A Hybrid Distributed Architecture for Indexing

Ndapandula Nakashole[1]* and Hussein Suleman[2]

[1] Max-Planck Institute for Computer Science
Saarbruecken, Germany
nnakasho@mpi-inf.mpg.de
[2] Department of Computer Science, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
hussein@cs.uct.ac.za

Abstract. This paper presents a hybrid scavenger grid as an underlying hardware architecture for search services within digital libraries. The hybrid scavenger grid consists of both dedicated servers and dynamic resources in the form of idle workstations to handle medium- to large-scale search engine workloads. The dedicated resources are expected to have reliable and predictable behaviour. The dynamic resources are used opportunistically without any guarantees of availability. Test results confirmed that indexing performance is directly related to the size of the hybrid grid and intranet networking does not play a major role. A system-efficiency and cost-effectiveness comparison of a grid and a multiprocessor machine showed that for workloads of modest to large sizes, the grid architecture delivers better throughput per unit cost than the multiprocessor, at a system efficiency that is comparable to that of the multiprocessor.

1 Introduction

Distributed architectures are de facto data scalability platforms as evidenced by the scale of data handled by service providers on the Web such as those that provide search and storage services. With ever-expanding digital library collections, scalable services are needed to provide efficient access to data.

In recent years, Web search engines have enabled users on the Web to efficiently search for documents of interest. Results are returned in a few seconds, with potentially relevant documents ranked ahead of irrelevant ones. These technology companies compete with one another to provide high quality search services requiring complex algorithms and vast computer resources, at no direct financial cost to the user. However Web search engine spiders often do not completely index data stored in digital libraries so search has to be provided as part of the digital library software suite.

Many large-scale Web service providers — such as Amazon, AOL, Google, Hotmail and Yahoo! — use large data centres, consisting of thousands of commodity computers to deal with their computational needs. In 2003, Google's

* Work done while a student at the University of Cape Town

search engine architecture had more than 15,000 commodity class PCs with fault-tolerant software [3]. The key advantage of this architectural approach is its ability to scale to large data collections and millions of user requests. For example, Web search engines respond to millions of queries per day at a low latency. Clusters of commodity computers are known for their better cost/performance ratio in comparison to high-end supercomputers. However, there is still a high cost involved in operating large data centres. Such data centres require investment in a large number of dedicated commodity PCs. In addition, they need adequate floor space, cooling and electrical supplies. IDC[3] reported that in 2007 businesses spent approximately $1.3 billion to cool and power spinning disk drives in corporate data centres and this spending is forecasted to reach $2 billion in 2009 [6].

For an organisation hosting a digital library, whose primary focus is not information retrieval, it may be difficult to justify expenditure on a data centre. In addition, if the computers will not be used for other tasks, they may not be highly utilised at all times. Furthermore, it is not clear how much more data centres can be scaled up at a reasonable cost if both data and workload continue to grow in the coming years. Baeza-Yates et al.[5] estimate that, given the current amount of Web data and the rate at which the Web is growing, Web search engines will need 1 million computers in 2010. It is therefore important to consider other approaches that can cope with current and future growth in data collections and be able to do so in a cost-effective manner.

This paper proposes an alternative architecture — a hybrid scavenger grid consisting of both dedicated servers and dynamic resources in the form of idle workstations to handle medium- to large-scale search engine workloads. The dedicated resources are expected to have reliable and predictable behaviour. The dynamic resources are used opportunistically without any guarantees of availability. These dynamic resources are a result of unused capacity of computers, networks and storage within organisations, exploiting work patterns of the people within an organisation. Dedicated nodes are needed to provide search services that are reliable and have a predictable uptime. Due to the limited number of dedicated nodes, they cannot provide the scalability required to handle indexing of large data collections. Thus the dedicated nodes should be augmented with the dynamic nodes that become available during non-working hours. From the dedicated nodes, the architecture gets reliability; from the dynamic nodes it gets scalability.

The rest of the paper is organised as follows: Section 2 discusses related work; Section 3 discusses the design and implementation details of the search engine; Sections 4 to 6 present evaluation details; and Section 7 provides concluding remarks.

[3] International Data Corporation (IDC) is a market research and analysis firm specialising in information technology, telecommunications and consumer technology. http://www.idc.com.

