Parallelising Harvesting

Hussein Suleman

Department of Computer Science, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
hussein@cs.uct.ac.za

Abstract. Metadata harvesting has become a common technique to transfer a stream of data from one metadata repository or digital library system to another. As collections of metadata, and their associated digital objects, grow in size, the ingest of these items at the destination archive can take a significant amount of time, depending on the type of indexing or post-processing that is required. This paper discusses an approach to parallelise the post-processing of data in a small cluster of machines or a multi-processor environment, while not increasing the burden on the source data provider. Performance tests have been carried out on varying architectures and the results indicate that this technique is indeed promising for some scenarios and can be extended to more computationally-intensive ingest procedures. In general, the technique presents a new approach for the construction of harvest-based distributed or component-based digital libraries, with better scalability than before.

1 Introduction

Digital library (DL) systems are rapidly growing in popularity as the technology matures and also because of the advocacy of groups such as the Open Access and Electronic Thesis and Dissertation communities. The effect of this popularity is that there are now more accessible collections, growing at relatively high rates - Lyman and Varian [10] estimated 5 exabytes of new digital information in 2002 alone!

There is a need for tools to manage these large and growing collections and meta-collections and make them accessible to the relevant audiences. However, these tools are not readily available and popular DL systems do not always scale appropriately [7] [6]. While much research has gone into the scalability of Web-delivered DL content (see, for example, [1]), access to services is only one dimension of the management tasks, which typically also include internal data processing for classification, preservation-related manipulation and ingest procedures.

At the same time, digital library tools need to be accessible to users and managers of collections of varying sizes. Keeping this in mind, this study has looked at how the current nature of harvesting of metadata, a popular first step in ingest mechanisms, can be recast to better scale with changes in underlying machine architectures. While harvesting is only one small part of a larger DL architecture, its operation can be parallelised with immediate benefits, without any changes to the data flow that may be needed when other services are parallelised.

2 Background

2.1 Metadata Harvesting

The Open Archives Initiative Protocol (OAI) created the Protocol for Metadata Harvesting (PMH) as a low barrier mechanism for computer systems to exchange metadata on a periodic basis [8] [9].

Metadata is encoded in XML and the exchanges happen as a layer over the HTTP protocol. The owner of the metadata is referred to as the data provider and the provider of services based on this data is referred to as the service provider. The act of transferring metadata from the data provider to the service provider is referred to as harvesting - thus the service provider operates a software tool called a harvester in order to initiate and control the process of harvesting metadata from the data provider.

Harvesting works as follows:

– The service provider executes its harvester to harvest metadata from a data provider. If metadata has not been harvested before, the harvester requests all metadata in a specified format.
– The data provider returns as much metadata as it can reasonably handle and sends back an opaque token, called a resumptionToken, to the harvester as a placeholder for more records.
– The harvester passes the records on to the service provider for ingest into the service provider's system.
– If the harvester encounters a resumptionToken at the end of the record stream, it sends a subsequent request to the data provider with this token as a parameter.
– The data provider sends back an additional chunk of records and a new token if necessary. This process continues in a cycle until all records have been transferred.
– When all records have been transferred, the harvester terminates its activities.
– At regular intervals afterwards, the service provider invokes the harvester to obtain records that have changed since the previous harvesting operation (by specifying the date of that operation). Every harvesting operation uses tokens as before to break up the responses into manageable pieces.

While this algorithm is partly sequential, some of the steps can clearly be carried out in parallel. Before the algorithm can be recast as a parallel one, it is necessary to investigate popular machine architectures that can support such parallelisation.

2.2 High Performance Computing

The following approaches to high performance (parallel) computing were considered for this work:

- Grid computing: refers to collaborative use of computers in a WAN or on the Internet to solve large problems. The EU-based DILIGENT project is investigating the adoption of grids for DL systems [4].
- Multi-processor/core machines: refers to single machines with multiple CPUs and/or multiple processing cores in each CPU. This is an ideal architecture for data-intensive operations such as indexing [1], but arbitrary scaling of the number of processors is usually not possible or prohibitively expensive.
- Beowulf cluster [3]: refers to a collection of machines all in the same location, connected to a high-speed LAN.

During the experimental phase, tests were conducted on a Beowulf cluster and a dual-CPU machine. Grid computing was not considered because of the requirement of a sufficiently fast underlying network, which is not available in the country where this research was conducted (and by extrapolation in some other countries where DL systems are used).

