet Archive started Heritrix development in the early part of 2003. The intention was to develop a crawler for the specific purpose of archiving websites and to support multiple different use cases including focused and broadcrawling. The software is open source to encourage collaboration and joint development across institutions with similar needs. A pluggable, extensible architecture facilitates customization and outside contribution. Now, after over a year of development, the Internet Archive and other institutions are using Heritrix to perform focused and increasingly broad crawls.

Introduction

The Internet Archive (IA) is a 5013C non-profit corporation, whose mission is to build a public Internet digital library. Over the last 6 years, IA has built the largest public web archive to date, hosting over 400 TB of data.

The Web Archive is comprised primarily of pages collected by Alexa Internet starting in 1996. Alexa Internet is a Web cataloguing company founded by Brewster Kahle and Bruce Gilliat in 1996. Alexa Internet takes a snapshot of the web every 2 months, currently collecting 10 TB of data per month from over 35 million sites. Alexa Internet donates this crawl to the Internet Archive, and IA stores and indexes the collection. Alexa uses its own proprietary software and techniques to crawl the web. This software is not available to Internet Archive or other institutions for use or extension.

In the latter part of 2002, the Internet Archive wanted the ability to do crawling internally for its own purposes and to be able to partner with other institutions to crawl and archive the web in new ways. The Internet Archive concluded it needed a large-scale, thorough, easily customizable crawler. After doing an evaluation of other open source software available at the time it was concluded no appropriate software existed that had the flexibility required yet could scale to perform broad crawls.

The Internet Archive believed it was essential the software be open source to promote collaboration between institutions interested in archiving the web. Developing open source software would encourage participating institutions to share crawling experiences, solutions to common problems, and even the development of new features.

The Internet Archive began work on this new open source crawler development project in the first half of 2003. It named the crawler *Heritrix*. This paper gives a high-level

overview of the Heritrix crawler, *circa* version 1.0.0 (August, 2004). It outlines the original use-cases, , the general architecture, current capabilities and current limitations. It also describes how the crawler is currently used at the Internet Archive and future plans for development both internally and by partner institutions. It also describes how the Archive currently uses the crawler and future plans for development both internally and by partner institutions.

Use Cases

The Internet Archive and its collaborators wanted a crawler capable of each of the following crawl types:

Broad crawling: Broad crawls are large, high-bandwidth crawls in which the number of sites and the number of valuable individual pages collected is as important as the completeness with which any one site is covered. At the extreme, a broad crawl tries to sample as much of the web as possible given the time, bandwidth, and storage resources available.

Focused crawling: Focused crawls are small- to medium-sized crawls (usually less than 10 million unique documents) in which the quality criterion is complete coverage of some selected sites or topics.

Continuous crawling: Traditionally, crawlers pursue a snapshot of resources of interest, downloading each unique URI one time only. Continuous crawling, by contrast, revisits previously fetched pages – looking for changes – as well as discovering and fetching new pages, even adapting its rate of visitation based on operator parameters and estimated change frequencies.

Experimental crawling: The Internet Archive and other groups want to experiment with crawling techniques in areas such as choice of what to crawl, order in which resources are crawled, crawling using diverse protocols, and analysis and archiving of crawl results.

Required Capabilities

Based on the above use cases, the Internet Archive compiled a list of the capabilities required of a crawling program. An important contribution in developing the archival crawling requirements came from the efforts of the International Internet Preservation Consortium (IIPC) a consortium of twelve National Libraries and the Internet Archive. The mission of the IIPC is to acquire, preserve and make accessible knowledge and information from the Internet for future generations everywhere, promoting global exchange and international relations. IIPC member libraries have diverse web resource collection needs, and contributed several detailed requirements that helped define the goals for a common crawler. The detailed requirements document developed by the IIPC can be found at:
"http://netpreserve.org/publications/iipc001.pdf"http://netpreserve.org/publications/iipc-d-001.pdf.

The Heritrix Project

Heritrix is an archaic word for *heiress*, a woman who inherits. Since our crawler seeks to collect and preserve the digital artifacts of our culture for the benefit of future researchers and generations, this name seemed appropriate.

Java was chosen as the implementation software language. As a high-level, object-oriented language, Java offers strong support for modular design and components that are both incrementally extendable and individually replaceable. Other key reasons for choosing Java were its rich set of quality open source libraries and its large developer community, making it more likely that we would benefit from previous work and outside contributions.