2 Related Work

Scavenger Grids. A "scavenger" or "cycle-scavenger" grid is a distributed computing environment made up of under-utilised computing resources in the form of desktop workstations, and in some cases even servers, that are present in most organisations. Cycle-scavenging provides a framework for exploiting these under-utilised resources, and in so doing providing the possibility of substantially increasing the efficiency of resource usage within an organisation. Global computing projects such as FightAIDS@Home [10] and SETI@Home [21] have already shown the potential of cycle-scavenging. The idea of a hybrid scavenger grid is an extension of cycle-scavenging — it adds the notion of dedicated and dynamic resources.

Hybrid Grid Architectures. Although there has been work done on information retrieval for cluster(for example Hadoop[14]), grid (for example Grid-Lucene[17]) and peer-to-peer(for example Minerva[18]) architectures, there has been virtually no published work that proposes the use of a hybrid scavenger grid for information retrieval. However a few works have investigated the use of hybrid scavenger grid architectures for other applications. A recent doctoral dissertation [1] investigated the use of a combination of dedicated and public resources in service hosting platforms. It observed that by designing appropriate resource management policies, the two types of resources can combined be utilised to increase the overall resource utilisation and throughput of the system. The BitTorrent[20] peer-to-peer video streaming platform relies on unused uplink bandwidth of end-user computers. Das et al.[7] have proposed the use of dedicated streaming servers along with BitTorrent, to provide streaming services with commercially valuable quality assurances while maintaining the scalability of the BitTorrent platform.

Enterprise Search Toolkits. A number of commercial systems dedicated to organisational search have been developed. FAST [9], OmniFind [19] and other traditional enterprise search engines are software toolkits. These toolkits do not mention the hardware infrastructure required to handle large scale intranet search. It is up to the organisation to determine the hardware infrastructure with the storage and computational power to deliver the desired scalability. The Google Search Appliance and the Google Mini Search Appliance [13] are hardware devices that provide intranet search able to handle up to millions of pages. These devices have initial acquisition costs as opposed to using resources already at the organisation's disposal in conjunction with a small number of dedicated resources.

3 Design and Architecture

The focus of this work is not on developing new information retrieval algorithms but rather on a different distributed architecture. Therefore, the developed prototype uses the Lucene [16] open source search engine as the underlying information retrieval engine. In addition, it uses the Condor job scheduler

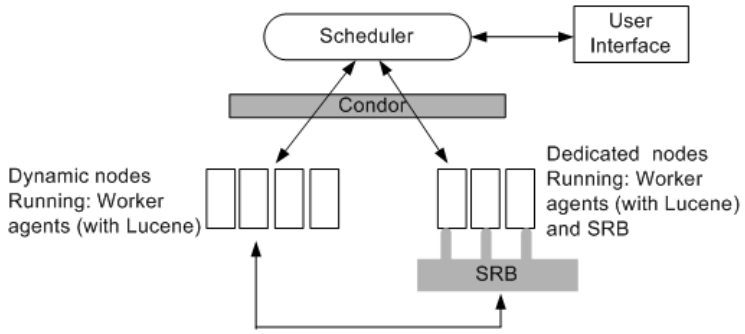

Fig. 1. High level components of the experimental search engine architecture. These are: User Interface, Scheduler, Condor, Worker Agents and SRB.

[15] for job submission and tracking. For distributed data management, the system employs the Storage Resource Broker (SRB)[4] data grid middleware which provides a layer of abstraction over data stored in various distributed storage resources, allowing uniform access to distributed data.

The architecture of the experimental search engine has five main components (see Fig. 1). The User Interface provides an access point through which queries enter the system. The Scheduler performs job allocation and uses Condor to distribute jobs to the dedicated and dynamic nodes which run the Worker Agents. Worker Agents refer to the software that executes on the nodes of the grid. The characteristics of the nodes dictate the types of jobs they can perform. The dedicated nodes are dedicated to search engine operations and thus as long as they are up and running they are available to execute jobs and provide services that are required for the search engine to operate. For this reason, the dedicated nodes are responsible for providing persistent storage for the indices via SRB and also for responding to queries. Because availability of the dynamic nodes cannot be guaranteed they only perform text indexing.