For the cluster it was also necessary to select an appropriate system software layer. openMosix [2] was chosen because it transparently makes a cluster appear as one large system, with no special programming or use of libraries. openMosix is a set of operating system tools that transparently migrate processes to balance the load across all nodes. It allows the use of standard System V IPC mechanisms (message queues, UNIX domain pairs, etc.) for synchronisation, therefore there would be no differences in the software that runs on openMosix, a multi-processor or a uniprocessor machine. In order to make best use of openMosix, however, software applications should be designed as a collaborating set of smaller processes (thus enabling migration of some of them). This technique is similarly an enabler for multi-CPU machines.

3 Parallel Harvesting

3.1 Basic Technique

Most data providers are production-mode digital library systems, with OAI-PMH support as an auxiliary service so processing multiple requests in parallel may be disallowed. Even if possible, there is no mechanism in OAI-PMH to request evenly-sized chunks of records - dates and sets may both be non-uniformly distributed within a collection. The only way to split a stream of records into reasonably-sized chunks is to rely on the data provider to do this by means of its resumptionToken mechanism.

In a parallel harvester, each process requests a chunk of records and passes the resumptionToken to an idle peer so it can get its own data and repeat the process until there are no more resumptionTokens.

A lightweight job scheduler serves not only to distribute harvesting jobs but also to intersperse those with post-harvesting data processing activities, wherever those can be parallelised as well, e.g., merging of sub-indices for a parallel-index-serial-query search engine.

Figure 1 illustrates the process of parallel harvesting and shows the various actors as described. In the illustration each process is depicted as being passed the token in sequence but in practice the scheduler will give the token to whichever node is currently idle (or randomly choose from among the idle nodes if more than one).

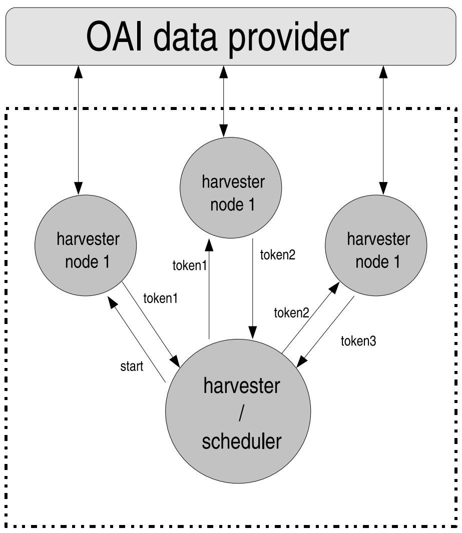

Fig. 1. Parallel harvester components and interaction

3.2 Distribution and Synchronisation

When harvesting begins, multiple processes are spawned(using fork). These processes are distributed as necessary to the various CPUs or cluster nodes (worker nodes) by the operating system, in a best effort to balance the load without application-specific information. Processes were chosen over threads because threads cannot be easily migrated in some parallel architectures.

The scheduler uses a work pool and processor farm approach to manage jobs [11]. The work pool is initialised to contain the usual first harvesting operation to obtain all records that have been changed since the date of the last harvest. The scheduler also maintains a set of flags to indicate which worker processes are busy. When there is at least one idle worker process and at least one job in the pool, the job is dispatched to the worker process (using a unix socket for communication). The worker process will then harvest the next chunk of data from the data provider and send the resumptionToken back to the scheduler as soon as it is obtained. The worker continues to process the data (e.g., create indices or reformat for ingestion) and sends a message to the scheduler when it is done with the job. The scheduler, in the meanwhile, could have signalled another worker process to deal with the new job that is in its pool. Thus, if the post-processing of data is time-consuming, the scheduler ensures that this is done in parallel, while the harvesting operations do not themselves overlap.

Figure 2 shows the overlapping of harvesting and processing operations in a parallel harvester, as compared to the traditional sequential harvester. Jobs with significant time spent on post-processing fare better in the parallel scenario.