The project homepage is <http://crawler.archive.org>, featuring the most current information on the project, downloads of the crawling software, and project documents. There are also public databases of outstanding feature requests and bugs. The project also provides an open mailing list to promote exchange of information between Heritrix developers and other interested users.

Sourceforge [SOURCEFORGE], a site offering free online services for over 84,000 open source software efforts, hosts the Heritrix project. Sourceforge provides many of the online collaborative tools required to manage distributed-team software projects such as:

Versioned source code repository (CVS)
Issue databases for tracking bugs and enhancement requests
Web hosting
Mirrored, archived, high-availability file release system
Mailing lists

A large community of developers knows and uses Sourceforge, which can help to create awareness of and interest in the software. The limitations of Sourceforge include:

Occasionally unavailable or slow
No direct control over problems that do arise
Issue-tracking features and reporting are crude
Can not be used to manage other internal software projects which may not be open source

Heritrix is licensed under the Gnu Lesser General Public License (LGPL) [LGPL]. The LGPL is similar to the Gnu General Public License (GPL) in that it offers free access to the full source code of a program for reuse, extension, and the creation of derivative works, under the condition that changes to the code are also made freely available. However, it differs from the full GPL in allowing use as a module or library inside other proprietary, hidden source applications, as long as changes to the library are shared.

Milestones

Since project inception, major project milestones have included:

-Investigational crawler prototype created, and various threading and network access

strategies tested, 2nd Q 2003

-Core crawler without user-interface created, to verify architecture and test coverage compared to HTTrack [HTTRACK] and Mercator [MERCATOR] crawlers, 3rd Q 2003

-Nordic Web Archive [NWA] programmers join project in San Francisco, 4th Q 2003 – 1st Q 2004, adding a web user-interface, a rich configuration framework, documentation, and other key improvements

-First public release (version 0.2.0) on Sourceforge, January 2004

-First contributions by unaffiliated developer, January 2004

-Workshops with National Library users in February and June of 2004

-Used as the IA's crawler for all contracted crawls – usually consisting of a few dozen to a few hundred sites of news, cultural, or public-policy interest – since beginning of 2004.

-Adopted as the official crawler for the NWA, June 2004

-Version 1.0.0 official release, for focused and experimental crawling, in August 2004

Heritrix Crawler Architecture

In this section we give an overview of the Heritrix architecture, describing the general operation and key components.

At its core, the Heritrix crawler was designed as a generic crawling framework into which various interchangeable components can be plugged. Varying these components enables diverse collection and archival strategies, and supports the incremental evolution of the crawler from limited features and small crawls to our ultimate goal of giant full-featured crawls.

Crawl setup involves choosing and configuring a set of specific components to run. Executing a crawl repeats the following recursive process, common to all web crawlers, with the specific components chosen:

1. Choose a URI from among all those scheduled
2. Fetch that URI
3. Analyze or archive the results
4. Select discovered URIs of interest, and add to those scheduled
5. Note that the URI is done and repeat

The three most prominent components of Heritrix are the *Scope*, the *Frontier*, and the *Processor Chains*, which together serve to define a crawl.