The Scheduler has the task of splitting the data collection into chunks and ingesting the chunks into SRB. The Scheduler also starts the Worker Agent software on the dedicated and dynamic nodes. It does this by first contacting Condor to get a list of the available nodes. The Scheduler then creates Condor jobs that instruct the nodes to run the Worker Agent software. Worker Agents request data to index. Upon receiving a request for a data chunk, the Scheduler allocates a new chunk to the requesting machine. The Scheduler specifies the data chunk allocated to the machine by indicating the location of the chunk on SRB. The Worker Agents run a Lucene indexer on the chunk and ingest the resulting sub-index on SRB. Once all the chunks are indexed, all the sub-indices located on a single SRB server are merged into a single index.

When a query is posed to the system via the User Interface, it is passed on to the Scheduler which routes the query to all the SRB storage servers that store indices. The SRB servers independently search their indices and return their re-

sults to the Scheduler. Finally, the Scheduler merges the results before returning them to the user. The next sections present the results of system performance evaluation.

4 Experimental Setup

4.1 Hardware

Four computing systems were used for the experiments. The first is a set of machines called the **Dynamic nodes** — they are 66 desktop machines within a 100 Mbps Ethernet network. Each machine is equipped with a 3 GHz Pentium 4 processor, 512 MB of RAM and a 40 GB hard disk. The second is a set of machines called **Dedicated nodes** which are 13 desktop class computers interconnected by a Gigabit Ethernet network. Each machine is equipped with a 3 GHz Pentium 4 processor, 512 MB of RAM and an 80 GB hard disk. The third is a desktop class computer with a 2.33 GHz Intel Core 2 Duo processor, 2 GB of RAM and a 250 GB SATA hard disk — this is the **Scheduler**. The fourth system is a **multi-core machine** (server) with a 3GHz Intel Quad-Core Xeon processor, 8 GB of RAM and a 500 GB SATA hard disk.

4.2 Data set and Query Set

The system was evaluated on a data collection crawled from the .ac.uk domain, which is the domain of academic institutions in the United Kingdom. The collection is 70.27 G of 825,547 documents. The collection has various file types(PDF 87,189; DOC 21,779; TXT 2,569; RTF 2,042 and HMTL 711,968). In order to test for data scalability, the collection was duplicated in cases where the data collection needed for the experiments is larger than the actual size of the collection. Query performance experiments did use duplicated data to simulate larger collections. The reason behind this is that duplicating the data collection only changes the index in one dimension. This can affect querying performance. It does not however affect indexing performance since in distributed indexing the data is indexed in small jobs and there are no duplicates within each indexing job. Each partial index is independent of subsequent partial indices and thus the index building process is not affected by data duplication.

Typical query logs from the domains crawled were not available. Instead, test queries used are top queries of the Web search volume made accessible via the Google Insights for Search service [12]. Google Insights for Search provides the most popular queries across specific regions, categories and time frames. The categories chosen are those that are typically covered by academic institutions, namely: Science, Sports, Telecommunications, Society, Health, Arts and Humanities. All the queries within the query set return a non-empty result set on the index of the AC.UK collection. The total number of queries in the set is 1008, with an average number of terms per query of 1.5 and the longest query contains 3 terms.

The first set of experiments, as shown in the next section, focused on how the dynamic nodes of the grid can be best organised and utilised to deliver the best indexing performance.

5 Varying Dynamic Indexers

Within the hybrid scavenger grid, the number of dynamic indexers plays a major role in indexing time. Ideally, as the number of dynamic indexers increases, indexing time decreases linearly. This experiment aimed to find the number of dynamic indexers that delivers the best performance. Best in this sense means that indexing time is reduced and also that the indexers involved are well utilised.

Distributed indexing can be organised in one of two ways. With `Local Data` distributed indexing, machines index data that is stored on their local disks and transfer the partial index to one of the index SRB storage servers. With `Non-local Data` distributed indexing, machines download source data that is stored on the storage servers on SRB and also store the resulting indices on SRB storage servers. Intuitively, the Local Data indexing approach achieves superior performance because of data locality and thus it incurs less network transfer time. The Local Data indexing approach was used in the experiments reported here.

Indexing performance for various data sizes was analysed. The accumulated indexing time is the total time spent on all tasks performed to index a collection of a given size, including job scheduling and communication. Indexing time is the time spent on actual indexing of data as opposed to other tasks such as file transfer or job scheduling. Communication and storage time is the time to transfer indices and to ingest the indices into SRB.

