Large Scale Metadata Harvesting Over Low Bandwidth Connections

Rickert Mulder
Department of Computer Science
University of Cape Town
Cape Town, South Africa
circlingthesun@gmail.com

ABSTRACT
There seems to be a widespread perception that large scale metadata harvesting requires a large amount of bandwidth. In this study a simple Python-based metadata harvester was created and run over a residential broadband connection. Results show that it is possible to build a metadata collection in the order of millions of records in just a few days over such connections.

Categories and Subject Descriptors
H.3.7 [**Information Systems**]: Digital Libraries—*Systems issues*

General Terms
Performance, Measurement

Keywords
OAI-PMH, Low Bandwidth, Harvesting

1. INTRODUCTION
There seems to be widespread perception that large scale metadata harvesting requires a large amount of bandwidth and that a fast Internet connection is essential for such an endeavour [7]. This view is undoubtedly influenced by the Web crawling paradigm. Web-crawling is a very resource intensive process that requires that a complete Web page be downloaded, indexed, then later revisited to check for possible changes. Large search engines have dedicated server farms designated solely for Web crawling. Metadata harvesting by means of the Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH) is regarded as a far more efficient process. Instead of downloading a complete resource from an archive, only the metadata that describes a resource is retrieved and indexed. A record will also never have to be revisited unless its repository indicates that it has been modified. It is easy to see how metadata harvesting would require significantly less bandwidth compared to Web crawling. But exactly how much bandwidth is required to harvest and maintain a metadata database?

The literature does not provide many answers. *Lessons Learned with Arc, an OAI-PMH Service Provider* [6] provides a detailed breakdown of issues experienced while implementing the ARC harvester, but does not make mention of bandwidth consumption metrics. Given this lack of information, questions regarding the feasibility of large scale metadata harvesting over relatively slow connections would largely dependant on rough estimates. Individuals unfamiliar with OAI-PMH are likely to dismiss harvesting millions of records over a slow connection as an unlikely feat.

2. RESEARCH QUESTION
In this study the aim was to determine whether large scale metadata harvesting over low bandwidth connections is feasible, and to discover what problems effect harvester efficiency in this context.

For the purpose of this study a Python based OAI-PMH harvester was created. The design goal of this prototype was to build a simple and robust OAI-PMH harvester to collect various performance metrics during a harvest cycle. The harvester was built from the ground up in order to gain a better understanding of possible complexities during implementation.

3. BACKGROUND
Before discussing the system design a brief overview of OAI-PMH is provided.

The OAI-PMH framework was created out of a need for an interoperable interface to disseminate scholarly research (e-prints) before publication to peer reviewed journals. It was designed to be a simple and efficient low barrier protocol. Although OAI-PMH's roots lie within e-prints, its application is by no means limited to e-prints. OAI-PMH is well suited to disseminate metadata about virually any digital object.

The protocol piggybacks on top of HTTP and is thus Client-Server orientated. Two roles are defined, namely: Data Providers (also Repositories) and Service Providers (also Harvesters). Data Providers maintain a collection of digital resources while Service Providers retrieve metadata describing these digital resources. To enable this interaction, a Data provider exposes an HTTP interface to a network.

Service Providers may harvest metadata from multiple Data Providers and use it to provide services such as search or browse. Popular OAI-PMH based Service Providers include *OAIster* [2] and *Worldcat* [4].

Service Providers make requests to Data Providers by using either HTTP GET or POST methods. Data Providers respond with XML encoded data. Data Providers are required to support six types of requests (also known as verbs). These verbs are ListRecords, Identify, ListMetadataFormats, GetRecord, ListIdentifier, and ListSets. To most harvesters the two most important verbs are ListRecords and Identify.

Identify tells the Data Provider to provide a list of information about itself. This information includes its name, granularity, and earliest date stamp. Granularity specifies the format in which date stamps are encoded. Repositories my use either YYYY-MM-DD or YY-MM-DDTHH:MM:SSZ granularities. Granularity is important since a Data Provider will generally not support dates in a different granularity. The earliest date stamp specifies the date stamp of the earliest metadata record in a collection. A request with an earlier date stamp will result in an error.

