International Journal on Digital Libraries manuscript No.
(will be inserted by the editor)

Hussein Suleman · Christopher Parker · Muammar Omar

Lightweight Component-Based Scalability

Abstract Digital libraries and information management systems are increasingly being developed according to component models with well-defined APIs and often Web-accessible interfaces. In parallel with metadata access and harvesting, Web 2.0 mashups have demonstrated the flexibility of developing systems as independent distributed components. It can be argued that such distributed components also can be an enabler for scalability of service provision in medium to large systems. To test this premise, this article discusses how an existing component framework was modified to include support for scalability. A set of lightweight services and extensions were created to migrate and replicate services as the load changes. Experiments with the prototype system confirm that this system can in fact be quite effective as an enabler of transparent and efficient scalability, without the need to resort to complex middleware or substantial system re-engineering. Finally, specific problems areas have been identified as future avenues for exploration at the crucial intersection of digital libraries and high performance computing.

Keywords scalability · service · component · cluster

This project was made possible by funding from University of Cape Town and NRF (Grant number: GUN2073203).

H. Suleman
University of Cape Town
Tel.: +27-21-650-5106
Fax: +27-21-689-9465
E-mail: hussein@cs.uct.ac.za

C. Parker
University of Cape Town
Tel.: +27-21-650-2663
Fax: +27-21-689-9465
E-mail: cparker@cs.uct.ac.za

M. Omar
University of Cape Town
Tel.: +27-21-650-2663
Fax: +27-21-689-9465
E-mail: momar@cs.uct.ac.za

1 Introduction

Digital libraries often have to deal with scalability concerns as either information collections grow or a greater need for services is expressed by users. Many researchers have already identified the growing size of digital collections as a major problem that needs to be addressed [2,10,13] and a growing size of data invariably means more processing, both internally and for end-user services. Such data collections may experience quality of service problems as a result of both increased user requests and larger data collections. Content delivery networks (CDNs) such as Akamai [28], for example, have solved this problem by making use of distributed networks of servers to deliver content to consumers. The drawback of such CDNs is that they merely provide a mechanism to deliver content and do not provide computational support for purposes such as indexing and querying, which are sufficiently expensive during peak periods to force a system to shutdown. The infamous public Web launch of Encyclopaedia Britannica, which was completely swamped by user requests and forced to shut down operations [9], is a reminder that developers of digital collections need to build scalability into their systems to seamlessly handle increases in demand for services.

More recently, analyses of systems such as DSpace and EPrints have brought to the fore some of their inadequacies - notably, their poor support for large collections and the potential for long processing times [6]. These issues are actively being addressed by many communities, with much promise demonstrated by the Fedora [22] and Greenstone [7] communities. In the former case, there has been an emphasis on building stable core components while the latter project has concentrated more recently on Web interfaces to create separable system components. These mutually-compatible approaches suggest that maybe discrete components are part of the solution to achieve a higher level of scalability.

A number of component and service frameworks have been developed in the digital library community, includ-

ing Dienst [14], OpenDLib [5] and ODL [23]. These frameworks and related work in defining protocols for inter-component interaction all suggest that DL systems should be built as collections of cooperating components rather than as monolithic systems, in keeping with current best practices in modern software engineering. It may be possible to then extend this notion and use components that operate in parallel on multiple machines as the basic building blocks of a high performance digital library system.

It is proposed in this article that such parallel processing in a component-based digital library system can be effective in making the systems transparently scalable and extensible as more resources are required, and without much overhead. Two fundamental scalability problems are addressed by giving components the ability to migrate and replicate. Migration may be defined as the movement of a service from one machine to another while replication is when a copy of the service is created on another machine. Migration may be beneficial when a machine is overloaded due to multiple co-existing services. Replication may be beneficial when a machine is overloaded due to a single very popular service. Figure 1 illustrates the difference in these approaches.

Migration and replication on-demand are necessary when the digital library grows over time. This is a crucial issue when digital library systems are initially developed and resourced for small collections, as is typical for many fledgling preservation projects. It must be possible to then add processing power without requiring a re-engineering of the system or a switch to a different software platform.

The rest of this article begins with an overview of recent and relevant work in digital library architectures and scalable systems. It then presents the architectural changes needed to convert a simple static component model into a dynamic scalable framework. Initial experiments with a reference implementation of this system are discussed. Finally, the outcomes are analysed and the implications for future systems discussed.

2 Background

This section aims to give some background into existing digital library systems and component frameworks, high performance computing models and scalable service models, in order to provide a foundation and context for the work that was done.

2.1 Digital Library Components and Systems

Numerous efforts have been made to create component frameworks. While there is little agreement on a component standard for digital library systems, some systems have emerged as a platform for experimentation, mostly based on a common understanding and acceptance of Web and metadata harvesting standards.