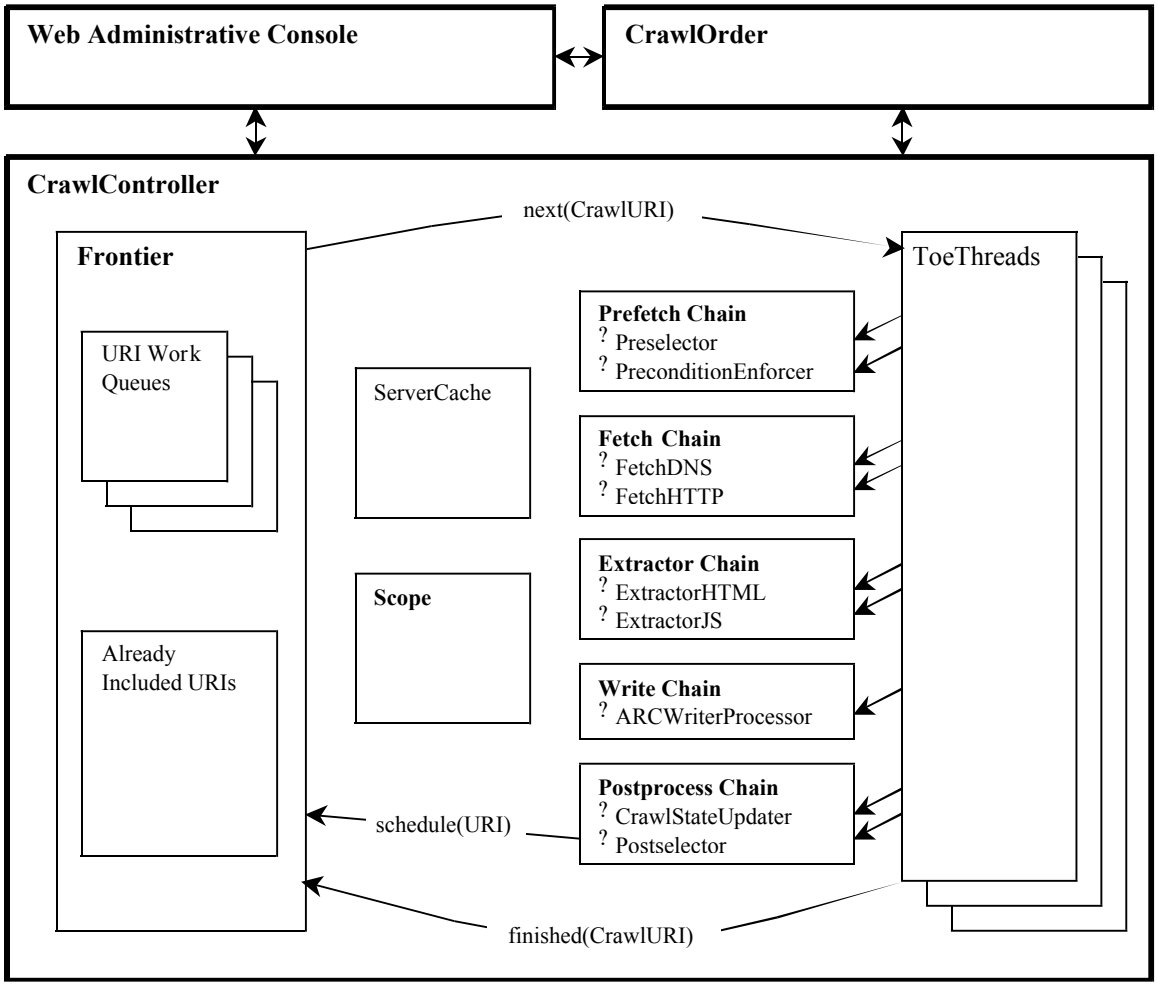
The Scope determines what URIs are ruled into or out of a certain crawl. The Scope includes the "seed" URIs used to start a crawl, plus the rules used in step 4 above to determine which discovered URIs are also to be scheduled for download.

The Frontier tracks which URIs are scheduled to be collected, and those that have already been collected. It is responsible for selecting the next URI to be tried (in step 1 above), and prevents the redundant rescheduling of already-scheduled URIs (in step 4 above).

The Processor Chains include modular *Processors* that perform specific, ordered actions on each URI in turn. These include fetching the URI (as in step 2 above), analyzing the returned results (as in step 3 above), and passing discovered URIs back to the Frontier (as in step 4 above).

Figure 1 shows these major components of the crawler, as well as other supporting components, with major relationships highlighted.

Figure 1: Major Components of Heritrix in a Representative Configuration



CAPTION: *The* Web Administrative Console *composes a* CrawlOrder*, which is then used to create a working assemblage of components within the* CrawlController*. Within the* CrawlController*, arrows indicate the progress of a single scheduled* CrawlURI *within one ToeThread's execution loop. A* CrawlURI *is provided by the* Frontier *via the* next() *operation, then presented to each Processor in turn, and finally returned to the Frontier via a* finished() *operation. At the* Postselector *step, any newly discovered* URI*s of interest are inserted into the* Frontier *via a* schedule*() operation.*

Key Components

In this section we go into more detail on each of the components featured in Figure 1.

The *Web Administrative Console* is in many ways a standalone web application, hosted by the embedded Jetty Java HTTP server. Its web pages allow the operator to choose a crawl's components and parameters by composing a *CrawlOrder*, a configuration object that also has an external XML representation.

A crawl is initiated by passing this CrawlOrder to the *CrawlController*, a component which instantiates and holds references to all configured crawl components. The CrawlController is the crawl's global context: all subcomponents can reach each other through it. The Web Administrative Console controls the crawl through the CrawlController.