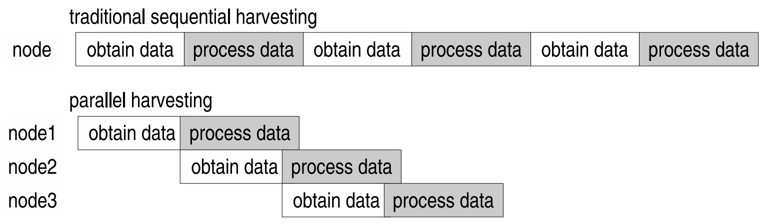

Fig. 2. Timing of sequential and parallel harvesting

4 Evaluation

In order to evaluate the efficiency of parallel harvesting, the platform was varied and tests were conducted for varying numbers of worker processes. Since the aim of this work was to support parallel harvesting irrespective of the underlying architecture, the operating system did all task allocation and/or migration implicitly.

Table 1 lists the different platforms used during testing and how they differed. The last column refers to whether or not the OAI-PMH data provider was on the same machine (if there was a single machine). Machine4 is so named because it is a single machine within the Simba cluster.

4.1 Typical performance

First, each platform was tested with a harvester that performed inverted file indexing of the metadata, with each metadata chunk kept independent and the inverted files written to disk after processing. Indices were created for each metadata field as well as the whole record, and for individual stemmed and stopped words as well as the whole contents of each field. This is a typical first operation performed by the indexing portion of a search engine.

For the Machine4 and Simba platforms, the data was stored remotely using NFS. All other platforms stored the data locally.

Table 1. List of platforms, and their characteristics, used for experiments

Name	Machine Description	OS	Data Source
Laptop	Centrino 1.5GHz	Linux 2.6.12	local
Banzai Local	Dual Pentium 3GHz	FreeBSD 6.0	local
Banzai	Dual Pentium 3GHz	FreeBSD 6.0	remote
Machine4	Pentium 3GHz	Linux 2.4.26	remote
Simba	8x Pentium 3GHz, connected with Gigabit Ethernet	Linux+openMosix 2.4.26	remote

Figure 3 shows the time taken for harvesting and indexing for each of the different platforms, each tested with 1, 2, 4, 8, 16 and 32 worker processes.

Machine4 and Laptop, as expected, did not perform as well as Banzai because of the number of CPUs. These single CPU machines, however still register an improvement in performance when multiple processes are executed simultaneously, presumably because of the overlapping of IO with computation.

Banzai and Local Banzai take approximately half the time of their single CPU counterparts. When the number of processes increases drastically, Banzai performs better, probably due once again to Local Banzai having to serve its own data provider in addition to its harvesting and indexing operations.

Having 8 CPUs, it could be expected that Simba will provide the best performance at all times. However, the data communication when processes are migrated to other nodes takes its toll, especially when there are few processes and the load is not high. For a very small number of processes, openMosix has more idle processors than busy ones so spends a lot of time moving processes around, without taking into account that processes may have substantial data footprints as well. As the number of processes increases, it is easier for openMosix to spread the load and maintain this even spread without further migrations. Thus, for more than 4 processes, Simba outperforms the single CPU platforms but because of the data communication for process migration, remote disk access and synchronisation, the multi-CPU machine still outperforms the cluster-based solution.

4.2 Varying of Workload

The results of the first round of performance trials did not favour the cluster and it is hypothesised that this is because of a small workload and excessive remote data access. To test that the workload is in fact the reason why a dual-CPU machine outperforms an 8-node cluster, the workload was varied and additional tests were conducted.

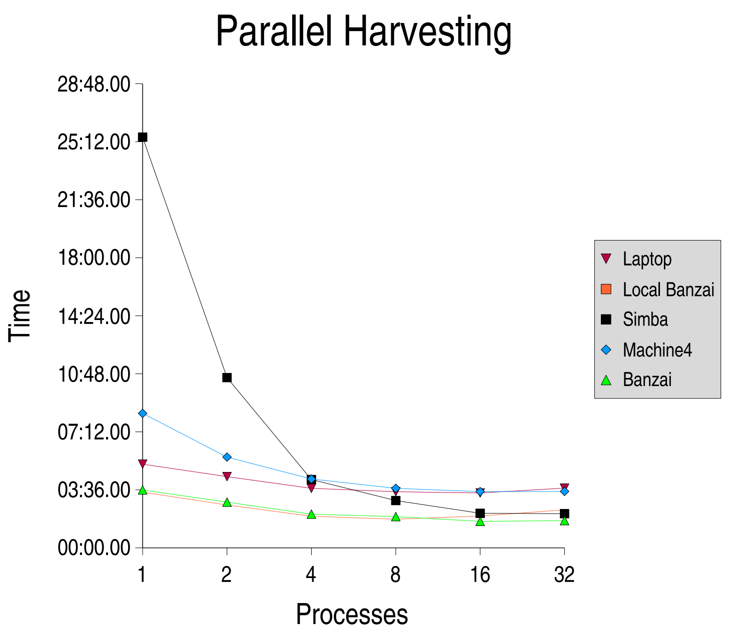

Fig. 3. Typical performance of different platforms for an indexing task

First, to remove any bias, only those platforms with remote data providers were considered. Then, the harvesters were set up to perform each of the following tasks on harvested data:

– index and commit to disk as before;
– index only; and
– index, perform some additional CPU-intensive calculations, and then commit indices to disk.

The results from these tests are shown in Figure 4. In the case of Indexing, Simba and Banzai perform equally well because the computational load is not high. With Indexing+Committing, Banzai outperforms Simba because of local disk access, as before. However, as the computional load is increased in the Indexing+Committing+Computing test, Simba begins to perform better than Banzai. This result shows that while disk-intensive operations may be better suited to a multi-CPU system, as the load of computational operations increases, a cluster of machines may offer a reasonable solution. From a digital library perspective, a cluster of machines may offer cost-effective possibilities for processing data for indexing, classification, automatic extraction, pattern detection and similar tasks.

Now, consider the data from this experiment depicted from the perspective of each machine rather than the tasks performed (see Figure 5). It is clear that

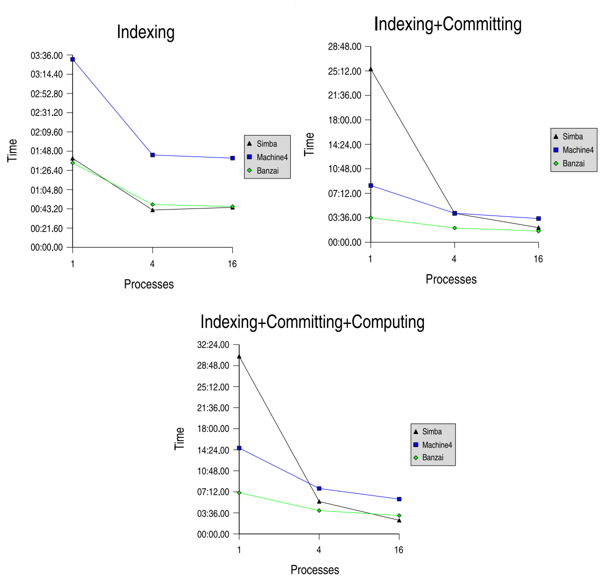

Fig. 4. Per-platform analysis of harvesting times, for each workload

a cluster of machines (Simba) has the advantage that for a sufficient number of processes, a higher computational load does not significantly increase the wallclock time. For the single processor and dual-processor machines, a higher computational load still results in a much higher processing time.

5 Conclusions

This work has begun to look at how existing digital library architectures can be made more scalable. The results naturally do not work for all scenarios and the performance may degrade in systems with large numbers of CPUs, for the given simple approach to parallelisation.

Nevertheless, the experimental validation shows that storage-intensive services can benefit from multi-processor machines, while computation-intensive services may work adequately on the more cost effective Beowulf clusters. In addition, the restructuring of OAI-PMH harvesters to include parallel network access and post-processing yields performance benefits on even single processor machines! In all cases, these gains were made purely by redesigning the harvester, without any modifications to the OAI-PMH and without adversely impacting the data provider. Also, the harvester is architected to work reasonably well on a single processor machine and easily scale up to make use of additional resources if they are available.