2.1.1 Open Digital Library

The Open Digital Library (ODL) project is characterized as an attempt to infuse interoperability into all aspects of the digital library [23], and make the provision of services as simple as the provision of data, effectively facilitating the development, management and interoperability of digital libraries. This is achieved by making use of a component-based approach, where components communicate by using standard network protocols wherever possible and custom protocols where necessary, while attempting to deviate as little as possible from existing best practices. This approach allows different components to be run on different machines, yet still function as one system. This powerful, modular and distributed design lends itself well to the field of cluster and grid computing, which is discussed in the next section. These features of the ODL framework make it a suitable candidate for this research.

2.1.2 OpenDLib

OpenDLib [5], like ODL, is a component-based digital library framework. These components - or services - are linked to each other using the OpenDLib protocol (OLP), an open protocol in which URLs are embedded in HTTP requests.

Like most current systems, OpenDLib promotes interoperability between itself and other digital library systems - it allows data to be harvested using the OAI-PMH [15] and it can ingest data from other systems using the same mechanism.

Another feature of OpenDLib that makes it more scalable than other digital library implementations is that it has support for load balancing among components or services. Where components are replicated to multiple servers, a service known as the Manager Service calculates the optimal component instance to which a request should be routed. The decisions for this routing are based on workload, availability and bandwidth metrics that the Manager Service maintains. Each OpenDLib component keeps data that the Manager Service collects in order to use for this purpose. Although OpenDLib promotes scalability by making use of load-balancing, it has no inherent notion of component migration or transparent increasing of resources available to a DL system.

2.1.3 Dienst and Fedora

Dienst is a component- or service-based distributed digital library where components communicate by making use of an open protocol, making it possible to combine services in innovative ways [14].

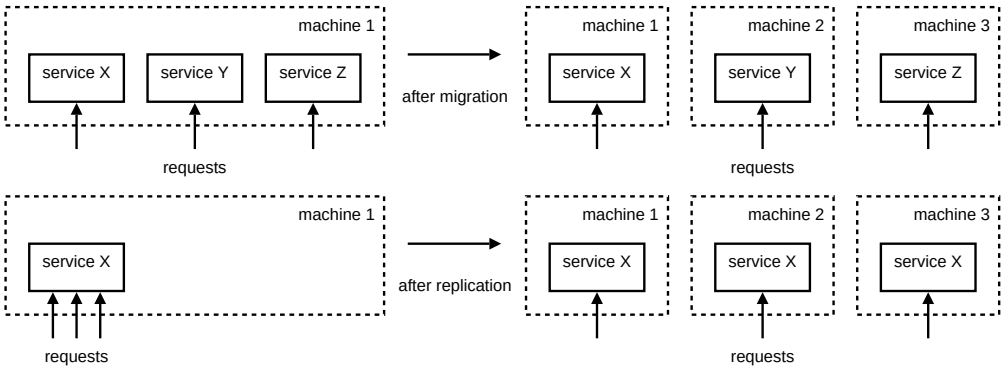

Fig. 1 Migration and replication of components

The distributed and component-based nature of the Dienst architecture ensures that large collections may be efficiently processed, for example when large document collections need to be indexed.

The service model of digital library systems has led more recently to the Fedora project [22], which has developed a well-defined interface to interact with a digital repository. The reference implementation of Fedora demonstrates the flexibility available to system designers and claims to be reasonably scalable (tested to 10 million objects). However, Fedora does not have a built-in mechanism to scale further as collections or service requests grow.

2.1.4 aDORe

The aDORe digital object repository developed by the Los Alamos National Laboratory (LANL) was designed to host a vast number of digital objects. Following a standards-based, modular approach, this system is made up of a number of components where interaction between these components is protocol-based [30].

Unlike other digital library systems that make use of externally hosted content, the aDORe system hosts the entire collection locally, thus removing the need to route users to external sites in order to acquire content, thereby improving control over digital assets and ensuring a high level of object availablity. In terms of scalablity, the aDORe system has data replication support in which it replicates digital objects ingested using the standard OAI-PMH protocol into Autonomous Repositories. A module known as the Identifier Locator can then be used to locate a particular object, upon request, from one of these repositories.

The modular, scalable approach taken in the design of the aDORe system should ensure that larger collections are able to be supported in the future. However, the current implementation cannot be run in a distributed Web environment, although this was mentioned as a possible future addition.

2.1.5 Greenstone

Greenstone is a system produced by the University of Waikato with the aim of making digital library software easy to deploy and use by a wide variety of organisations, including those in developing countries [7].

The Greenstone software has undergone many changes over the past years, probably the most noticeable of which is, with the release of Greenstone 3, the transition to a component-based approach. This component-based approach has provided Greenstone with the ability to operate as a distributed system, with well-defined SOAP interfaces to communicate among parts of the system. Now it is possible to separate the interface from the underlying service layer, and create systems that are both more flexible and remotely deployed.

While it is possible to connect together multiple Greenstone systems, it is not possible to duplicate popular parts of a single Greenstone system.

2.2 High Performance Computing

Many digital libraries contain very large collections of digital objects. Providing services over such large collections can be made scalable - and sometimes even possible - by leveraging the power of high performance computers, often referred to as supercomputers.

High performance computing can be accomplished using a number of different approaches - of these cluster and grid computing are very popular because they can be built using commodity computers.