ListRecords is the primary method of retrieving records from a repository. A ListRecords request may contain various parameters. A Data Provider receiving such a request is required to return all metadata records that match supplied parameters. A ListRecords request must always specify the metadata format in which records are to be returned in. All repositories are required to at least support the Unqualified Dublin Core metadata format. Additional supported formats may be discovered by using the ListMetadataFormats verb. All metadata records in a repository are date stamped to indicate the time at which they were added to the repository. When specifying from and until parameters only records that were added during this interval will be returned. In the absence of from and until parameters, no lower or upper bounds will be assumed. Data Providers may also classify records by making them part of a set such as Mathematics or Biology. A setSpec parameter may be passed to return only records from a specified set. A repository usually only returns a fixed number of metadata records per request. If all records are not returned, a resumption token is appended to the end of a response. The harvester may request the remainder of a result set by reissuing a ListRecords request, but only passing the resumption token as parameter. The repository will keep returning resumption tokens until no more records are pending. The absence of a resumption token indicates that all records have been sent.

For a more in-depth overview of OAI-PHM, refer to the *OAI-PHM Specification* [5].

4. DESIGN AND SPECIFICATION

Scythe is a simple cross platform OAI-PMH harvester coded for the purpose of this study. Scythe was coded in Python 2.6 and runs on MySQL 5.1 using the InnoDB storage engine.

4.1 Architecture

The basic architecture revolves around two task queues. The first queue (named job queue) is fed by a populate function. This function searches the database for stale repositories, then pushes new harvest jobs to the queue. This job queue has a number of associated harvest threads are tasked to process job objects from the queue. A harvest thread uses parameters from a job object to send a series of requests to a repository. Responses from repositories will contain several records. These records are pushed onto a secondary indexing queue. An indexer thread is tasked to processes all records from the indexing queue by inserting metadata into the database.

Other types of objects are also fed into the two queues. For instance, AddRepo objects that contain new repository information are pushed onto the job queue from where harvester threads pick them off. The harvester thread will execute an Identify request and add retrieved info onto the object and push it onto the indexing queue. The indexer thread will in turn add the new repository to the database.

As various job objects move through the system, statistics related to various stages of a task are accumulated. When these jobs reach the indexer thread, all attached statistics are also indexed in separate relations.

4.2 Interface

Scythe has a basic command line interface that takes several parameters. Command line flags include options for starting a harvest (*-h*) and adding new repositories (*-a*) to the database. When passing the *-a* flag, an XML file with the location of multiple repositories and their harvest constraints needs to be supplied as a parameter. Scythe will attempt to send an Identify request to all repositories in this file. Database entries will be created for all repositories that successfully responds to the Identify request. For testing purposes a second method for repository adding was created, which downloads the *OAI Registered Data Providers List* [1]. The list contains the base URL of over 1200 repositories. By passing the *-f* flag along with a number range in *int:int* format, the specified range of repositories in the list will be added to the database.

4.3 Harvesting

When starting a harvest, harvest jobs for all new repositories are enqueued along with any existing repositories that are out of date. Scythe has a global harvesting frequency that can be modified in settings.xml. Once harvest jobs have been enqueued, multiple threads start processing them concurrently. A harvest thread will keep fetching new jobs until the job queue is depleted.

The indexer thread processes each record individually. If a record has a deleted flag set it indicates that its associated resource was deleted from the repository. Such records contain no metadata and will trigger its removal from the database if it is present. If a record was updated since its last retrieval, it's outdated version will be deleted and replaced with the new version. If a new record has an identical ID and date stamp to a pre-existing one in the database, it will be logged and dropped. The result of all insert operations are logged to the database. After all records in a response object are processed, accrued connection statistics are logged to the

database. This includes the amount of data transferred and the time it took to complete a request.

When incrementally harvesting it is important that the from date stamp overlaps with the start time of the last successful harvest of a given repository. This assures that no new records that might have been added during the previous harvest is omitted. The amount of overlap required is a function of a repository's granularity level. Repositories with day level granularity need to overlap incremental harvests by one day. For second level granularity, only a second overlap is required but 5 seconds is used in practice to compensate for possible timing errors. Overlapping gives rise to possible duplicate records being harvested during incremental harvests. Day level granularities naturally results in more duplicates than second level granularity.

4.4 Logging

Scythe has two logging systems. The one discussed earlier logs mainly connection-related metrics and inserts them into the database. A second logging system prints various debugging messages to a log file. These messages can also be printed to the console by passing the *-d* flag.