The CrawlOrder contains sufficient information to create the Scope. The Scope seeds the Frontier with initial URIs and is consulted to decide which later-discovered URIs should also be scheduled.

The Frontier has responsibility for ordering the URIs to be visited, ensuring URIs are not revisited unnecessarily, and moderating the crawler's visits to any one remote site. It achieves these goals by maintaining a series of internal queues of URIs to be visited, and a list of all URIs already visited or queued. URIs are only released from queues for fetching in a manner compatible with the configured politeness policy. The default provided Frontier implementation offers a primarily breadth-first, order-of-discovery policy for choosing URIs to process, with an option to prefer finishing sites in progress to beginning new sites. Other Frontier implementations are possible.

The Heritrix crawler is multithreaded in order to make progress on many URIs in parallel during network and local disk I/O lags. Each worker thread is called a *ToeThread*, and while a crawl is active, each ToeThread loops through steps that roughly correspond to the generic process outlined previously:

Ask the Frontier for a *next()* URI
Pass the URI to each Processor in turn. (Distinct Processors perform the fetching, analysis, and selection steps.)
Report the completion of the *finished()* URI

The number of ToeThreads in a running crawler is adjustable to achieve maximum throughput given local resources. The number of ToeThreads usually ranges in the hundreds.

Each URI is represented by a *CrawlURI* instance, which packages the URI with additional information collected during the crawling process, including arbitrary nested named attributes. The loosely-coupled system components communicate their progress and output through the CrawlURI, which carries the results of earlier processing to later processors and finally, back to the Frontier to influence future retries or scheduling.

The *ServerCache* holds persistent data about servers that can be shared across CrawlURIs and time. It contains any number of *CrawlServer* entities, collecting information such as IP addresses, robots exclusion policies, historical responsiveness, and per-host crawl statistics.

The overall functionality of a crawler with respect to a scheduled URI is largely specified by the series of Processors configured to run. Each Processor in turn performs its tasks, marks up the CrawlURI state, and returns. The tasks performed will often vary conditionally based on URI type, history, or retrieved content. Certain CrawlURI state also affects whether and which further processing occurs. (For example, earlier Processors may cause later processing to be skipped.)

Processors are collected into five chains:

Processors in the *Prefetch Chain* receive the CrawlURI before any network activity to resolve or fetch the URI. Such Processors typically delay, reorder, or veto the subsequent processing of a CrawlURI, for example to ensure that robots exclusion policy rules are fetched and considered before a URI is processed.
Processors in the *Fetch Chain* attempt network activity to acquire the resource referred-to by a CrawlURI. In the typical case of an HTTP transaction, a Fetcher Processor will fill the "request" and "response" buffers of the CrawlURI, or indicate whatever error condition prevented those buffers from being filled.
Processors in the *Extract Chain* perform follow-up processing on a CrawlURI for which a fetch has already completed, extracting features of interest. Most commonly, these are new URIs that may also be eligible for visitation. URIs are only discovered at this step, not evaluated.
Processors in the *Write Chain* store the crawl results – returned content or extracted features – to permanent storage. Our standard crawler merely writes data to the Internet Archive's ARC file format but third parties have created Processors to write other data formats or index the crawled data.
Finally, Processors in the *Postprocess Chain* perform final crawl-maintenance actions on the CrawlURI, such as testing discovered URIs against the Scope, scheduling them into the Frontier if necessary, and updating internal crawler information caches.

Table 1: Processor modules included in Heritrix 1.0.0

	Name	Function
Prefetch	Preselector	Offers an opportunity to reject previously-scheduled URIs not of interest.
	PreconditionEnforcer	Ensures that any URIs which are preconditions for the current URI are scheduled beforehand.
Fetch	FetchDNS	Performs DNS lookups, for URIs of the "*dns:*" scheme.

	FetchHTTP	Performs HTTP retrievals, for URIs of the "*http:*" and "*https:*" schemes.
Extract	ExtractorHTML	Discovers URIs inside HTML resources.
	ExtractorJS	Discovers likely URIs inside Javascript resources.
	ExtractorCSS	Discovers URIs inside Cascading Style Sheet resources.
	ExtractorSWF	Discovers URIs inside Shockwave/Flash resources.
	ExtractorPDF	Discovers URIs inside Adobe Portable Document Format resources.
	ExtractorDOC	Discovers URIs inside Microsoft Word document resources.
	ExtractorUniversal	Discovers legal URI patterns inside any resource with an ASCII-like encoding.
Write	ARCWriterProcessor	Writes retrieved resources to a series of files in the Internet Archive's ARC file format.
Postprocess	Postselector	Evaluates URIs discovered by previous processors against the configured crawl Scope, scheduling those of interest to the Frontier.
	CrawlStateUpdater	Updates crawler-internal caches with new information retrieved by earlier processors.