This section aims to give a short overview of each of these classes of high performance computing, as they relate to digital library systems.

2.2.1 Cluster computing

A cluster computer is a group of machines that are located in a single geographical location (although this requirement is not strongly maintained), have high-speed network interconnections, have low-overhead and high

availability, are dedicated resources and are built in such a way as to facilitate expansion.

Clusters are used to solve a variety of problems, however, the most common use is when large problems are broken down into smaller parts that can then be divided amongst the nodes of the cluster. The way in which these problem divisions are made is somewhat of an art, however there are many tools available to assist application programmers with this task. An application built using these tools is known as a cluster-aware application as it is specifically designed to work on this parallel platform. Tools such as MPI [17] and PVM [27], which enable message passing for communication among nodes, allow an application programmer to make use of parallel resources by making high-level function calls. This software is installed on all nodes in a cluster in order for the components to communicate with one another while solving a problem collaboratively.

On the other hand, clusters are just as widely used for parameter studies, where the same application is used to perform computation over different sets of data. This type of application, called a cluster-unaware application, derives its parallelism from the data and not the application. No tools such as MPI or PVM are necessarily required for such types of cluster applications.

In both cases, a cluster management suite (such as OSCAR [18] or ROCKS [20]) or a scheduling system (such as CONDOR [29] or PBS [11]) is required in order to schedule jobs and monitor their progress on cluster nodes.

Clusters provide an ideal environment for scalability of digital library systems because the resources are dedicated and the high-speed networks support movement of data and computation as required. Cluster-unaware applications can provide service-level scalability while fine-grained functionality, such as is required by distributed indexing, can be more cluster-aware.

2.2.2 Grids

A grid is a collection of computers on a wide area network that are made to cooperatively work towards solving problems. Machines on a grid are generally owned by multiple parties, are not dedicated as in the case of clusters and also do not necessarily belong to a cluster. Furthermore, grid resources usually include machines that have heterogeneous architectures and operating systems as well as hardware specifications that range from ordinary desktop machines to servers.

Grids are an attractive option to many organisations because they are a cheap alternative to potentially expensive clusters - existing equipment and equipment owned by external parties can be utilized.

Grids have many drawbacks, including an inherent instability as components are not necessarily controlled by a central authority - owners of grid resources may opt to remove their machines from the grid at any time and hence any computation in progress may be affected. The heterogeneous architectures and operating systems also complicate the design of grid middleware and application software. Further, the slow communications links restrict the range of applications suited to grids.

These disadvantages notwithstanding, the potential computational power that can be attained with grids can far exceed the capacity of even the largest cluster. This is evident in projects such as SETI@Home [1] and the World Community Grid [12], where such studies would be almost impossible to perform with any other high performance architecture.

The EU-funded DILIGENT project is currently looking into how to combine the architectures of digital library systems and grids to provide digital library designers with a flexible and scalable framework [4]. Based on initial extensions to OpenDLib and the European grid and Globus toolkit [8], DILIGENT is one of the few attempts to create scalable systems based on a strong component framework foundation.

The study reported on in this article is similar to DILIGENT, but it diverges by looking into the feasibility of a lightweight framework rather than the often complex APIs provided by existing grid toolkits.

2.2.3 Summary

High performance computing systems such as clusters and grids can provide a solid grounding for solving digital library scalabilty problems. In this area, digital libraries would be more concerned with the distributed storage capabilities that clusters and grids provide as compared to their computational benefits. However, distributed querying and indexing systems, which by their nature are computationally intensive problems, can benefit greatly from high performance computing installations.

3 System Design

In order to validate the premise that a coarse-granularity component-based system can provide scalability as a consequence of its distributed nature, a prototype system was developed as an extension to an existing component framework.

The system was designed to require a minimal number of changes to the core framework. The ODL framework and components were adopted as being typical component technology - naturally the ideas are equally appropriate to other frameworks. Static DL systems were then constructed using instances of these components and these then were modified to introduce additional functionality needed to support scalability of services.

3.1 The ODL Framework

Before discussing the scalability extensions, it is essential to present some fundamental ideas of the ODL framework.

An ODL digital library system is made up of a set of independent software tools that communicate using standard or custom-developed Web-based protocols [26]. In the discussion that follows, a component refers to the installed files of the software tool. An instance is created when the software tool has been configured to provide services, typically via a Web interface that is its service endpoint. In object-oriented terminology, the components are classes and the instances are objects. Most ODL components may have multiple instances - for example, there can be multiple search engines using one set of shared libraries.

ODL instances communicate among themselves using protocols such as the OAI-PMH [15], which is used by a search engine to obtain data from an archive. This is a trivial but popular use case. Other components have been developed to provide browse services, recommendations, peer review, etc.

In the typical system, a user experience layer (or interface) communicates with back-end instances using RESTful ODL protocols, with requests encoded as HTTP GET operations. The back-end instances may then communicate with other instances in the process of fulfilling the request. The XML-encoded response is finally sent back to the user experience layer, where it is parsed and acted upon.

ODL components have been demonstrated to be at an appropriate granularity to support a wide range of systems and do so with a reasonable level of efficiency [24].

