THE LCC: A LOCAL CONTENT CACHE

The LCC is a Local Content Cache system facilitating the fetching and storage of network accessable content.

There are three parties involved. A “content provider” that provides the data (typically a web server), the LCC that stores the data, and a “content user” can then interrogate the LCC to retrieve cached content from the local system.

Typical users of this system can range from global internet search engines through to central indexers of corporate intranets or universities.

The traditional approach to this problem involves spidering with link extraction. This involves continual periodic polling of content to ensure the cache is kept up to date.

Advantages found by using the LCC include:

Scalability
: Webpages are only collected when necessary. This provides much more cost effective use of bandwidth and more efficient use of time, allowing more to be indexed with given resources.

Changes are noticed immediately
: Modified items are collected soon after notification implying an up to date index. Modifications are visible in the index typically within a day of changes instead of within weeks or months as with traditional spidering systems. This is a very useful function for news sites which can use automatic notification of change.

Website structure is more easily detected
: With traditional spidering and link extraction, a lot of work is necessary to determine the structure of the website; to find framed pages for example. With the LCC protocol it is possible to specify a “browse URL” different from the URL used to retrieve the content directly (the “fetch” URL). This mechanism provides flexibility for the use of different future data types as well, reflecting differing “views” of the same data.

Bandwidth and time of day control
: With the LCC protocol, it is possible to specify the maximum allowable bandwidth used for fetching content from a site. This may be high for a well connected site, or low for a site with less resources available or dynamically served pages. Additionally, it is possible to specify in detail at what time periods the robot can come calling.

Free open source software
: The software implementing the protocol and a high performance collection system is both open source and free, protected by the GNU GPL license.

WHERE CAN I GET IT?

The main project page can be found at:

    http://www.nlnet.nl/projects/lcc/

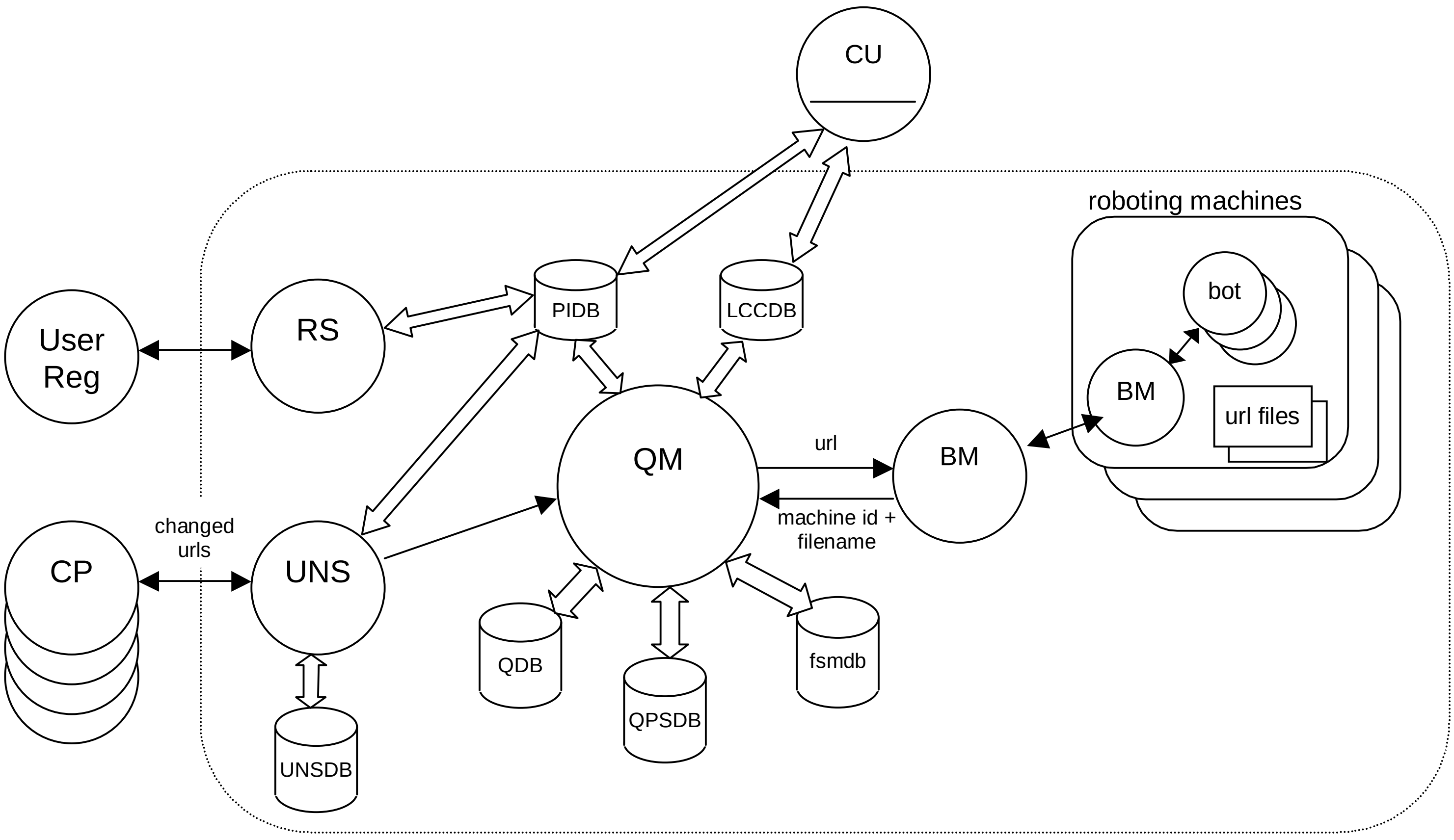
with LCC source available from sourceforge as the LoCoCa project:

```
          http://sourceforge.net/projects/lococa

It currently runs on Linux systems.
```