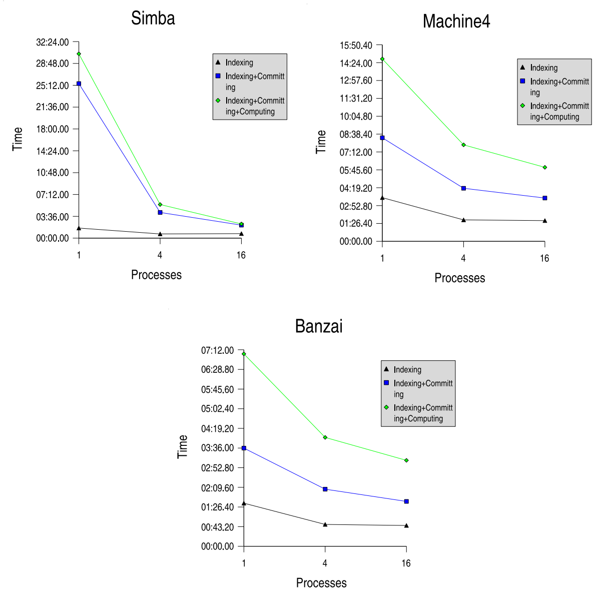

Fig. 5. Per-workload analysis of harvesting times, for each platform

6 Future Work

For distributed digital library systems, these experiments have shown that there is benefit in paralleling even the most basic harvesting operation. The next step is to parallelise the various processing operations that take place within a digital library system, including indexing and querying. Early work with the parallel harvesting framework has shown that the scheduler can be used to manage multiple types of jobs simultaneously - thus some nodes could be harvesting and post-processing while others could be merging indices. For systems where multiple services require different processing operations, it is possible to use a computational pipeline, with each stage performing a particular operation.

There were some problems with data and process movement in openMosix. In looking at alternatives, the distribution of processes and data will depend on the specific data flow patterns of a digital library system. Dongarra et al. [5] emphasise that parallelism is only a part of the solution and that data flow must be considered. Further work is therefore needed to determine what the data flow patterns are and how best to optimise the distribution of processes, communication among processes and disk access patterns for typical DL services.

Eventually, in order to scale digital library systems arbitrarily, it may be necessary to rethink the fundamental nature of data storage, movement and processing in digital library systems. The OAI-PMH data provider enforces a notion of ownership or stewardship of data, but quickly becomes a bottleneck in large scale collections. Data ownership may need to be redefined in its relationship to data storage and locality so that scalable services have optimal access to data when needed.

7 Acknowledgements

This project was made possible by funding from University of Cape Town, NRF (Grant number: FA2005041200001), NRF-THRIP, Telkom and Siemens.

References

1. Andresen, Daniel, Tao Yang, Omar Egecioglu, Oscar H. Ibarra, and Terence R. Smith (1996), "Scalability Issues for High Performance Digital Libraries on the World Wide Web", Technical Report 1996-03, Department of Computer Science, University of California Santa Barbara, March 1996.
2. Bar, Moshe (2003), "openMosix, a Linux Kernel Extension for Single System Image Clustering", in Proceedings of Linux Kongress: 10th International Linux System Technology Conference, 15-16 October, Saarbrücken, Germany.
3. Brown, Robert G. (2004) Engineering a Beowulf-style Compute Cluster, Duke University Physics Department. Available http://www.phy.duke.edu/ rgb/Beowulf/beowulf_book/beowulf_book/index.html
4. Diligent (2006) A Digital Library Infrastructure on Grid Enabled Technology. Website http://www.diligentproject.org/
5. Dongarra, Jack, Ken Kennedy and Andy White (2003) "Introduction", in Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, Andy White (eds): Sourcebook of Parallel Computing, Morgan Kaufman, Amsterdam.
6. Haedstrom, Margaret (2003), "Research Challenges in Digital Archiving and Long-term Preservation", NSF Post Digital Library Futures Workshop, 15-17 June 2003, Cape Cod. Available http://www.sis.pitt.edu/ dlwkshop/paper_hedstrom.html
7. Imafouo, Amlie (2006), "A Scalability Survey in IR and DL", TCDL Bulletin, Volume 2, Issue 2. Available http://www.ieee-tcdl.org/Bulletin/v2n2/imafouo/imafouo.html
8. Lagoze, Carl, and Herbert Van de Sompel (2001), "The Open Archives Initiative: Building a low-barrier interoperability framework", in Proceedings of the ACM-IEEE Joint Conference on Digital Libraries, Roanoke, VA, USA, 24-28 June 2001, pp. 54-62.
9. Lagoze, Carl, Herbert Van de Sompel, Michael Nelson and Simeon Warner (2002), The Open Archives Initiative Protocol for Metadata Harvesting – Version 2.0, Open Archives Initiative, June 2002. Available http://www.openarchives.org/OAI/2.0/ openarchivesprotocol.htm
10. Lyman, Peter, and Hal R. Varian (2003) How Much Information 2003?, University of California. Available http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/index.htm
11. Wilkinson, Barry, and Michael Allen (1999) Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice Hall, New Jersey.