3.2 Design Overview

Figure 2 illustrates the high-level architecture of part of a component-based DL system, with subsystems needed for scalability services included.

At its core, this DL system is made up of two instances - a search engine and an archive - that communicate with each other in order to provide services to end-users. For simplicity, the diagram does not depict the user experience layer (or interface), or other parts of the larger DL system.

There are two scalability-related subsystems: the service resolution subsystem, that is used to map logical to physical URLs for services; and the load balancing subsystem, that is used to redistribute the instances to make better use of computational resources.

Each of these is discussed in the following sections.

Table 1 Parts of Logical Naming Scheme

Description	Purpose	Example
user name / instance owner	to support multiple overlapping systems sharing one cluster	james
component name	to bind instances to their software implementation	ODL_search
version number	to uniquely identify instances with different machine interfaces	1.1
instance name	to select an instance from a pool associated with a component	ndltd

3.3 Service Abstraction

The first step in achieving scalability is to separate the instances from the physical machines they reside on, thus making it possible to relocate instances. Each service instance is therefore given an abstract or logical service endpoint name, comprising: user name, component name, version number and instance name, as described in Table 1.

Thus, a typical logical name for an instance could be:

```
james/ODL_search/1.1/ndltd
```

This logical service endpoint must then be mapped to a physical service endpoint or URL using an appropriate resolution service. This is analogous to the name resolution provided by the DNS system for machine names mapped to IP addresses, or DOIs mapped to URLs for digital objects.

3.4 Service Resolution Subsystem

Service resolution was modelled on the DNS system, with resolution of logical service names to physical service URLs taking place at a resolver in each node, but with a lightweight global registry maintaining information about all mappings.

In Figure 2, when the instance named *search* on ServerA wishes to connect to the instance named *archive* on ServerB, it first connects to the resolver on ServerA (1) to request resolution of the logical service endpoint. If the physical service is local to the machine, the resolver sends back the details of the physical service endpoint (4). In the figure, however, the physical service is on a remote machine, so the resolver contacts the registry (2). The registry then looks up the physical service endpoint and sends it back to the resolver (3) on ServerA, which in turn sends it back to the search instance (4). The search instance can now connect to the archive instance (5).

Each resolver maintains a list of instances and associated mappings of logical to physical names on its local

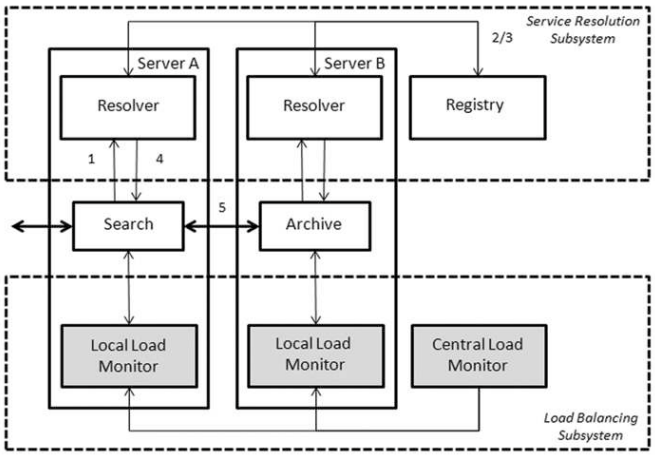

Fig. 2 Architecture of service resolution and load balancing

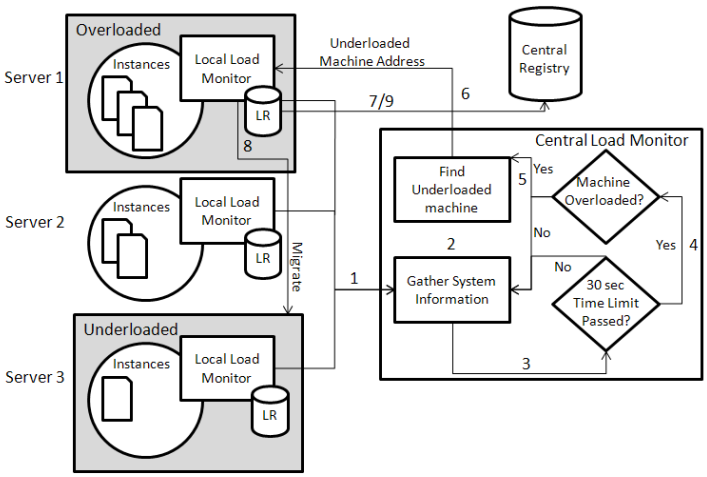

Fig. 3 Load Balancing Subsytem Process Flow

machine and communicates changes to the registry. The registry maintains a global state mapping of logical to physical services endpoints.

While only a single central registry was used in the prototype, replicating this using techniques analogous to DNS and similar systems should pose no significant problems.

3.5 Load Balancing Subsystem

In sections 3.4 and 3.5, discussions on how to separate component instances from the physical machines on which they reside and then how the physical location of a service endpoint is abstracted away have been given. With these mechanisms in place, components may be migrated or replicated in order to balance the workload in a cluster of machines. This load balancing phase leads to increased scalability as component instances can be shifted around in order to minimise load on single machines. This section will give an overview of how a load balancing scheme was implemented in the experimental system.