HOW DOES IT WORK?

In order to create a high performance system and aid scalability, the LCC has several modules that appear as separate processes, as shown below.



The main processes have the following functions:

CP A "Content Provider" process.
This is a process run by a registered provider on his own machine that submits changed-URL information to the LCC.

UNS The "Update Notification Server" process.
This is the "front end" to the LCC. It accepts an XML stream generated by a CP indicating a set of changed-URL information which is then buffered into local files.
It responds to the CP with an XML stream indicating errors (if any) of previous fetches for that provider, the current state of fetching for that provider and current quota information.
The buffered changed-URL information is forwarded to the QM if and when the QM is available.

QM The "Queue Manager" process.
This coordinates the fetching and storage of URL data. As such, it places URLs to be fetched into a MySQL table representing a queue and, after the URL has been fetched, updates the main URL repository table.

BM The "Bot Manager" process.
This distributes fetching work to one or more BOT processes or one or more BM processes. There is a single BM process on each machine containing BOT processes (managing filenames for robotting on that machine) and one BM

connected to the QM. On small systems where a single machine performs all tasks, it suffices to have a single BM process that performs both functions.

BOT	A multi-threaded "Robot" process. Each thread simply performs fetches into local files. There can be many BOT processes; the number of threads per process is configurable.
CU	A "Content User", ie, some process that retrieves local fetched data.
RS	A “Registration Server”, to be implemented later, that automates the user registration process.

MySQL tables are used to hold data regarding registered provider information (including the URL roots that can have URLs submitted and maximum fetch bandwidth information), a queue of URLs to be fetched (the QPSDB and QDB tables) and information regarding URLs whose content has been fetched (the LCCDB).

SUBMITTING URLS TO THE SYSTEM: lcccp, lcccpstate

URLs are normally submitted to the UNS by the use of a CP, or Content Provider process. An example CP is provided, suitable for use on Unix systems. It consists of a low-level communicator process that communicates with the UNS and understands the XML protocol; lcccp. <url> containers describing modified URLs are given to this process and they are shipped up to the UNS.

For systems that want to remember state and submit only the URLs that have changed since last time a submission took place, another process is provided that can be used as a filter prior to lcccp; lcccpstate. <url> containers representing the *entire local site* are given to lcccpstate, along with a state file, and lcccpstate will forward only those urls that have been changed, added or deleted, updating the statefile appropriately.

A simple script, file2url.sh, can convert a simple filename to an appropriate <url> container. It can be invoked with 'find' parameters to locate files, or can be given a sequence of filenames on stdin.

