Imperial College of Science,
Technology and Medicine
(University of London)
Department of Computing

Calculation of PageRank Over a Peer-To-Peer Network

by

Matthew Cook

Submitted in partial fulfilment
of the requirements for the MSc
Degree in Computing Science of the
University of London and for the
Diploma of Imperial College of
Science, Technology and Medicine.

September 2004

Abstract

Modern computer networks contain enormous quantities of unstructured data, and providing efficient methods of identifying interesting documents continues to be one of the largest challenges such networks present. Internet search company Google have had the most prominent success in this arena, and their achievements are in part due to their development and use of the PageRank algorithm, which is a technique employed to improve the relevance of search results by analysing the structure of hyperlinks in the underlying web graph. Currently PageRank is calculated centrally on a monthly batch cycle. This report suggests that calculating document relevance ratings in this way has drawbacks in terms of scalability, overreliance on centralised coordination and latency of resource discovery and inclusion, and presents an alternative method that may overcome these problems with the use of a modified algorithm calculated over a next-generation peer-to-peer protocol such as Chord.

Thanks to Jeremy Bradley for his guidance and sound advice throughout the project, and to Douglas De Jager, Saar Miron and Lida Tsimonou for sharing their insights into the subject.

Dedicated to Rebekah, light of my life

Contents

Chapter 1

Introduction

1.1 Search

Efficient document search is recognised as one of the most valuable services that can be provided on computer networks, but as the quantity of information stored online continues to proliferate the ability to provide useful search results becomes ever more difficult. The most prominent success in this field has been that of Google, the search company founded by Sergei Brin and Larry Page. Google began as a research project in 1998, and it is known from the papers published by Page and Brin [1, 2, 3] prior to the formation of Google as a commercial venture that one of the key distinguishing features of their approach lies in their method of computationally generating a relevance rating for each document in their index. The technique, known as the PageRank algorithm, analyses the connectivity of the internet's underlying web graph, providing each document in the web graph with a ranking, known as its PageRank, derived from the inbound links it receives. The more inlinks a document receives, the higher its potential for relevance in a search and the higher its PageRank.

1.2 Peer-to-Peer Systems

Peer-to-peer systems have become prominent in recent years with the increased availability of high-bandwidth internet connections as a way of sharing resources in a distributed way without a requirement for centralised coordination and control. File-sharing networks such as Gnutella [4, 5] and Napster [6] have generated enormous amounts of publicity and raised serious concerns among media companies because of their potential to allow free distribution of copyright material. The rapid escalation of these concerns is an indication of the success of such networks in providing functionality users want. The key characteristic of a peer-to-peer

network is that members can act in the capacity of both client and server. The extent to which this happens varies between systems. The original Napster application allowed for sharing of files between peers, but maintained a centralised directory of files available within the network to facilitate searching, a fact which eventually led to it being shut down. By contrast, the Gnutella protocol conducts file-searching tasks using a peer-to-peer query flooding technique. This has the key advantage of removing the potential for failure of a single node to bring down the entire network, as well as greatly complicating legal actions against perceived breach-of-copyright offences, though it also attracts criticism for lacking scalability and wasting bandwidth. Following the successes of Napster and Gnutella, a number of research groups have produced work seeking to explore the possibilities of peer-to-peer networks and to rectify some of the shortcomings of the popular first generation of applications. Protocols such as Chord [7, 8, 9], Pastry [10] and CAN [11] present the opportunity to exploit peer-to-peer scalability and power in a range of new ways.

1.3 Report Objectives

This report will review the areas of document search and peer-to-peer networks and examine potential crossovers between the two topics. Section two consists of a brief discussion of the purpose of search engines and the tasks they have to complete in order to perform their function. Section three presents an overview of some peer-to-peer protocols in wide general use (Gnutella, BitTorrent and Freenet), and three research protocols that have been proposed to form the foundation of next-generation peer-to-peer applications (namely CAN, Pastry and Chord). Section four examines the PageRank algorithm in detail, reviewing the characteristics of the basic algorithm and a variety of proposed modifications to it, including proposals that facilitate its use in a distributed network. The process of implementing a distributed document ranking mechanism, incorporating topics such as application architecture, tools employed and problems encountered, is discussed in section five. Section six consists of a presentation of experimental results obtained from the distributed ranking application, providing evidence that the algorithm chosen produces results compatible with those of the traditional centralised PageRank calculation and that in addition to reaching this basic suitability threshold they also demonstrate benefits in the form of rapid convergence properties and ability to recalculate incrementally as documents are added to the repository. Section seven discusses areas for further development to enhance the application and extend its functionality. Section eight contains conclusions about the potential for ranking of documents in a distributed environment and suggests some possible applications.