The database contains various stored procedures which allow the compilation of harvest reports containing various metrics. Scythe includes a bash script that generates CSV reports on a range of sessions passed as arguments. Reports include the following metrics: session duration, number of repositories harvested, number of records returned, number of requests made, records harvested per second, total data transfer, data transfer rate, a break down of the record insert result, and error frequencies. Typically at the start of a harvest session, all harvesting threads run concurrently. Multiple concurrent harvests make sure the connection is optimally utilised. Should a single connection slow down other connections will generally take up this unused capacity. However, as the harvest draws to an end, not all threads are used. This leads to the connection being under-utilised as the session nears its end. Therefore a peak transfer rate (which we arbitrarily defined as the first 5 minutes) is reported along with the mean transfer rate (which takes into account the whole session duration). It should be noted that data transfer reported is only downstream and excludes any protocol overhead.

5. PROCEDURE

Sample repositories for this study were selected from the *OAI Registered Data Providers List* [1] that contained 1206 base URLs at the time. Only 865 repositories could be successfully identified and were subsequently added to the database. Scythe was set to harvest daily from 6pm via a cron job. An initial harvest was set to only retrieve new records added over the previous 6 days. Scythe successfully executed daily for 9 consecutive days. The maximum number of concurrent harvester threads was set to 20. Non-responsive connections were set to time-out after 10 seconds and could retry a maximum of 3 times. Temporary HTTP redirects were cached and set to expire after 5 minutes. The harvest was run over a 4096 kbps ADSL connection.

6. RESULTS

6.1 Connection Metrics

The theoretical maximum transfer rate for the connection used in this study was 512 Kb/s, without taking protocol overhead into account. A quick benchmark showed that a typical HTTP download from a local server averaged at speed of 420 Kb/s.

6.1.1 Initial harvest

Results reported in *Table 1* show that the initial harvest's peak transfer rate was 366.71 Kb/s. This is slightly lower than the benchmark. This figure, however, represents a series of short concurrent downloads from multiple repositories scattered across the globe. Lower transfer rates could be explained by a number of factors including TCP slow start, HTTP header overhead and slow international uplinks. During this peak transfer rate interval, records were arriving at a rate of 182.25 records/s. The mean data transfer rate over the full duration was 135.25 Kb/s and the mean record retrieval rate was 71.6 records per second. This could be explained by network under-utilisation near the end of a harvest due to lack of concurrency.

6.1.2 Incremental Harvests

Results from eight incremental harvest sessions subsequent to the initial import are aggregated in *Table 1*. Both peak and mean transfer rates of incremental harvests were lower than that of the initial harvest at 239.76 Kb/s and 85.92 Kb/s respectively. Peak and mean record retrieval rates dropped to 138.74 records/s and 51.72 records/s. This drop in efficiency could be the result of an increase in empty result sets as indicated in *Table 3*.

6.2 Record Indexing

Table 2 shows that, on initial import, 99.03% of records harvested were inserted successfully. Successful inserts dropped significantly to 26.42% on subsequent incremental harvests. *Table 2* shows that 14.88% of records harvested were duplicates and, strangely, 58.68% were updates. An investigation of the application code base revealed problem in the section that distinguishes between updates and duplicates. One can assume that the majority of updates shown were actually duplicates. Overlapping harvests is the source of these duplicates. Most duplicates would be from repositories with day level granularity. It would clearly be more efficient to harvest these repositories less often. Invalid records were reported at 0.28% and 0.02% for initial and incremental harvests respectively. Invalid records are typically due to invalid dates being reported and corrupt XML.

6.3 Harvesting Errors

Table 3 shows the frequency of various harvest results. Most results in the table, apart from Success and No Records Match, are in fact errors that lead to the termination of a harvest. No Records Match indicates that a request returned no records. The fact that *No Records Match* occurs more frequently during incremental harvests come as no surprise. No Records Match is not classified as an error, therefore it follows that 92.6% of incremental harvests and 94.28% of initial harvests completed successfully. It appears that the most frequently occurring errors are connection time-outs and XML parsing errors.

Table 1: Connection Metrics

Metric	Initial Harvest (n=1)	Incremental Harvest (n=8)
Number of Records Retrieved	191 778	95 094
Number of Requests	2353	909
Data Transferred (MB)	362.40	157.96
Duration (min)	44:37	30:39
Mean Number of Records per Request	81.5	104.61
Mean Record Retrieval Rate (Records/s)	71.6	51.72
Peak Record Retrieval Rate (Records/s)	182.25	138.74
Mean Transfer Rate (Kb/s)	135.25	85.92
Peak Transfer Rate (Kb/s)	366.71	239.76

Table 2: Indexing Results

	Total Records	Inserted %	Duplicates %	Updated %	Deleted %	Ignored %	Invalid %
Initial	191 754	99.03	0.68	0	0	0	0.28
Incremental	760 568	26.42	14.88	58.68	0	0	0.02