Features and Limitations

Features

Heritrix 1.0.0 offers the following key features:

- Collects content via HTTP recursively from multiple websites in a single crawl run, spanning hundreds to thousands of independent websites, and millions to tens of millions of distinct resources, over a week or more of non-stop collection.
- Collects by site domains, exact host, or configurable URI patterns, starting from an operator-provided "seed" set of URIs
- Executes a primarily breadth-first, order-of-discovery policy for choosing URIs to process, with an option to prefer finishing sites in progress to beginning new sites ("site-first" scheduling).
- Highly extensible with all of the major Heritrix components – the scheduling Frontier, the Scope, the protocol-based Fetch processors, filtering rules, format-based Extract processors, content Write processors, and more – replaceable by alternate implementations or extensions. Documented APIs and HOW-TOs explain extension options.

Highly configurable. Options include:

- Settable output locations for logs, archive files, reports and temporary files.
- Settable maximum bytes to download, maximum number of documents to download, and maximum time to spend crawling.
- Settable number of 'worker' crawling threads.
- Settable upper bound on bandwidth-usage.
- Politeness configuration that allows setting minimum/maximum time between requests as well an option to base the lag between requests on a multiple of time elapsed fulfilling the most recent request.
- Configurable inclusion/exclusion filtering mechanism. Includes regular expression, URI path depth, and link hop count filters that can be combined variously and attached at key points along the processing chain to enable fine tuned inclusion/exclusion.

Most options can be overridden on a per domain basis and then in turn further amended based off time-of-day, a regular expression or URI port match 'refinements'.

Robots.txt refresh rate as well as rich URI-retry-on-failure configurations.

Web-based user interface (UI) – "control panel" – via which crawls can be configured, started, paused, stopped, and adjusted mid-crawl. The UI can be used to view the current state of the crawl, logs and generated reports.
Logs retrievals – the URI of the document downloaded, the download result code, time to download, document size, and the link that pointed to the downloaded document – as well as processing errors encountered fetching, scheduling and analyzing documents.

Reports capture the number of URIs visited, the number of URIs discovered and pending, a summary of errors encountered, and memory/bandwidth/storage used – both in total and on a sampling interval. Other runtime reports detail state of each crawl worker thread and the state of the Frontier's URI queues.

Includes link extractors for the common web document types -- HTML, CSS, JavaScript, PDF, MS WORD, and FLASH – as well as a catchall "Universal Extractor" for extracting links from everything else.
Archives the actual, binary DNS, HTTP, and HTTPS server responses, using an open storage format ("ARC").
Negotiation of "http://www.faqs.org/rfcs/rfc2617.html" RFC 2617 (BASIC and DIGEST Authentication) and HTML form-based login authentication systems.
Written in portable Java, so works under Linux, Windows, Macintosh OS X, and other systems.
All source code available under an open source license (Library/Lesser Gnu Public License, LGPL).

Limitations

Heritrix has been used primarily for doing focused crawls to date. The broad and continuous use cases are to be tackled in the next phase of development (see below).
Key current limitations to keep in mind are:

- Single instance only: cannot coordinate crawling amongst multiple Heritrix

instances whether all instances are run on a single machine or spread across multiple machines.
- Requires sophisticated operator tuning to run large crawls within machine resource limits.
- Only officially supported and tested on Linux
- Each crawl run is independent, without support for scheduled revisits to areas of interest or incremental archival of changed material.
- Limited ability to recover from in-crawl hardware/system failure.
- Minimal time spent profiling and optimizing has Heritrix coming up short on performance requirements (See Crawler Performance below).