Chapter 2

Search Engines

Search engines have the distinction of being perhaps the most broadly used of network applications, as well as being one of the most demanding in terms of resources required to generate satisfactory results. Even the most casual of internet users will be familiar with the idea of entering terms into a search engine such as Google or Yahoo with the expectation that a set of documents containing information in some way relevant to those terms will be returned. In 2003 over 55 billion searches were submitted to Google [12]. In order to respond to search requests effectively, an internet-wide search engine (as opposed to more narrowly-focused engines such as those targeting specific topics or corporate intranets) has to access and analyse a substantial proportion of the material currently stored online. Current estimates place this in the region of 4 billion documents, with a continuing high growth rate. Furthermore, for a search engine to retain user confidence it needs to maintain rapid query response times and a high level of relevance between the search terms entered and query results returned.

2.1 Motivation for Search Engines

There are reasons for users to use search engines, and reasons for commercial search engine providers to provide the search service, and the motivations of each party do not necessarily align perfectly. The user of a search service is seeking to extract information from the global repository of data that the internet constitutes; this pursuit of information is optimised for the user when the barest minimum of input (i.e. search terms) results in the richest (i.e. most relevant, whatever that means in the context of a specific user) output. The search provider is operating a business and needs to satisfy their customers in order for their business to continue to exist; comprehensive search services for a repository on the scale of the entire internet require substantial resources to operate, and yet the consumers

of search resources in the current environment would find the notion that they should pay for the service completely alien. The search provider's users are, in other words, not the same as the search provider's customers, and are not the only body that it needs to satisfy; the search engine customer is generally some sort of advertiser, although the form the advertising takes varies between providers. In some cases the search service can be seen as a loss-leader for other aspects of the provider's range; in other cases search is the core function. Providers face a dilemma between their relationship with their users and their relationship with their customers. If their users can consume the service without ever encountering information about their customers, their customers will have no reason to continue supporting the service with their revenue. On the other hand, if the users find the presence of communication from the customers invasive they will cease to use the service and its usefulness to the customer will cease to exist. The service provider has to rely partly on inertia among the users (i.e. that once people start using something they are inclined to stick with it unless they are presented with a compelling reason to change their minds, although the barriers to changing service are far lower for internet search than they are with, for instance, banking), and partly on providing a service that the user is satisfied with. Currently the main participants in the centralised search market are Google, Microsoft's MSN, Yahoo, AOL, Altavista and Ask Jeeves. Apart from Altavista, each of these providers now provides an explicitly labelled section on its results page indicating sponsored results, i.e. links to pages paid for by advertisers that the search engine has calculated are relevant to those results.

2.2 Search Engine Tasks

The Tasks a search engine performs can be summarised roughly as crawling, ranking, indexing and searching.

Crawling entails detecting and acquiring documents stored on servers attached to the internet. Techniques employed include iterative caching and search of documents pointed to by hyperlinks in downloaded documents. A basic methodology for this is to use depth-first searching, whereby a page is parsed for links, each of those pages are themselves parsed for links, and so on. Key problems to be addressed in implementations of this task are deciding how to represent and store the vast quantities of data that can be retrieved from a webcrawl, and ensuring that crawl queries are submitted in a fashion that shows due consideration for the resources of other network users. Search engines should not act as perpetrators of denial-of-service attacks.

Indexing entails generating an inverted index of terms included in all of the crawled documents. An inverted index consists of a table-like structure where the key is the search term, and the value attached to the key is an array of locations within the crawled documents that contain the term. Other tasks under this heading include stop-word processing (i.e. filtering out commonly-occurring words such as 'the' and 'and' that add no value to search results), and implementing techniques for returning multi-word query responses.

Ranking entails assigning a relevance rating to a document. For any given set of search terms, the search engine will need to make a decision as to what order to display the results of this search in. This decision is crucial; in a body of data as large as today's internet, the number of documents containing even intuitively obscure search terms can be far larger than is practical for a human reader to examine. Even diligent users seldom have the patience to enquire beyond the first 100 results returned by the search engine. Therefore, a methodology for placing results the user is more likely to want nearer the top of the hit list is desirable, and this is the area in which Google is most famous for its innovation. The PageRank algorithm is a patented technique employed by Page and Brin which was instrumental in differentiating Google from rival search engines that gave precedence to search results based on advertiser sponsorship or manual selection [2, 1, 3]. It is noted in [3] that the PageRank measure is particularly effective at providing a relevant result when the search terms provided are minimal; a carefully formulated query is likely to have fewer hits and therefore be more dependent on the breadth of crawled material and the analysis performed on it. An intuitive justification for PageRank is that a document conveys a measure of authority to another document by containing a link to it, and that when the community of documents contained in the internet as a whole are considered in aggregate, the link structure between them can be used as a barometer of the authority of an individual page in the eyes of its peers. The PageRank score conferred from one page to another is a function of the PageRank score belonging to the donor page, and the number of outlinks contained on the page; that is to say, the PageRank score generated by receiving a link from a popular page will be higher than that generated from a link from an unpopular page, and the score generated by a link from a page with many outlinks will be less than that generated by a link from a page with the same popularity but fewer outlinks. The details of the calculation required to arrive at a PageRank value will be discussed further in chapter four.