3.5.1 Architecture

In order to determine whether or not a machine is overloaded and hence needs to be load balanced, each service instance maintains a counter of the time taken to process requests. A local load monitor on each machine, see Figure 3, then tracks the relative use of different instances. The local load monitor also keeps track of the overall system load on the machine, including the load caused by processes other than ODL instances, acknowledging that clusters are often multi-purpose systems rarely devoted to a single application.

In addition to a local load monitor on each machine, a central load monitor polls each local load monitor at regular intervals (1) in order to gather load information (2). Based on the average load over all machines, the central load monitor determines if any machine has a significantly higher than expected load (4). A significantly higher load is nominally defined as more than two standard deviations from the mean - though this was tested experimentally, as discussed below. If an overloaded machine is identified, the central load monitor identifies a corresponding underloaded machine (5) - this is defined as the machine with the minimum load average. A message is then sent to the local load monitor (6) on the overloaded machine to redistribute some of its load to the underloaded machine. The decision on how to resolve the load imbalance is made at the local level, while the decision that an imbalance exists is made globally, thus minimising global-level operations.

The overloaded machine then either replicates or migrates one of its instances to a different machine.

- If there is only one instance on an overloaded machine, replication is selected, as it is assumed that the load is caused solely because the instance cannot handle the requests and not by multiple instances coexisting.
- If there is more than one instance on an overloaded machine, migration is selected because the load could be caused by too many busy instances existing on a single machine.

Once migration or replication has been triggered, the instance is sent a message to initiate a transfer. The instance immediately informs the central registry (7), via the local resolver, that a transfer is in progress and the central registry and local resolver will temporarily delay any incoming requests for that logical service endpoint. The instance then creates a package of its configuration files and copies it across to the destination server (8) where it is unpacked and installed. As a final step, the registry and resolvers are informed that the process is complete (9). The registry unblocks the logical service endpoint, either with a new physical endpoint or an ad-

ditional physical endpoint, depending on whether migration or replication was triggered respectively.

Static files are copied across but databases are accessed remotely. This is a complex problem to solve in general as costs depend on data size as well, initialisation penalties, etc. A more advanced algorithm could dynamically quantify the cost of remote access as compared to the cost of data transfer but this may be significantly more complex.

Finally, with both service resolution and load balancing in operation, the instances are able to redistribute themselves within a cluster of computers to make better use of computational resources, without any explicit actions or reconfiguration of the DL system.

3.5.2 Performance

The criterion for migration, and even replication, can lead to thrashing if the system is very lightly loaded or there are lots of instances with no clear redistribution that will better optimize resource usage. This is apparent in transparent high performance computing environments such as OpenMosix [3], where the system works well under heavy loads but can perform poorly under light loads [25]. To avoid such problems, a combination of strategies have been used to dampen the effects of the dynamic system:

- Migration/replication may be triggered only at discrete intervals of 30 seconds, to prevent rapid migration affecting overall system performance and to allow the system to stabilize before further redistribution.
- Migration/replication may only involve one instance at a time, to prevent multiple operations attempting to resolve a single problem.
- The effects of a migration are first simulated by the local load monitor, using information about the source and destination nodes. Since the local load monitor knows the relative load of instances and the absolute system load (on both nodes), it can simulate the change in system load that will occur if each instance is migrated. The criterion that is tested is: will the candidate migration result in both source and destination nodes having a load average that is closer to the global mean? If this is true, the best candidate is chosen for migration - otherwise, it is assumed that migration will not result in improvement and the operation is cancelled.

4 Experiments and Results

4.1 Redistribution of Load

The prototype was evaluated for its ability to successfully balance the load in a small cluster of machines. 3 machines were set up to connect with a central registry and central load monitor. A simple DL system was set up, with instances for a search engine, browse index and archive, all initially located on a single machine. 8 simultaneous clients were then used to generate 1024 browse requests while another 8 simultaneous clients generated 1024 search requests.

Because of the imbalance in resource use, the system quickly separated the three instances such that each machine only hosted one instance. The resulting response times over the period of all 2048 requests is indicated in Figure 4 and Figure 5.

It is clear from the results that there is a significant improvement in performance for both search and browse operations because of a better distribution of instances to machines. The initial requests have a longer response time because the load on the system had not yet been spread across all the machines. After a short period of overloading, around request 113 on the graphs, the load balancer migrates instances to spread the load and both services are then able to operate with faster response times.

This experiment was repeated for combinations of experimental parameters [19] from the following:

- How often to redistribute the load on the system (varied between 10 and 50 seconds)
- When to select a node as overloaded (varied between 1 and 5 times the standard deviation from the mean global load)
- The rate of incoming requests (uniform, sawtooth, random, periodic)

In all cases, the combinations of parameters that yielded the best system response rate changed over the course of the sequence of requests. However, the difference was always minimal, thus there was no clear reason to choose a particular set of parameters over another. In an operational system, these parameters could be explicitly initialized or dynamically changed as the overall system load changes - for example, a relatively stable system would not need as much redistribution and the criteria for overloading can be set to be lower.