Table 3: Harvest Results

	Incremental (n=6048)		Initial (n=856)	
Result	Occurrences	Freq %	Occurrences	Freq %
No Records Match	3490	57.71	346	40.42
Success	2110	34.89	461	53.86
Connection Timed Out	196	3.24	26	3.04
XML Parsing Error	75	1.24	6	0.7
HTTP 500 Internal Server Error	51	0.84	5	0.58
Bad Argument	45	0.74	5	0.58
Name or Service not Known	19	0.31	0	0
HTTP 503 Service unavailable	13	0.21	1	0.12
HTTP 400 Bad request	13	0.21	1	0.12
Bad Resumption Token	12	0.2	3	0.35
No Route to Host	11	0.18	0	0
Connection Refused	10	0.17	0	0
Connection Reset by Peer	0	0	1	0.12
HTTP 404 Not Found	2	0.03	0	0
HTTP 502 Bad Gateway	1	0.02	0	0
HTTP 504 Gateway timeout Error	0	0	1	0.12

7. EVALUATION

During the initial import, a total of 191 778 records were harvested from 856 repositories in 44.63 minutes, of which 99.03% were successfully inserted into the database. This represents an effective mean rate of 71.6 records/s. Should one succeed in keeping the connection fully utilised, an effective peak rate of 182.25 records/s should be attainable. If Scythe were to harvest unhampered at this rate for 24 hours during an initial import, one should theoretically be able to retrieve approximately 15.7 million records. For comparison, the OAIster database contains just over 23 million records [3]. However, should one only succeed in maintaining an effective rate of 71.6 records/s, one could harvest 6.2 million records in a day. Harvesting a metadata collection rivalling that of OAIster should not take more than a few days over an average ADSL connection.

Maintaining such a collection should prove to be a bit more complex. The strategy of incrementally harvesting repositories on a daily basis proved to be rather inefficient. Results show that only 26.42% of harvested records were successfully indexed. This, and a peak retrieval rate of 138.74 records/s, translates roughly into an effective peak retrieval rate of 36.66 records/s. This is 80% less efficient than the initial harvest peak retrieval rate.

Based on the initial import of 6 days' new records it was estimated that the sample repositories add new records at a rate of 31 963 records per day. An average record in the sample repositories is approximately 2 KB in size. A days worth of records will thus be about a 63.92 MB download. Given a 73.56% duplicate overhead, one would have to download 241.75 MB daily to maintain the collection at our current harvest frequency. After an initial harvest it should thus be possible to maintain our current collection via a 56k dial-up modem.

Managing unnecessary duplication is still a big efficiency issue during incremental harvests, especially with repositories with day level granularity. One solution to this problem would be to simply harvest repositories less frequently. However, some services depend on frequent harvests and up to date data. There is thus a trade-off between an up to date metadata collection and harvest efficiency. Another strategy would be to avoid repositories which use day level granularities and only harvest from Data Providers that support second level granularity. This strategy minimises duplicates but has the downside of excluding some repositories. A duplication management strategy would largely depend on the service one wishes to provide.

8. FUTURE WORK

Scythe is very basic harvester and lacks some functionality that could simplify future research. Future projects could implement a number of useful features. Possible features include: a Web interface, advanced logging daemon, retention of original XML metadata, support for additional metadata formats, XML Schema validation, decouple Scythe into independent reusable components, etc.

9. CONCLUSION

Our research shows that large scale metadata harvesting over low bandwidth Internet connections is very feasible. A run-of-the-mill broadband connection can be used to create a large metadata collection with millions of records in a matter of days.OAI-PMH does, however, seem to be less efficient when maintaining collections because of overlapping harvests. There seems to be a trade off between harvester efficiency and the freshness of a collection.

10. ACKNOWLEDGMENTS

Special thanks to Hussein Suleman for supervising this project.

11. REFERENCES

[1] Oai registered data providers list. http://www.openarchives.org/Register/BrowseSites.
[2] Oaister. http://www.oclc.org/oaister.
[3] The oaister database at a glance. http://www.oclc.org/oaister/about/default.htm.
[4] Worldcat. http://oaister.worldcat.org.
[5] The open archives initiative protocol for metadata harvesting. http://www.openarchives.org/OAI/openarchivesprotocol.html, Dec 2008.
[6] X. Liu, K. Maly, M. L. Nelson, and M. Zubair. Lessons learned with arc, an oai-pmh service provider. *Library Trends*, 53(4):590–603, 2005.
[7] H. Suleman. Personal Communication, Feb 2010.