Searching is the process of receiving a query from a user, parsing the query into a form the backend data storage can use, generating a set of results and returning them to the user. A search service succeeds or fails based on how favourably users view its output; it has to be both relevant and fast. This report will not focus on user interface response times, suffice to say careful design is required to complete the tasks involved in the few seconds users generally expect. The ranking calculation is one of the key inputs in assuring a high relevance rating, although commercial search engines use other inputs as well, such as user location. Achieving an accurate ranking number will be the search-related topic of prime concern in this report.

2.3 Problems With Search Engines

The approach to provision of search services followed by Google has clearly been successful. However, it exhibits some properties which may be considered problematic.

Firstly, the calculation of the PageRank value of each document is conducted in a centralised way, using a large cluster of servers operated by Google. While there will clearly be redundancy between individual servers within the cluster, there is still an inherent risk that the service has a single point of failure (or more likely a small number of points of failure; the exact configuration of Google's server farm is unknown, but it is highly likely they have more than one physical location) in the event that the cluster is rendered inoperational by a denial-of-service attack or catastrophic event - the sort of event the internet was designed to withstand. Those inclined towards paranoia may also consider the centralised search model problematic because of the power it conveys to the to body granting scores to documents and the potential abuses that could be made of that power, such as inflating ratings for commercial reasons or suppressing pages for political reasons [13]. Google state that they do not sell PageRank, but as the algorithms they use are certain to have been modified and refined in the years since they became a commercial operation, and any of these changes will be proprietary and not open to public scrutiny, there is no way of knowing exactly how it is now derived; even if the techniques used are entirely 'fair' now, there is no way of being sure it will always remain so. It would therefore be desirable to dissipate the ability to rank pages among the wider community, minimising the capability of any individual body to manipulate scores and safeguarding access to untainted sources of information.

Secondly, the centralised allocation of PageRank is recalculated by Google on a monthly cycle, as generating values for 4 billion pages takes several days. This means that assigning scores to pages is not very responsive to dynamic content,

thereby depriving users from potentially valuable results. It would be advantageous for the mechanism that assigns scores to documents to be able to cope with incremental addition of documents, assigning scores to them as and when they became available. This would be of benefit, for example, in integrating current news items into mainstream search results (contingent, of course, on the documents having been captured by the crawling service first). Currently news information is processed via a separate service, which appears to be a keyword search on particular news publishers; a near-realtime ranking calculation would allow news items to attract citations from individuals and other sources, thereby providing a mechanism for assessing general interest in particular pieces of commentary. Conversely, some parts of the web graph will remain stable from month to month with no change in link structure between documents, and yet the centralised calculation still consumes processing resources recalculating their PageRank.

Finally, there are concerns about the sustainability of generating a centralised calculation of document rank for an internet that shows no signs of slowing growth. An account of seemingly anomalous search results being returned from Google in the summer of 2003 suggests this was a consequence of software update required in Google's indexing system to modify the document identifier from four bytes to five bytes [14]. While the facts of this matter are unsubstantiated, the potential problems can be envisaged easily enough. For a repository of billions of documents, a change in identifier length by a small number of bytes can translate into an increased storage requirement of hundreds of terabytes. The decision on when to introduce this change would need to balance the amount of time the change would buy until storage became a problem again against the expense of buying more storage than necessary at current prices. Changes of this sort in large-scale applications are not trivial to implement, and the likelihood of difficulties arising can only increase with the frequency they have to be made. While it is clear that many of the people who are most competent at solving this sort of problem work for Google, it is certainly worthwhile considering whether the problem would exist at all were the calculation to be conducted over a peer-to-peer network, where storage capacity, bandwidth and processing power should increase in tandem with demand.

Chapter 3

Peer-to-Peer Systems

Peer-to-peer systems have come to prominence in recent years as a mechanism for sharing information with a looser requirement for central control than traditional client-server models. There is no hard-and-fast definition of what it is to be a peer-to-peer system, but characteristic features are as follows:

- Participating nodes can act both in the capacity of client and server, i.e. they can both request services from and supply services to other participants in the network. The idea is that peers can contribute resources as well as consume them, and that the capacity of the network as a whole to supply resources should increase proportionally with the demand to acquire them.
- Centralised control is completely absent, or minimised to a high degree.
- The network adapts gracefully to participants joining and leaving the network.
- The network may support 'ultrapeers', a category of peer considered to have high network capacity and/or availability, that can be used as a member of a set of well-known peers that facilitate admission to the network.

There are a variety of different approaches to implementing peer-to-peer networks in common use, and as proposals generated from research. A few of these are outlined below.

3.1 File-sharing Networks

Gnutella, BitTorrent and Freenet are three popular techniques for establishing peer-to-peer networks for the purpose of sharing files.