4.2 System Overheads

Any system that uses indirect connections may incur the penalties of extra communication to monitor and manage global state and transfer data and processes among machines. This communication was monitored and analysed to determine the exact impact on overall system performance of the additional layers introduced to achieve scalability.

Over a series of 400 requests, the search instance, on average, sent a 90 byte request to the resolver and received a 393 byte response. The average time taken to process the requests was 0.079 seconds. Then, the resolver, on average, sent 45 bytes to the registry and re-

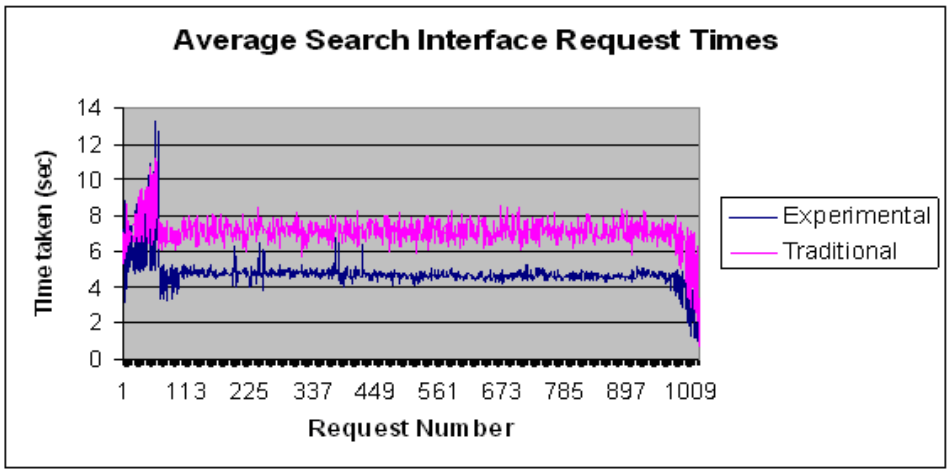

Fig. 4 Response times for 1024 search instance requests

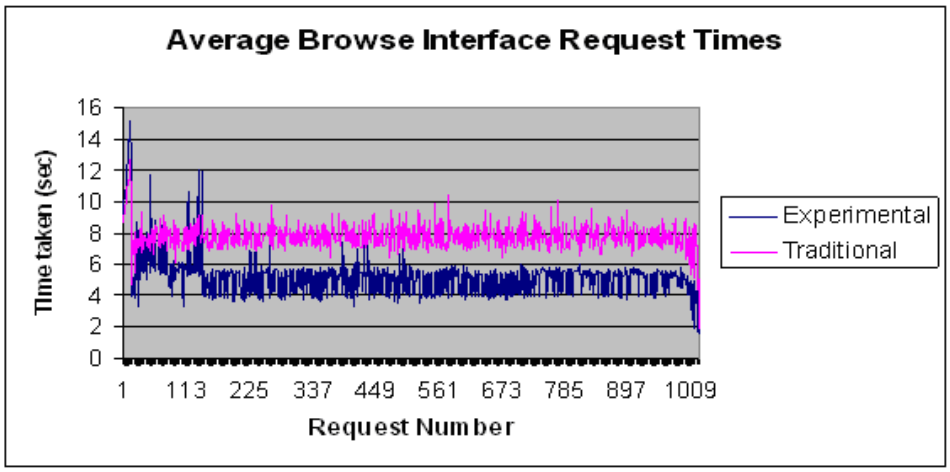

Fig. 5 Response times for 1024 browse instance requests

ceived a 393 byte response, taking an average of 0.031 seconds.

These tests were conducted on a single machine as remote connection overheads depend on specific of networking equipment and protocol stacks. Also, in many instances, the registry will never be contacted because of caching at the resolver.

The net result is that the overall time taken for inter-component communication to maintain additional information is minimal, so the extra housekeeping will not have a major impact on most DL systems based on clusters.

Migration and replication pose a greater concern as the amount of time taken may cause the system to perform at its worst when it is needed the most. The time to transfer instances was therefore measured and, of 86 recorded values, the average time taken to migrate an instance was 0.790 seconds, where the instance was 277kB in size on average. This reasonably low figure and the quick stabilisation of the performance graphs above indicate that migration and replication do not have a large impact where the size of instances is not large. This experiment did not consider transferring of databases - that may cause problems but the database is not usually moved around in a typical multi-tier application environment.

4.3 Summary of Results

In summary, these experiments have verified that, independently of a number of parameters that can be optimized, the component-based digital library system with minimal additional functionality is able to successfully make use of additional resources to scale seamlessly and with little overhead in most cases.

5 Future Work

This section aims to give an overview of possible future work with respect to component migration and replication and the potential benefits that can be gained from such techniques.

5.1 Load Balancing

The load balancing techniques employed in this research are not highly sophisticated but serve as a proof of concept for component migration and replication. The satisfactory results obtained thus far justify further study into more effective and generalisable load balancing schemes.