From Fig. 2 it is clear that as the number of dynamic nodes increases, indexing time decreases and that a large part of indexing time is spent on actual indexing. Communication and storage time for transmitting and storing indices on storage servers remains more or less the same even as the number of dynamic indexers increases. What has not been shown is how resource utilisation is affected as more dynamic nodes are added to the grid. Fig. 3 shows the grid system efficiency for varying numbers of dynamic nodes. System efficiency measures utilisation of resources — how busy the resources are kept.

Parallel system performance of a system consisting of n processors is often characterised using speedup — the serial execution time divided by parallel execution time: $speedup(n) = time(1)/time(n)$.

System efficiency is defined as the speedup divided by the number of processors:

$$\textit{system efficiency}(n) = speedup(n)/n$$

Thus efficiency is maximised at 1.0 when n=1. From Fig. 3, it can be seen that for the case of 32 GB, when more than 24 nodes are used, system efficiency goes down to below 50% of full efficiency. Therefore, at the 24 node point adding

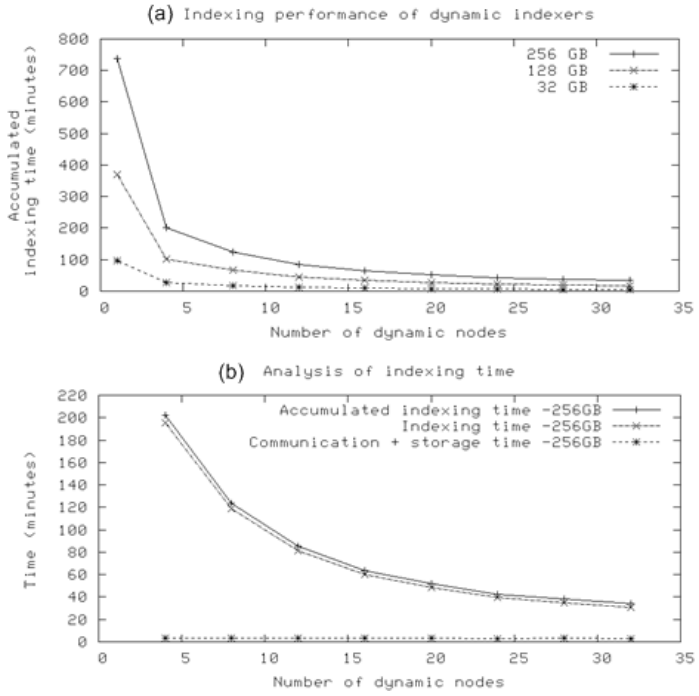

Fig. 2. Indexing performance for dynamic indexers, with 6 SRB storage servers.

more nodes decreases indexing time but utilisation per machine decreases to levels where each machine does little work, with for example each machine doing under 60 seconds of indexing. For the 128 GB and 256 GB cases, system efficiency also declines with increasing numbers of dynamic nodes. However, due to the increased workload the system remains relatively efficient, reaching a minimum of 67% and 68% efficiency respectively.

This experiment has shown that for a given workload, the number of dynamic nodes can be increased to index the collection in the shortest possible time. However, adding more nodes to the grid in order to achieve the shortest indexing time can result is poor utilisation of resources with system efficiency falling to levels below 50%. Therefore, the optimal number of dynamic nodes is the one that results in lowest indexing time at a system efficiency above a defined threshold.

The experiment reported thus far has shown indexing performance of the hybrid scavenger grid. The question to ask at this stage is how performance of the hybrid scavenger grid comparecompares to other architectures and whether it is worth investing in dedicated nodes and maintaining a grid, if the cost is the same as that of a middle or high end multi-processor server which has comparable performance.