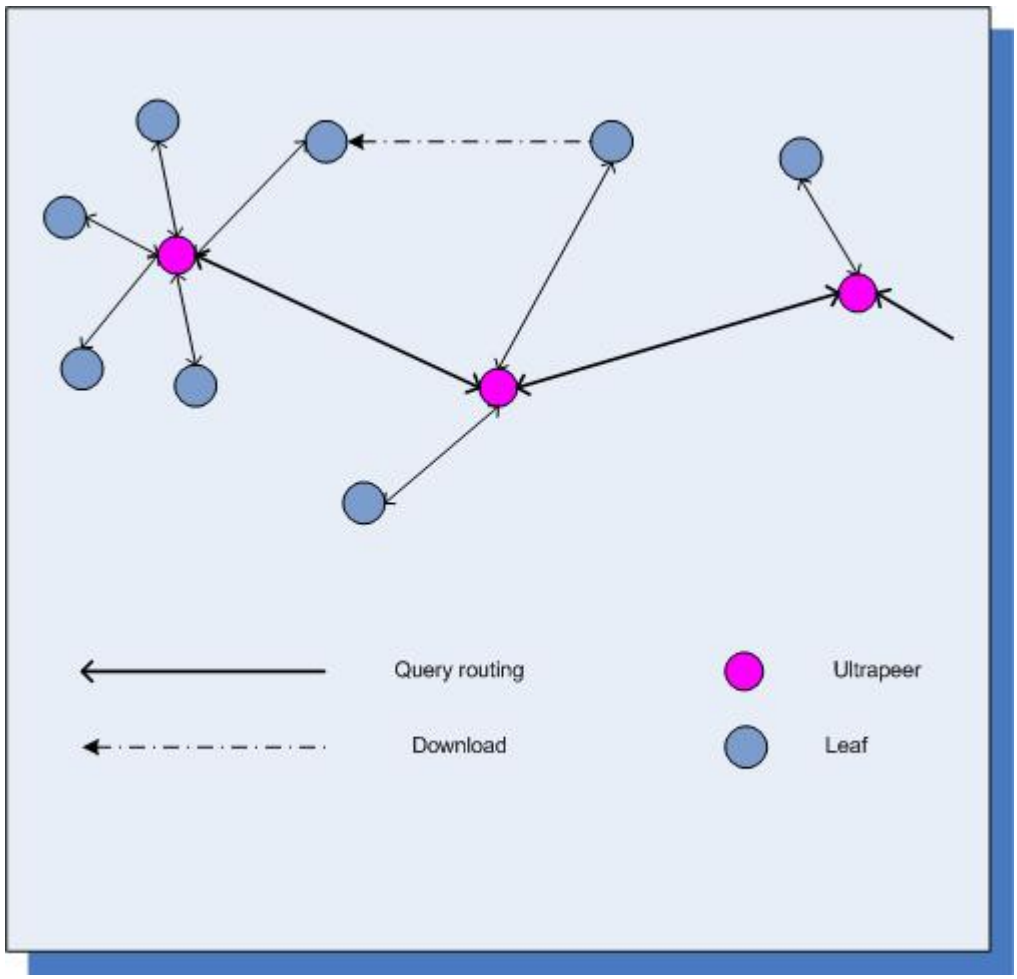


Figure 3.1: Illustration of communication flows within a Gnutella network

3.1.1 Gnutella

The first version of Gnutella was released in 2000. It is an open protocol that has continued to evolve from discussion within the developer community (the Gnutella Developers Forum), and a wide variety of clients can access the network by complying with the protocol. The principle it uses is that a peer only needs to be aware of a small number of neighbouring peers. When connecting to the network, the peer attempts to contact a small number of addresses (perhaps supplied with the client, retained from previous sessions, or stored in an online cache of well-known addresses), with a strong expectation that at least one of these peers will be currently connected to the network. Gnutella distinguishes between network participants focusing on consumption and supply of data (regular peers), and those concentrating on handling queries (ultrapeers). This distinction has been shown to improve overall network performance by allocating the function of routing queries to those participants best able to handle them. Once a peer has received a response to its query, it is then able to negotiate exchange of the document concerned directly with the document source via a HTTP connection. The Gnutella protocol does not support the concept of a PageRank-like relevance measure; it presents query matches to the client as and when they are received from responding peers. This is in keeping with its primary use as a means of exchanging media files, where the content sought is likely to be a specific set of documents rather than textual information from a general field. Perhaps the closest it comes is in the support of hashing to identify whether document content is as expected. However, there is no rating to be sorted with a document hash - it either agrees or it does not. Gnutella's query-routing technique (known as flooding) has faced criticism as being wasteful of bandwidth. While the introduction of the principle of ultrapeers, and various other refinements has certainly cut down the amount of unsuccessful queries floating around the network, it is fair to say that a query for a resource is at best a speculative venture; one can never be certain if there is definitely no match in the system or whether a match is present but is still en route.

3.1.2 BitTorrent

BitTorrent specialises in optimising the performance of large file downloads. It works on the theory that particular files will be popular at a particular time, and that any party seeking to acquire a resource should contribute to satisfaction of general demand for that resource by donating its own upload bandwidth while it takes download bandwidth from its peers. It differs from Gnutella in several respects

Firstly, the BitTorrent protocol itself does not support search at all. Resource

discovery is left to the user to establish by external means; online torrent directories exist for this purpose.