An appropriate http:// prefix can be automatically applied to each url using the --urlprefix parameter to lcccp.

Thus a simple file based system that wants to remember state can use the following to provide content to the UNS:

```
    file2url.sh --find docroot -name '*.html' \
            | lcccpstate --statefile=state.txt \
            | lcccp     --uns=remotehost:9000 \
                        --pid=nnn --pwd=xxx \
                        --urlprefix=http://somesite/somedir/
```

In the XML stream sent to the UNS, aside from administration-related containers, is a sequence of <url> containers indicating changed (to be fetched) URLs. Each <url> container is an empty container having the following textual attributes:

curl	A “conceptual” URL. This is used as a “key” to disambiguate the URL from other URLs submitted by the provider. It is normally just the name of the URL itself.
mimetype	The mimetype of the URL content. The LCC can be configured to accept only a limited set of mimetypes.

subtype	A “subtype” is simply a text field of the form “type: value” that can be used in addition to the mimetype to further disambiguate the URL content type on cooperating systems.
burl	A “browse” URL. The URL to use to visualise the content in a browser. This might include a surrounding frame, for example. If omitted, the value of “curl” is used.
furl	A “fetch” URL. The URL to use to fetch the content. If omitted, the value of “curl” is used.
md5	An md5 checksum of the URL content.
len	The length in bytes of the URL content.
mtime	The modification time of the URL content.

The “curl” and “mimetype” attributes are mandatory, the others are optional.

For the attributes “md5”, “len” and “mtime” supplied, a comparison is done with values stored in the LCC db to determine if change has actually taken place. A missing value is taken to indicate change.

It should be noted that the XML stream sent to the UNS can indicate whether the set of <url> containers submitted represents the entire site (a “full set”) or just incremental changes (the normal case). The maximum number of unsolicited “full set” submissions allowed is configurable.

For those that want to communicate using XML directly to the UNS, a description of the protocol between the CP and UNS can be found in the HTML documentation of the LCC.

PROVIDER INFORMATION

Each URL submitted to an LCC comes from a particular 'provider'. The MySQL tables "pidb" (provider information database) and "proots" (provider roots) hold provider-related information.

The information in a pidb row includes the following that should be set to define a new provider:

pid	A unique provider identifier number.
password	A password that must be submitted when the CP connects.
filesmax, spacemax	Quota limits for the provider.
fullsetsallowed	The number of unsolicited “full sets” that can be submitted by this provider.
bandwidth	A maximum number of bytes per second for URL data fetches from this provider.
timeofday	A bitmask giving the allowable times for URL data fetches to occur from this provider. Refer to the section FETCHING below for more information.

priority	A small number (0, 1 or 2) giving a priority for fetches from this provider. Fetches are performed preferentially from a provider having a higher numbered priority.

There are also some other fields that are updated to reflect current provider statistics, namely:

fileused, spaceused	Resource actually used.
lastconnseq, lastconnip	An incrementing sequence number, incremented with each connection and passed back to the CP for informational purposes. The lastconnip is the IP address of the last connection from that CP, given back to the CP for informational purposes.
nurlerrs, nurlproc	The number of fetch errors of URLs from this provider since the last CP connection, and the number of URLS from this provider currently in the LCC fetch queue.

The "proots" table gives the set of valid URL prefixes of valid URLs from this provider. If a CP submits a URL not matching a valid prefix from this set, the URL is not fetched and an error is normally returned.

There will eventually be an automated or semi-automated registration process to add a new provider, but for now one must insert a new row in the pidb and one or more rows in the proots table to add a registered provider.

FETCHING

It is the QM process that actually coordinates URL fetching.

A queue of URL sets to be fetched is maintained in a MySQL table. Each set represents a submission of one or (normally) more than one URL from a CP connection. Each URL in the set is present in another MySQL table.

It is important to note several things about how fetching occurs:

- fetching URLs from a particular provider occurs in a *serial* fashion

- after each fetch occurs, a bandwidth calculation is performed to find out if the next fetch from that provider can occur immediately or after a certain delay.

- after each fetch occurs, a complete calculation of the bandwidth used by the local system is performed to see if the next fetch from anywhere can occur immediately or if all fetching should stop for a period of time.