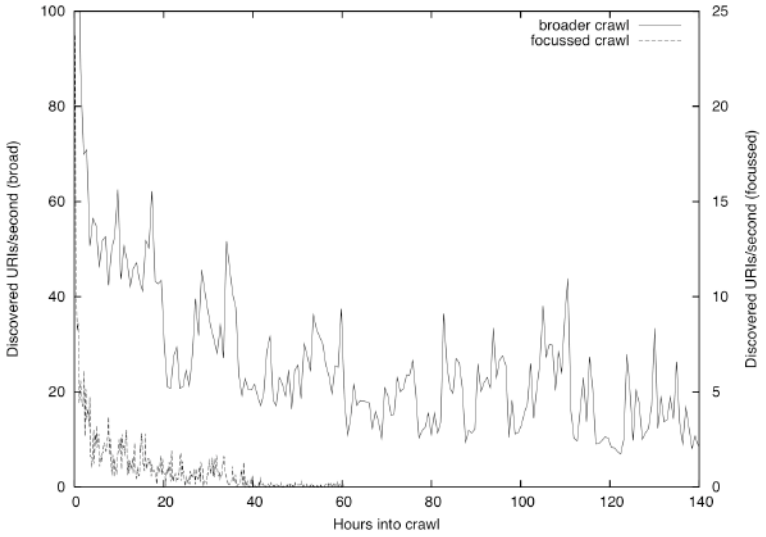
Crawler Performance

Currently the crawler is being used by Internet Archive to do several focused crawls for our partners. It is also being used by several of the Nordic country libraries for domain crawling. In each of these production crawls the Heritrix crawler is being tested in a "true web" environment. By performing weekly crawls, IA continuously tests the capabilities of the crawler and monitors performance and quality compared to other crawlers. A typical weekly crawl configuration and administration is illustrated in on the Heritrix crawler website at http://crawler.archive.org/cgi-bin/wiki.pl?HomePage.

To evaluate the operational performance of the crawler Internet Archive measures the crawl documents captured over time. Typical performance of the Heritrix crawler for crawls spanning over several days to several weeks are illustrated in the chart below.

Chart 1 demonstrates the performance of the crawler during a small (less than 20 seeds) focused crawl and a much broader crawl(several hundred seeds). For the small focused crawl the rate of documents discovered quickly drops off and slows the crawler as the discovery process is completed. For the larger crawl (line 2) the rate of documents discovered over time remains unchanged as the discovery process continues over time. This is the type of behavior one would expect for a broad crawl if there is always a queue of URIs in the frontier.

Chart 1 Documents discovered per second over time



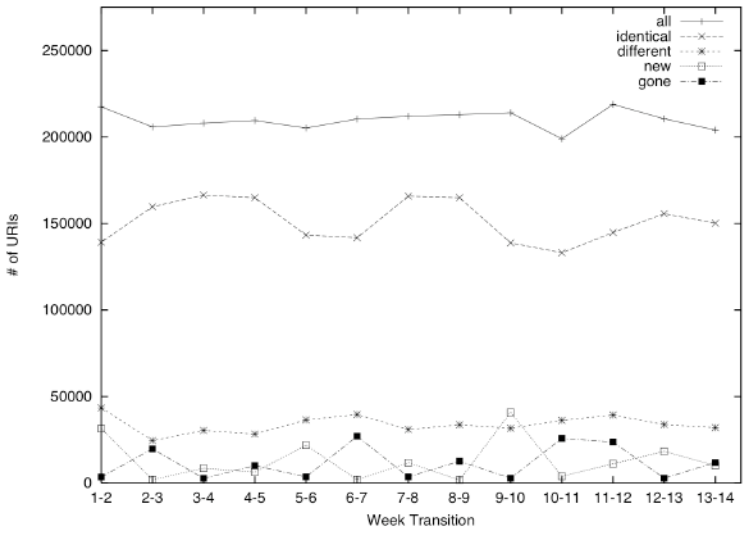
A set of tools was developed at the Internet Archive for internal use to determine the quality of a crawl compared to other independent crawlers or between periodic crawls. The URI comparison tool is a post-crawling analysis tool that compares two sets of URIs and produces statistical reports on similarity and the differences between two crawls. This tool can be used to compare two crawls having the same URI seeds or to compare coverage of different crawling software.

The URI comparison tool requires input of two crawl sets of URIs and their respective HTTP response codes, content lengths, and MIME types. When two different crawls,

When using the comparison tools, one can determine the percent overlap between two crawls and what percentage of content is unique to both crawls.

This tool can also be used to measure URI overlap between periodic crawls (ie weekly or daily) starting from the same seed list. The chart below shows for this particularly weekly crawl this is a 70% overlap of identical URI's. In this particular case where the overlap is so high one would want to employ deduplication techniques if plausible. This chart also shows the number of new URI's discovered each week and number of URIs which have disappeared from the prior week.