Secondly, the user is not permitted to download content without simultaneously uploading, thereby avoiding the problem of users 'leeching' on the network.

Files distributed via torrent are broken into segments, each of which can be exchanged individually, and the content of which is verified using hash digests. In this way, 'swarm' downloads can be used whereby different segments of the file are acquired from different sources concurrently to maximise download efficiency whilst retaining confidence that the contents of the file have not been altered.

Participants in a BitTorrent network are known as trackers, downloaders and seeds. The function of a **tracker** is to identify peers to each other. To initiate a torrent connection, a peer first contacts a tracker at a URL specified in the .torrent file; the tracker responds with a list of peers currently downloading the same file (**downloaders**). For a torrent to succeed, at least one of the downloaders, referred to as a **seed**, must be in possession of the complete file. Once the tracker has supplied the joining peer with a list of downloaders, any further negotiation of download terms is negotiated directly between peers. A variety of techniques are used to minimise unnecessary network traffic, maximise bandwidth use and counteract the possibility of particular file segments becoming 'rare' and thereby slowing down download completion [15].

3.1.3 Freenet

The key attribute that distinguishes Freenet from other content-sharing systems is its support for anonymity between users. This makes it valuable for communication which is potentially sensitive to censorship, for example in countries with oppressive regimes. Files uploaded to the network are stored in an encrypted form and replicated throughout the network. Each Freenet member designates an area of their own local storage to contribute to the network, and has no control over what content the network decides to store on their machine. The encryption makes it difficult (although not impossible) for the user to know what the content they are storing actually is, while the replication process makes it extremely difficult for a third party to determine whether a particular node is the source of content or merely acting as a transit point between source and target (in theory this could be defeated by traffic analysis, but doing so is thought to be prohibitively difficult). The objective of this is to make it tough to identify any individual as being responsible for a specific document. A further benefit of the replication used within Freenet is that popular files are dispersed among more nodes, thereby limiting the potential for individual nodes to be swamped by requests, whether legitimate or malicious. Freenet does not have any built-in search mechanism; resource discovery is achieved via Freenet-hosted discussion boards and directory