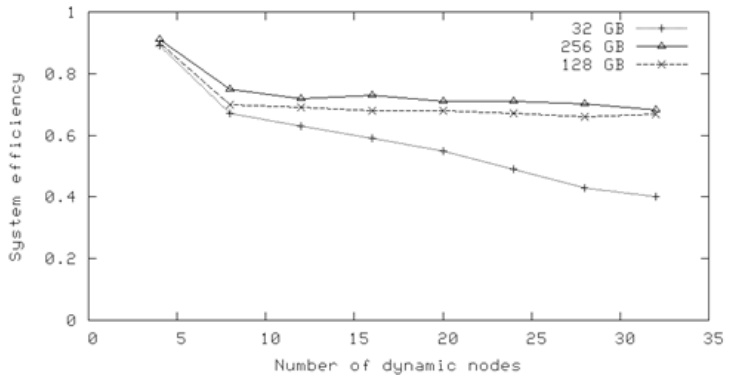

Fig. 3. System efficiency during indexing

6 Hybrid Scavenger Grid Versus Multi-core

While cost-effective scalability is one of the advantages of a hybrid scavenger grid-based search engine, the challenge is the process of designing, implementing and running a search engine on such a distributed system. Limitations of the grid such as the unpredictable nature of dynamic nodes and job failure rate can hinder performance. Thus it can be argued that with the advent of multi-core technology, powerful servers can be purchased for low prices and thus the centralised architecture should be the architecture for workloads of certain magnitudes.

Experiments were carried out to compare the cost-effectiveness and system efficiency of the quad-core machine to that of the hybrid scavenger grid.

To determine the cost-effectiveness of a system with n processors which cost $cost(n)$, performance and cost are combined to obtain cost/performance [22]:

$$costperf(n) = \frac{cost(n)}{1/time(n)}$$

A system is more cost-effective than the other when its costperf value is smaller than the other system's. The cost of a system depends on one's point of view. It can be hardware cost for processors, memory, I/O or power supplies. For the purpose of this experiment, the cost only includes processor cost. The prices used are list prices in US dollars (as of 7 December, 2008)[11]. The processor (Intel Quad-Core Xeon X5472/3 GHz) in the quad-core machine costs $1,022 and a typical desktop processor (Intel Core 2 Duo E8400/3 GHz)[4] costs $163.

[4] The experiments used Pentium 4 machines, however these are no longer listed in the price list from Intel — currently new desktop computers typically have an Intel Core 2 Duo processor and thus the price of a Core 2 Duo was used.