- a "time of day" bitmask is used to limit fetching on a per-provider basis to the times of day deemed desirable by that provider.

The "time of day" bitmask is determined when the provider is registered.

A rolling time period is used by the LCC that is determined in the LCC configuration file. It is normally either a day or a week with a granularity (also configurable) of one hour. When a bit is set in a provider’s time of day bitmask it implies fetches from that provider can occur in that time period according to the local timezone of the LCC. Any timezone mapping occurs during the registration process.

The two normal configuration options are:

- A "time of day" mask of 24 bits, representing hours in a day where it is legal to fetch.
- A "time of day" mask of 168 bits, representing hours in a week where it is legal to fetch.

All the above functions are performed in the QM. The BM allocates files to be used as destinations of URL content, and each BOT thread simply performs a fetch into a nominated file.

EXTRACTING URL INFORMATION FROM THE SYSTEM

Currently this is all done manually.

The LCC db must be interrogated directly, such as:

```
select furl, mimetype, fetched_machineid, fetched_fileid
      from lccurl
      where fetched_time > xyz;
```

Then, depending on your system, the fetched_machineid, fetched_fileid pairs are mapped to a real accessable filename.

Some simple scripts are available to aid use with various indexers, namely NexTrieve, SWISH-E and ht://Dig.

RELIABILITY

The system is set up so as to present a reliable interface to CPs. Any process that is terminated for whatever reason and restarted will cause other processes to automatically reconnect, for example.

For maintenance reasons, it might be deemed necessary to terminate the QM process (stopping fetching and URL database updates). In this case, CPs can still submit changed-URL sets to the UNS, which automatically buffers the information in low-overhead files until the QM is restarted.

BOT processes can be killed or started for whatever reason, and the BM (and QM) are automatically notified of increased or decreased fetching capacity. Any fetches being performed by a BOT that has been killed are automatically rescheduled.

If a hardware or software failure occurs that results in lost fetched URL data, the data can simply be fetched again. If the hardware or software failure results in lost information regarding the URLs themselves, it is possible to notify one or more CPs (when they next connect) that they are requested to send a 'full set' of their URL information to be fetched. This flag is not reset until the CP at some later date does, indeed, send a full set of URLs.

EFFICIENCY

Some effort has been expended to make the system have as few bottlenecks as possible. It is possible (and intended, for large systems) to have the QM and BM/BOT processes on different machines with local MySQL tables.

A not insignificant advantage was seen by using MySQL statements of the form:

```
insert into table values (a, b, c), (d, e, f),...
```

rather than using multiple statements, one per row.

The amount of caching that occurs of this form is not currently configurable. The caching is done in such a way that if a process is killed containining un-executed (cached) SQL statements nothing special need be done before the system is restarted. In worst case scenarios, the following happens:

- Some URLs that have been fetched may be fetched again.

- Some local file information (representing URLs stored in local files, or local files available for re-use) may be lost, resulting in unused files.

Both cases represent minor problems. A simple verification script can be run periodically if required to locate instances of the second case and to mark the otherwise lost files as available for re-use.

USING THE LCC FOR TRADITIONAL SPIDERING

Although the LCC is primarily targeted at Content Providers that can "push" data, it is still possible to use the LCC in traditional spidering applications.

In this case, "registered providers" are set up as before, indicating sites that can be fetched from. Instead of waiting for URL notifications to be pushed, however, the LCC can be populated with initial fetches of some top-level pages from these sites. Thereafter, periodic scans can be done of the LCC database locating any URLs that have been fetched after the previous scan. These URLs are accessed from the local machine, link extraction is performed, and the links are submitted as if the provider was notifying the LCC of change. The link submissions can be marked to be only fetched if the page does not already exist in the LCC and can also be marked to be silently dropped (rather than generating errors) if the URL to be fetched falls outside the valid bounding tree of the provider.

OUTSTANDING ISSUES

There are a number of points not yet resolved in current LCC system:

- Only HTTP requests can be performed at the moment. There are some hooks internally at some points in the system for other protocols, but nothing workable.

- Security needs to be looked at regarding the content provider connection and provider password. Currently everything (including the password) is transmitted and stored in the clear.

- An investigation should be performed to see if SQL transation support is really necessary to guarantee system integrity, at least as an option.