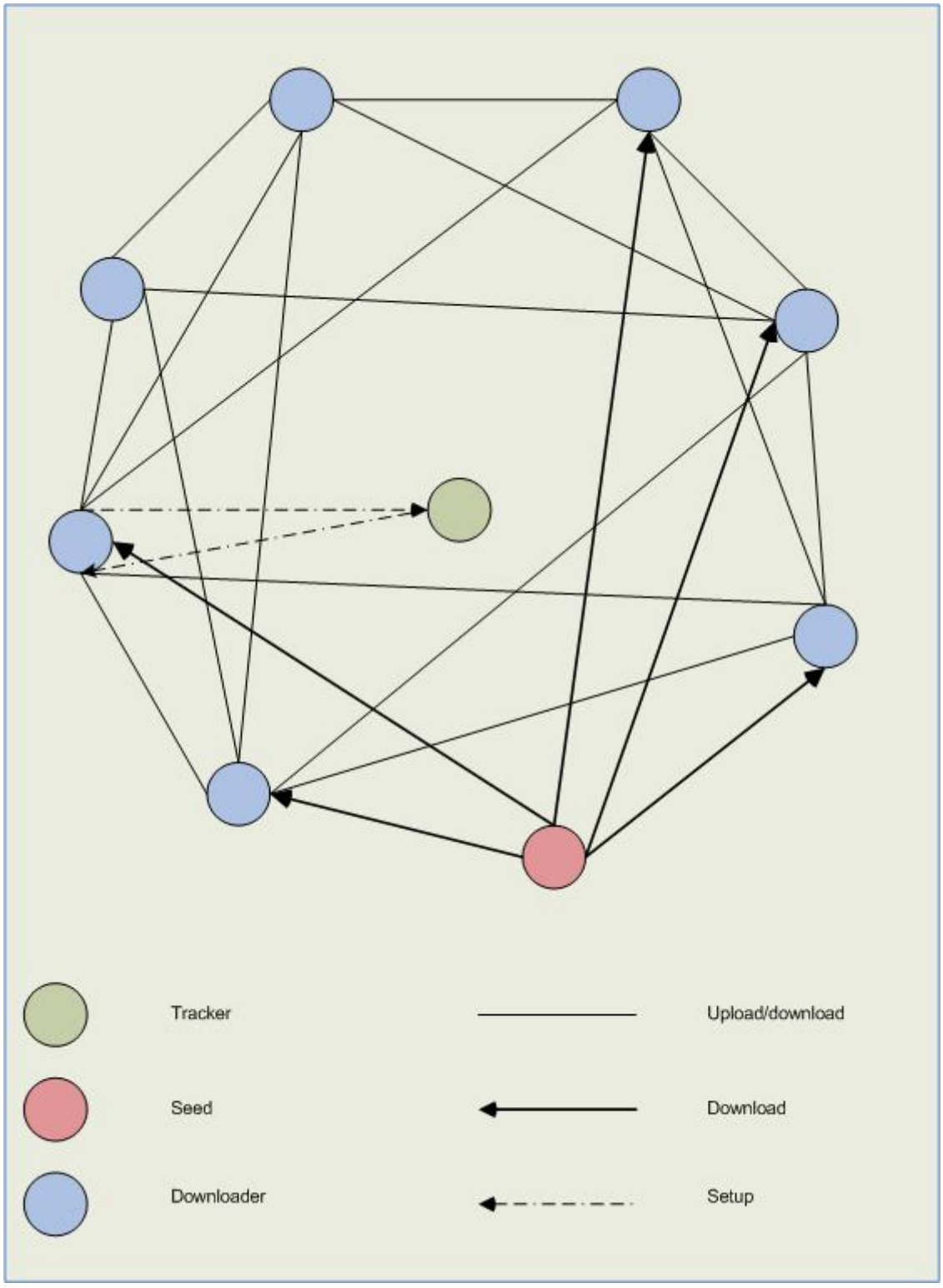


Figure 3.2: A BitTorrent network

sites [16].

3.2 Distributed Hash Tables

Distributed Hash Tables are a technique developed to store data over peer-to-peer networks. They can be seen as an attempt to construct the foundations of a new generation of peer-to-peer system, building on the lessons learned from the trial-and-error evolution of applications such as those already discussed and underpinning their architecture with rigourous scientific research. They are intended to be used as a primitive layer providing a lookup service to a variety of applications. Their function is analogous to that of a standard hash table, whereby data is stored in key-value pairs, but instead of the table being stored locally on one machine, its contents are spread around the members of the DHT network. Thus, to lookup the value stored at a particular key in a DHT it is necessary first to ascertain which node is responsible for that key, and then to perform a standard hash table lookup at that node. The key used for the DHT is typically a long bit sequence produced by applying an encryption algorithm, such as MD5 (128-bit) or SHA-1 (160-bit), to the content to be stored as the value in the table.

Several proposals for techniques to implement DHTs have been put forward (e.g. CAN, Pastry, Chord), each using different approaches, but attempting to achieve essentially the same thing. Typical characteristics of DHT protocols are as follows:

- The network provides reliable support for nodes joining and leaving. Real-world systems, especially domestic internet connections, exhibit unpredictable behaviour when connecting to other systems. They may not remain connected at all times, and may lose connection without warning. For a peer-to-peer system to be robust it needs to accommodate these events gracefully; the systems discussed below manage this via a combination of passing keys from neighbouring nodes to newly joining nodes, and caching keys in neighbouring nodes. The restriction of these maintenance functions to ‘close’ peers helps to reduce the network overhead involved.

- The protocol provides operations allowing for storage and retrieval of data in the DHT. Applications are able to use the storage/retrieval facility of the network as a foundation layer for further services.

- Designs seek to minimise the number of hops lookup messages need to take when seeking the location of a key held in the table, in order to minimise latency and unnecessary traffic.

- The encryption algorithm chosen generally ensures an even distribution of keys among peers, promoting an even distribution of workload in hosting of values and in processing of network maintenance tasks.

- To further improve load-balancing, and to compensate for the varying capabilities of individual machines in any peer-to-peer network, it should be possible to operate multiple logical nodes on a single physical node. Thus machines with higher processing power and/or network bandwidth should be able to operate more nodes (and thus handle more traffic) than those with lower specifications.

Potential applications for DHTs include the following:

- File-sharing tasks such as those carried out by BitTorrent and Gnutella [17].

- Distributed database applications [18]. These are suggested as useful solutions for cases where centralised database ownership is undesirable for reasons such as administration cost, or where the data to be queried resides most naturally in a distributed environment, such as network monitoring applications. DHTs provide a foundation that supports the routing of queries and collation of results that this sort of application requires. It should be noted that databases of this kind are not suitable for applications where completeness of information is important, due to the inherent fluctuation in membership and potential for node failure to affect results. DNS is suggested as one type of application that could be usefully migrated onto DHT-based database, as such a move would introduce greater flexibility in naming structure and would remove the need to use specialised hardware to provide the service.

- As the basis for a distributed file system [8].

- Application-level multicasting (as opposed to to IP-level multicasting) to support video-conferencing and TV distribution [11].

3.2.1 CAN

CAN (Content-Addressable Network) uses an address space that can be pictured as a d-dimensional toroidal Cartesian coordinate system (i.e. coordinates wrap around). The coordinates are purely a logical construction and bear no relationship to the physical placement of nodes. Keys are mapped onto this coordinate system using a deterministic hashing function. Two nodes are defined as neighbours if their coordinates overlap in $d - 1$ dimensions and abut on one dimension. Nodes are responsible for keys held within their region of the key space, and manage

routing of queries for keys they are not responsible for by consulting a list of neighbours and choosing the one nearest to the key value. Nodes maintain a list of $2d$ neighbours and their IP addresses for this purpose. By default, routing of queries will follow the shortest Cartesian path through the coordinate space, but in the event of a node failure they can be routed around the failed node. Contingency methods exist to use multicasts in the event of a widespread failure rendering all neighbours inoperative at once. The joining process involves identifying an existing node, being notified of a region the joining node is to be responsible for, and advising neighbours of the arrival of the new node and its IP address. The coverage region is determined by splitting an existing zone in two along one dimension and allocating half of it to the existing node and half to the new arrival. Maintenance algorithms ensure any disruption caused to the arrangement of zones by node departure is repaired promptly [11].