Future work into this area could focus on drawing together multiple metrics such as CPU usage time, I/O time, number of database accesses and number of components on the nodes. These metrics should provide more accurate information on system usage and should result in better load balancing decisions. A further extension that requires some investigation would be to perform inference over historical load-balancing data in order to deduce trends in system load and therefore be able to pre-empt component migration and replication. Lastly, database access and migration must be considered. A system that is capable of deciding when ideal conditions are met for entire database migration or replication could potentially enhance performance even further.

5.2 Security

Since this research was conducted by making use of a secure cluster, there currently exists no mechanism to ensure that instances are not tampered with during migration and replication. If one were to deploy such a component-based digital library system, where instances move from one distributed system to another, it may be necessary to ensure that the instance is secured during transit. Conventional encryption or hashing algorithms could be used for this purpose. Security at the component level will lead to a site-neutral framework, avoiding the existing differences in security policies across machines. In particular, the security approach provided by Globus could be adopted.

5.3 Cluster/Grid Hybrid

A natural next step is to consider grid middleware toolkits to provide scalability services, without incurring the overheads of existing heavyweight frameworks such as Globus. A campus grid, for example, based on Condor, provides an ideal experimental framework to incorporate both clusters and desktop machines into a single heterogeneous grid, with existing mechanisms for fault tolerance and resource management.

How digital libraries will interact with cluster and grid technology in the future is uncertain. Since core digital library components are critical to the operation of the system itself, grids by themselves cannot be fully relied upon to ensure the stability of the entire system. Future work in this field then could benefit from a study that contrasts the benefits of using a purely cluster-based approach to digital library component migration and replication with a purely grid-based approach, and then a combination of the these approaches. As alternative high performance computing approaches emerge, such as multi-core and cell processors, these also must be considered as alternatives in increasing the scalability of digital library systems.

5.4 Client-Side Computing

Most component-based systems assume that services are provided on servers - however it is possible to migrate some of the components to the client machines and have them execute in the user's browser. This approach to scalability ensures minimal processing on the server and scales the available processing resources with the number of users. With the emerging popularity of Ajax and similar client-side technologies, this approach to scaling of systems may yield substantial benefits and alter the balance of distributed processing once again.

6 Conclusions

Digital libraries have an ever more urgent need to scale to meet the demands of larger collections of data and greater service requirements. This need may be satisfied by a myriad of different solutions in disciplines ranging from databases to high performance computing. This article has discussed an attempt to create scalable digital library systems based on minimal extensions to an existing component framework for digital library systems. The complexity inherent in incorporating transparent scalability is certainly not a prohibitive factor, and for some services and some digital library architectures this may be a natural next step. Most importantly, this work has validated the importance of component architectures since the ability to scale efficiently and transparently relies on the flexibility of the underlying architecture. While this study has looked into minimal extensions for cluster-based DL system, the feasibility of such architectures extends naturally to more fully developed production systems and validates the approach envisaged by DILIGENT. As and when grid middleware matures and is widely adopted by DL developers, digital library systems based on component architectures can easily make the transition to utilize such lower level tools. Designers of monolithic systems will have to return to the drawing board.