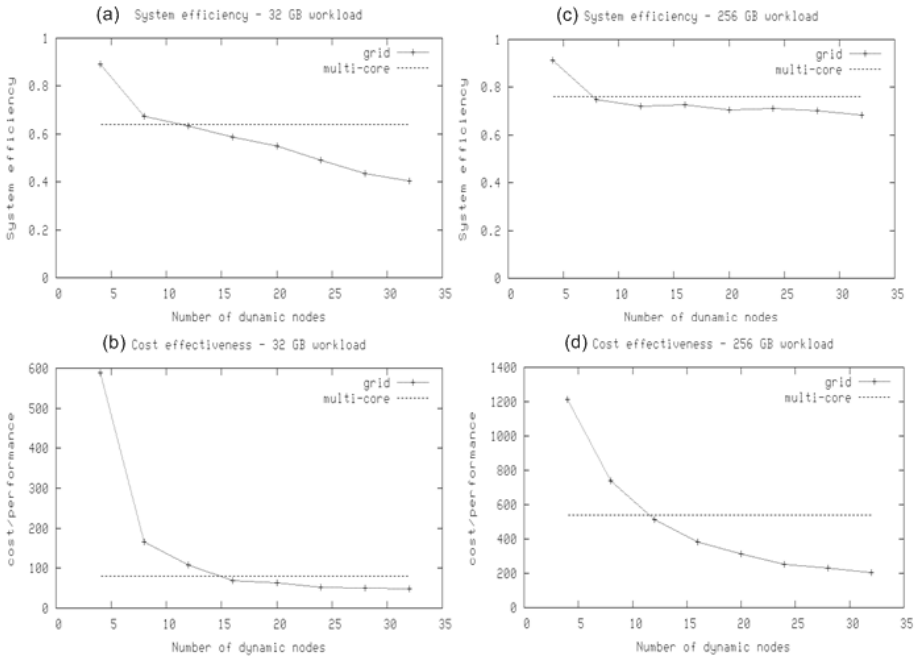

Fig. 4. System efficiency and cost-effectiveness: 32 GB and 256 GB

Fig. 4 (a) and (b) show system efficiency and cost-effectiveness of both systems, for the workload of 32 GB. The system efficiency of the multi-core is constant at 0.64 since the number of cores are fixed, whereas that of the grid varies with the number of dynamic nodes. It can be seen that for more than 12 dynamic nodes, the efficiency of the grid is lower than that of the multi-core, and continues to decline as more nodes are added. It can also be seen that the cost-effectiveness (Fig. 4 (b)) of the grid is only significantly better than the multi-core when 24 or more nodes are used. However, at this point the efficiency(Fig. 4 (a)) of the grid is 0.49 whereas that of the multi-core is 0.64. Therefore for this particular workload it can be concluded that multi-core is a better choice since employing the grid leads to poorly utilised resources.

Fig. 4 (c) and (d) show system efficiency and cost-effectiveness of both systems, for the workload of 256 GB. The system efficiency of the multi-core is 0.76. The efficiency of the grid is lower than that of the multi-core when more than 8 dynamic nodes are used — it remains relatively high and reaches a minimum of 0.68 for 32 dynamic nodes. It can be seen in Fig. 4 that the grid performs better and is more cost-effective when 12 or more dynamic nodes are used. At that point the grid has a system efficiency of 0.72 which is 4% less than that of the multi-core. For this workload, it can be concluded that the grid is more cost-effective and at the same time utilisation of the grid resources is relatively high.

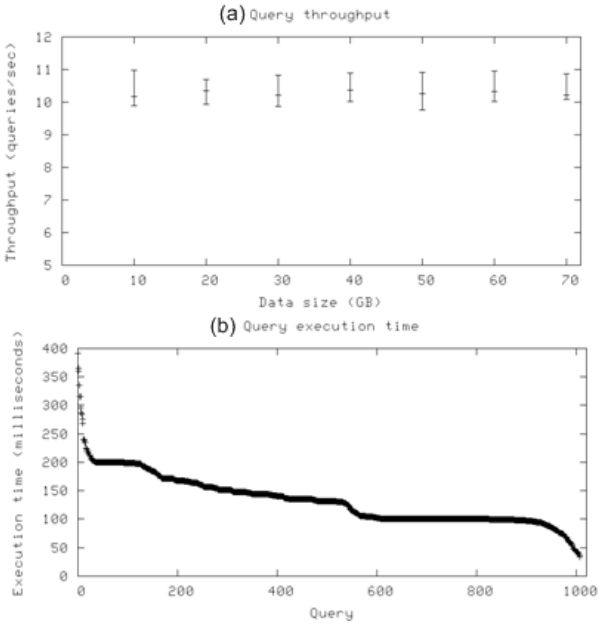

Fig. 5. Querying performance. The query response times in (b) are sorted in descending

This experiment has shown that for small workloads, although the grid provides better performance and cost-effectiveness for large numbers of dynamic nodes, the system efficiency goes to low levels that render the usefulness of the grid questionable. For modest to large workloads, the grid is a more beneficial approach achieving better cost-effectiveness and maintaining relatively high system utilisation.

Having established that the hybrid scavenger grid is a beneficial architecture for search engine indexing, it important to also evaluate its performance for searching.

7 Querying Performance Analysis

In a scalable search engine, query response time should remain more or less constant even as the size of the searched index increases. Moreover, the index distribution among the index storage servers should enable query response times to be more or less the same for different queries — the time to respond to individual queries should not be substantially longer for some queries while it is shorter for others.

From Fig. 5(a) it can be seen that the average query response time remains fairly constant even as the data size is increased. Query throughput is determined by the performance of the query servers and also by the arrival rate of

queries at the scheduler [2] . The attained average throughput is 10.27 queries per second. This means that the 6 storage servers used can process up to 36,972 queries per hour or roughly close to a million queries per day. With the average throughput of 10.27, the average time per query is 0.10 seconds. The query throughput attained is comparable to that of other distributed systems. Badue et al [2] reported that with a cluster of 8 machines, they observed a query throughput of 12 queries per second, with an average time per query of 0.12 seconds.

Fig. 5(b) shows that response times for all the queries is below one second, with an average query response time of 0.13 seconds, a minimum of 0.034 seconds and maximum of 0.39 seconds. This experiment has shown that the average query response time remains fairly constant, that query response times are below one second and that the variance in query response times is not substantial.