In the basic CAN specification, nodes periodically send messages to their neighbours to confirm their continuing presence in the network; if such messages are not forthcoming for a specified interval, the neighbours will assume the node has failed and will assume responsibility for its zone. If a node leaves for reasons other than failure, it is able to explicitly inform its neighbours of this, allowing them to begin the takeover of its zone immediately and also to assume responsibility for the key-value pairs it stores. Failure of a node results in the loss of these values. One of the proposals for enhancing the robustness of data retention is to use multiple hash functions, thereby allowing for the storage of the same value at multiple (physically unrelated) key locations. This operation obviously brings overhead in terms of network messaging and storage capacity. Other techniques suggested include allocating multiple nodes to be responsible for the same zone, and postponing any zone-splitting operations until a specified threshold of nodes-per-zone has been reached, and having multiple coordinate systems (referred to as realities) within the network, with every value being assigned a key in each reality.

To address the issue of latency, CAN is able to leverage the concept of dimensions in its Cartesian space. The higher the number of dimensions, the higher the number of neighbours, and therefore the fewer the amount of hops a request will have to make to locate the node responsible for a given key. This is important because as the zones are assigned on a purely logical basis, a single logical hop between zones could actually involve several physical IP hops. Other optimisations proposed are for nodes to cache data they have retrieved recently, in the event they receive a request for it again, and for nodes that receive a large number of requests for specific popular keys to push that data out to all of their neighbours.

3.2.2 Pastry

Pastry is an alternative approach to implementing a DHT, with similar objectives to CAN, i.e. scalability, self-maintenance, minimised latency and fault-tolerance. Again, it uses the concept of nodes within the network being responsible for specific keys. Pastry's distinguishing features are the in finer points of how the relationships between nodes are managed, and in the fact that it has the capability to incorporate physical network distance metrics into its routing tables. Nodes joining a network are assigned an ID (using a semi-random technique such as generating a hash of their IP address) which places them at a point in a circular space covering the numerical range $[0 - 2^{128})$. Node IDs and keys are treated as digits with base 2^b, where b is a system parameter typically set to 4. Messages are routed to the node with ID numerically closest to the specified key. The data structures used to achieve this routing are as follows:

- The routing table is used in routing messages to the appropriate node. It is a table of $\log_{2^b} N$ rows (where N is the number of nodes in the network), each containing $2^b - 1$ entries. Each row n of the table contains entries that refer to node ID/IP address pairs where the first $n - 1$ digits of the node ID are the same as those if the local node itself. It is possible for entries on a row to remain empty if a node with a the required address cannot be found. Where there is an excess of candidate nodes available, a distance metric is used to select nodes that are closer in terms of network topology. This ability to bias routing in favour of nodes likely to be more responsive is distinctive to Pastry.

- The neighbourhood set contains addresses of nodes that are closest in terms of the chosen distance metric, as opposed to the numerical node ID. It is used in favouring selection of near nodes, rather than in routing messages.

- The leaf set is a set of nodes with numerically closest node IDs to that of the local node, and is used in message routing. The leaf set is consulted first when deciding how to route a message, and if one of the entries in it is determined to be numerically nearest to the key in question, that will be the final destination of the message. If the leaf set does not return a match, the routing table is consulted. An entry from the row with the highest possible number of leading digits the same as the key is chosen, such that the entry is the nearest to the local node.

The process for nodes joining a network is similar to that described for CAN. The entering node advises an arbitrarily chosen node of its intention to join. The existing node issues a query to determine which node is numerically closest to the joining node. Every node encountered on the route from the queried node to the

closest node supplies the contents of its neighbourhood set, leaf set and routing table to the joining node. The joining node uses this data to populate its own tables, and then uses this information to notify any nodes that need to know of its arrival.

Maintenance of network state information is performed by cross-referencing the local node's status tables with those of its peers. A leaf set entry that cannot be contacted is assumed to have failed; the failed entry is repaired by requesting the leaf table from the nearest live leaf table entry and taking the appropriate value from there. Failure of routing table entries is recovered from firstly by consulting entries on the same row of the table as the failed entry, and progressively extending the search until a match is eventually found. Neighbourhood set entries are queried periodically and failed entries are repaired by querying live neighbourhood set members for their own neighbourhood set.

Pastry does not explicitly mandate any techniques for replication of values within the DHT; it is left to applications that use Pastry as a supporting layer to organise such matters in whatever way they see fit.