Chart 2 Weekly comparision Crawl of 15 seed domains



Many of these tests and quality assurance procedures are still in development as we continue to learn more about quality archival crawling.

Future Plans

Current plans for the future development of Heritrix, post 1.0.0, fall into three general categories: improving its scale of operation; adding advanced scheduling options; and incremental modular extension.

Our top priority at the Internet Archive is to dramatically increase the scale of crawls possible with Heritrix, into crawls that span hundreds of millions and then billions of web resources. This entails two necessary advances.

First, the ability to perform crawls of arbitrarily large size and duration on a single machine, with minimal operator tuning, limited only by available disk storage. This will require changes that strictly cap the RAM used by the crawler, and the conversion of every crawler data structure that grows with crawl size or duration to use disk-based data structures, trading away performance for long, large runs.

Second, a way to arrange for networks of cooperating crawl machines to intelligently divide up the work of a crawl. This will enable the rapid acquisition of ever-larger collections of web resources, limited only by the bandwidth and local hardware resources devoted to the crawl. Enabling effective distribution across machines requires strategies for dividing the work, sharing results, and recovering from partial failures. Other work in parallel crawling clusters, such as that done by the UbiCrawler project, provides useful precedent for the necessary additions.

Separately, there are a number of incremental feature improvements planned to expand the base crawler capabilities and the range of optional components available. The Internet Archive plans to implement:

- Support for FTP fetching
- Improved recovery of crawls from set checkpoints
- Automatic detection and adaptation to many "crawler traps" and challenging website design practices (such as URI-line session-IDs)
- Additional operator options for specifying crawl scopes, dealing with in-crawl problems, and forcing the retry of groups of URIs.

Collaboration with other Institutions

A top priority of several outside efforts building atop Heritrix is to add greater sophistication to the scheduling of URIs. This will likely include:

- Periodic revisits of sites or URIs at predetermined intervals
- Working to only fetch, or only store, changed resources
- Adaptively predicting rates of change and using such information to change revisit schedules
- Synamically prioritizing work based on diverse measures of site and URI value

Discussions are underway for such work to be coordinated with interested European National Libraries.

In the open source community

Others are using the crawler outside of the archiving and library community. While anyone can download Heritrix with no obligation to report back on its use or extension, questions and patches coming in to the general Heritrix discussion list will sometimes comment on the projects to which Heritrix is being applied. One group uses Heritrix to harvest web pages as a means of testing whether their commercial tokenizing tools continue to work against the latest web content. Another group is using Heritrix to take frequent snapshots of particular web servers for a University research project. It appears someone else is investigating Heritrix as a way to harvest institutional contact addresses, perhaps to make unsolicited commercial approaches. (We have no veto power over outside uses of our released software.)

A main focus for the future is continued fostering of the community that is growing up around Heritrix, to help improve the common code base with diverse crawling experience and enhancements.

Conclusion

The Internet Archive's digital collections include a large and unique historical record of

web content. This Web collection continues to grow each year, with regular Alexa snapshots and now, material collected by the new Heritrix web crawling software. Heritrix software is the product of an open source approach and international collaboration. In its 1.0.0 version, Heritrix is a full-featured focused crawler with a flexible architecture to enable many divergent crawling practices, now and in the future. Its evolution should continue to benefit from an open collaborative process, as it becomes capable of larger and more sophisticated crawling tasks.

Acknowledgements

This paper includes significant contributions from others at the Internet Archive, including Brewster Kahle, Raymie Stata and Brad Tofel,

References

[GUTENBERG] http://www.gutenberg.net/
[ICDL] http://www.icdlbooks.org/
[IIPC] http://www.netpreserve.org/
[ALEXA] http://pages.alexa.com/company/index.html
[BURNER97] http://www.webtechniques.com/archives/1997/05/burner/
[WAYBACK] http://www.archive.org/web/web.php
[NWA] http://nwa.nb.no/
[SOURCEFORGE] http://www.sourceforge.net
[LGPL] http://www.gnu.org/copyleft/lesser.html
[HTTRACK] http://www.httrack.com
[MERCATOR] http://research.compaq.com/SRC/mercator/
[JETTY] http://jetty.mortbay.com/
[ROBOTS] http://www.robotstxt.org
[UBICRAWLER] http://ubi.imc.pi.cnr.it/projects/ubicrawler/