8 Conclusions

The hybrid scavenger grid proves to be a feasible architecture for a search engine that supports medium- to large-scale data collections within an intranet. The system reduces indexing time and responds to queries within sub-seconds. The resources of the system can be organised in a way that delivers the best performance by using the right number of nodes. The desired levels of performance and system efficiency determine the optimal number of dynamic/static nodes to index a collection.

The scalability of the architecture comes from the fact that more dynamic nodes can be added as required. Data scalability is vital as collections within digital libraries continue to grow at fast rates. For example, digital libraries of scholarly publications have become increasingly large as academic and research institutions adopt open access institutional repositories, using tools such as EPrints[8], in order to maximise their research impact. As an institution grows, so does the amount of the data it produces but also human resources increase. Assuming the normal practice of one computer per person, there will always be enough dynamic nodes to deal with the increasing data within an institution.

One possible future work direction is to evaluate the system's ease of maintenance. Maintaining a distributed system requires significant human effort. Large data collections of several terabytes of data require a large grid consisting of large numbers of dynamic nodes. As the size of the grid grows, the effort required to operate and maintain the grid also becomes greater. Therefore, it would also be of interest to know the human cost of a hybrid scavenger grid operation in comparison with the other architectures while taking into account performance, hardware cost-effectiveness and resource efficiency.

References

1. S. Asaduzzaman. Managing Opportunistic and Dedicated Resources in a Bi-modal Service Deployment Architecture. PhD thesis, McGill University, 2007.
2. C. Badue, P. Golgher, R. Barbosa, B. Ribeiro-Neto, and N. Ziviani. Distributed processing of conjunctive queries. In Heterogeneous and Distributed IR workshop at the 28th ACM SIGIR Salvador,Brazil, 2005.
3. L. A. Barroso, J. Dean, and U. Hölzle: Web search for a planet: The Google Cluster Architecture. IEEE Micro, 23(2):22-28, March/April 2003.
4. C. K. Baru, R. W. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research,Toronto, Canada, 1998.
5. R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In ICDE, pages 6-20, Istanbul, Turkey, 2007. IEEE.
6. Computerworld Inc. Storage power costs to approach $2B this year. Website, 2009. http://www.computerworld.com
7. S. Das, S. Tewari, and L. Kleinrock. The case for servers in a peer-to-peer world. In Proceedings of IEEE International Conference on Communications, Istanbul, Turkey, 2006.
8. EPrints. Open access and institutional repositories with EPrints . Website, 2009. http://www.eprints.org/.
9. FAST. FAST enterprise search. Website, 2008. http://www.fastsearch.com.
10. FightAIDS@Home. Fight AIDS at Home. Website, 2008. http://fightaidsathome.scripps.edu/.
11. Intel Cooporation. Intel processor pricing. Website, 2009. http://www.intc.com/priceList. cfm.
12. Google. The Google Insights for Search. Website, 2008. http://www.google.com/insights/search/.
13. Google. The Google search appliance. Website, 2008. http://www.google.com/enterprise/ index.html.
14. Hadoop. Apache Hadoop. Website, 2008. http://hadoop.apache.org/
15. M. Litzkow and M. Livny: Experience with the condor distributed batch system. In Proceedings of the IEEE Workshop on Experimental Distributed Systems,1990.
16. Lucene. Lucence search engine. Website, 2008. http://lucene.apache.org/.
17. E. Meij and M. Rijke. Deploying Lucene on the grid. In Open Source Information Retrieval Workshop at the 29th ACM Conference on Research and Development on Information Retrieval, Seattle, Washington, 2006.
18. S. Michel, P. Triantafillou and G. Weikum. MINERVA: a scalable efficient peer-to-peer search engine. In Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware. Grenoble, Greece, 2005.
19. OmniFind. OmniFind search engine. Website, 2008. http://www-306.ibm.com/software/ data/enterprise-search/omnifind-yahoo.
20. J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In 4th International Workshop on Peer-to-Peer Systems, pp. 205216, Ithaca, NY, 2005.
21. SETI@Home. Search for extraterrestrial intelligence at home. Website, 2007. http://setiathome.berkeley.edu/.
22. D. A. Wood and M. D. Hill. Cost-effective parallel computing. IEEE Computer, 28:69–72, 1995.