References

1. Anderson, DP., Cobb, J., Korpela, E., Lebofsky M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM **45**, *11*, 56–61. ACM Press (2002)
2. Andresen, D., Yang, T., Egecioglu, O., Ibarra, O. H., Smith, T. R.: Scalability Issues for High Performance Digital Libraries on the World Wide Web. Technical Report, March 1996, Department of Computer Science, University of California Santa Barbara (1996)
3. Bar, M.: openMosix, a Linux Kernel Extension for Single System Image Clustering. In: Proceedings of Linux Kongress: 10th International Linux System Technology Conference, Saarbrücken, Germany, 15-16 October, 94–102. (2003)
4. Candela, L., Castelli, D., Pagano, P.: A Reference Architecture for Digital Library Systems. In: DELOS Conference on Digital Libraries. Grand Hotel Continental - Tirrenia, Pisa, Italy, 13-14 February. (2007)
5. Castelli, D., Pagano, P.: A system for building expandable digital libraries. In: Delcambre, L., Henry, G. (ed.) Third ACM/IEEE-CS Joint Conference on Digital Libraries, Houston, USA, 27-31 May, 335–345. IEEE Computer Society, Washington, DC, USA (2003)
6. Catalyst IT: Technical Evaluation of selected Open Source Repository Systems. Open Access Repositories in New Zealand (2006) https://eduforge.org/docman/view.php/131/1062/Repository%20Evaluation%20Document.pdf
7. Don, K. J., Bainbridge, D., Witten, I. H.: The design of Greenstone 3: An agent based dynamic digital library. Technical report, December 2002. Department of Computer Science, University of Waikato, Hamilton, New Zealand (2002) http://www.greenstone.org/manuals/gs3design.pdf
8. Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: Gaudiot, J., Ni, L. (ed.) IFIP International Conference on Network and Parallel Computing, Beijing, China, 30 November-3 December, LNCS 3779, 2–13. Springer-Verlag (2005)
9. Glasner, J.: Britannica Redux. Wired Magazine, March 1999. (1999) http://www.wired.com/techbiz/media/news/1999/11/32278
10. Haedstrom, M.: Research Challenges in Digital Archiving and Long-term Preservation. In: Proceedings NSF Post Digital Library Futures Workshop, Cape Cod, 15-17 June. (2003) http://www.sis.pitt.edu/ dlwkshop/paper_hedstrom.html
11. About OpenPBS. http://www.openpbs.org/about.html
12. IBM World Community Grid. http://www.worldcommunitygrid.org/about_us/viewAboutUs.do
13. Imafouo, A.: A Scalability Survey in IR and DL. TCDL Bulletin, **2**, *2*. http://www.ieee-tcdl.org/Bulletin/v2n2/imafouo/imafouo.html
14. Lagoze, C., Davis, J. R.: Dienst - An Architecture for Distributed Document Libraries. Communications of the ACM, **38**, *4*, 47. ACM Press (1995)
15. Lagoze, C., Van de Sompel, H.: The open archives initiative: building a low barrier interoperability framework. In: Borgman, C. (ed.) JCDL 2001, Roanoke, VA., USA, 17-23 June, 54–62. ACM Press, New York, NY (2001)
16. Lyman, P., Varian, H. R.: How Much Information 2003?. University of California (2003) http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/index.htm
17. MPI-2: Extensions to the Message-Passing Interface. MPI-1.2 and MPI-2 standards. http://www.mpi-forum.org/docs/mpi2-report.pdf
18. Naughton, T., Scott, S. L., Barrett, B., Squyres, J., Lumsdaine. A, Fang, Y., Mashayekhi, V.: Looking Inside the OSCAR Cluster Toolkit. In: Dell Powers Solutions (2002) http://www.csm.ornl.gov/PR/dell_1.html
19. Omar, M.: Component-based Digital Library Scalability using Clusters. MSc Thesis. University of Cape Town (2007)
20. Papadopoulos, P. M., Katz, M. J., Bruno, G.: NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters. In: 3rd IEEE International Conference on Cluster Computing (CLUSTER'01), Sutton Place Hotel, Newport Beach, California, USA, 8-11 October, 258. IEEE (2001)
21. Smith, M., Bass, M., McClellan, G., Tanlsey, R., Barton, M., Branchofsky, M., Stuve, D., Walker, J. H.: DSpace: An Open Source Dynamic Digital Repository, D-Lib Magazine, **9**, *1*, January (2003). http://www.dlib.org/dlib/january03/smith/01smith.html
22. Staples, T., Wayland, R., Payette, S.: The Fedora Project: An Open-source Digital Object Repository System. D-Lib Magazine, **9**, *4*, April (2003). http://www.dlib.org/dlib/april03/staples/04staples.html
23. Suleman, H., Fox, E. A.: A Framework for Building Open Digital Libraries. D-Lib Magazine, **7**, *12*, December (2001). http://www.dlib.org/dlib/december01/suleman/12suleman.html
24. Suleman, H.: Analysis and Evaluation of Service-Oriented Architectures for Digital Libraries. In: Turker, C., Agosti, M., Schek, H. (ed.) Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures, 6th Thematic Workshop of the EU Network of Excellence DELOS, Cagliari, Italy, 24-25 June 2004, Revised Selected Papers. Lecture Notes in Computer Science 3664, 130–146. Springer (2005) http://pubs.cs.uct.ac.za/archive/00000278/01/delos_2005_paper_eval_full_revised.pdf
25. Suleman, H.: Parallelising Harvesting. In: Sugimoto, S., Hunter, J., Rauber, A., Morishima, A. (ed.) Proceedings of 9th International Conference on Asian Digital Libraries (ICADL 2006), Tokyo, Japan, 27-30 November, 81–90. Springer-Verlag (2006). http://pubs.cs.uct.ac.za/archive/00000328/01/icadl_2006_oaihpc.pdf
26. Suleman, H., Fox, E. A.: Designing Protocols in Support of Digital Library Componentization. In Agosti, M., Thanos, C. (ed.) Proceedings of 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2002), LNCS 2458, Rome, Italy, 16-18 September, 568–582. Springer Berlin / Heidelberg (2002) http://www.husseinsspace.com/publications/ecdl_2002_paper_odl.pdf
27. Sunderam, VS.: PVM: a framework for parallel distributed computing. Concurrency: Practice and Experience, **2**, 315–339. (1990)
28. Su, A., Choffnes, D.R., Kuzmanovic, A., Bustamante, F.E.: Drafting behind Akamai (travelocity-based detouring). In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2006), Pisa, Italy, 11-15 September, 435–446, ACM (2006)
29. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor - A Distributed Job Scheduler. In: Sterling, T. (ed.) Beowulf Cluster Computing with Linux. The MIT Press (2002)
30. van de Sompel, H., Bekaert J., Liu, X.: aDORe: a modular and standards-based digital object repository at the los alamos national laboratory. In: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05), 367, ACM (2005)