3.2.3 Chord

Chord is a DHT protocol designed by a team from MIT. If focuses on providing the service of mapping keys to nodes in a network with dynamic membership, leaving other considerations such as replication of data to client applications. The method it uses for providing this service is easier to visualise than those employed by CAN and Pastry, is provably scalable, and delivers robust message routing even under rapid change of network population. The Chord protocol operates as follows:

- Nodes are assigned an identifier using a n-bit identifier using a hashing function, for example the SHA-1 hash of the node's IP address. This places the node at a point in a circular space with 2^n points.
- Keys are assigned to the first node in the identifier space that has in identifier equal to or higher than its own (n.b. 'higher than' in this context is taken modulus 2^n). This is known as the *successor node*. Every node stores details of its own successor.
- When a node leaves the network responsibility for its assigned keys is passed to its successor. Changes to network state information are confined to a local area, i.e. only those nodes that know about the state of the departing node.
- To facilitate faster routing of messages to the correct destination, each node maintains a *finger table*, which is a table of at most *n* entries, where the

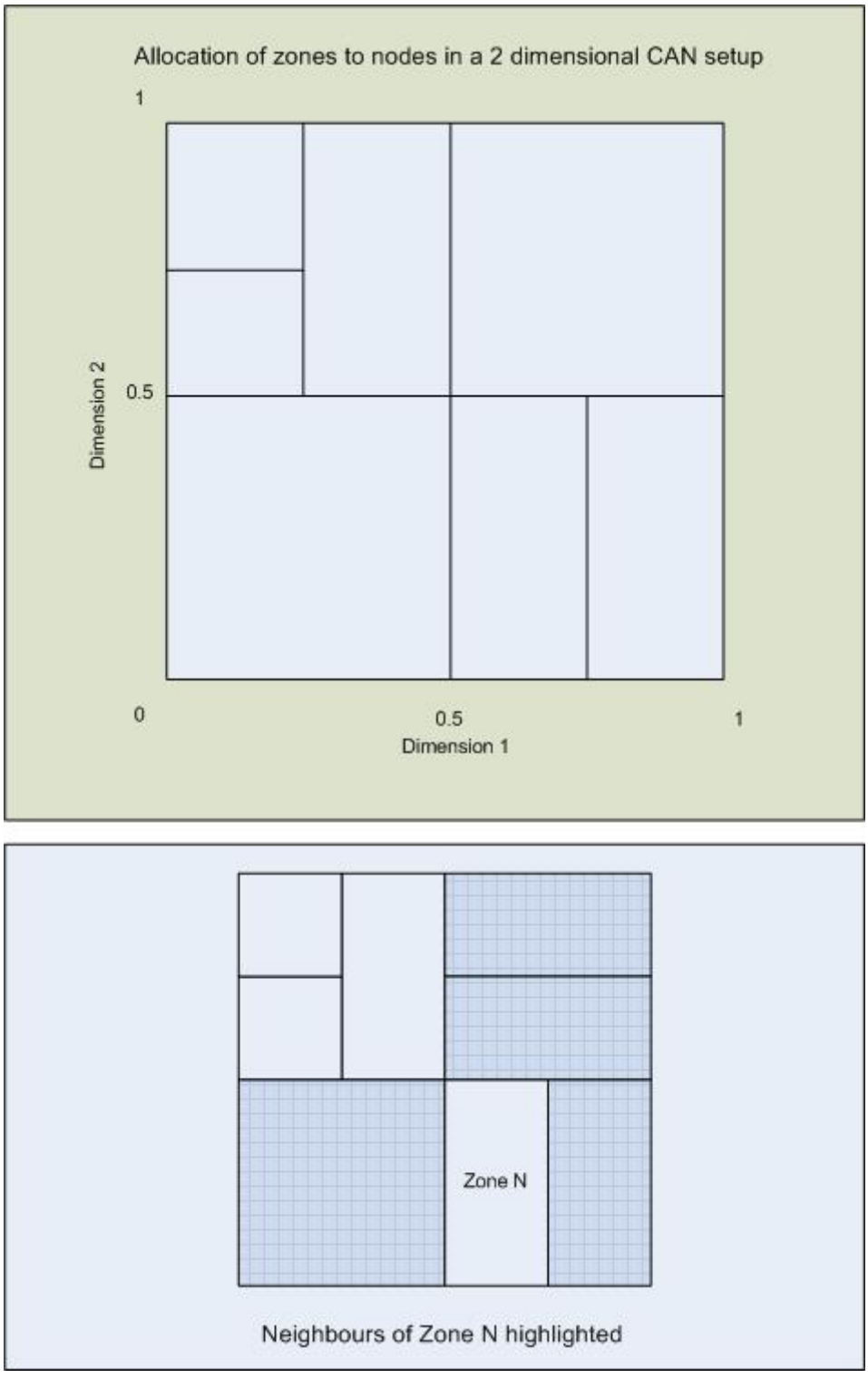


Figure 3.3: A Content-Addresable Network

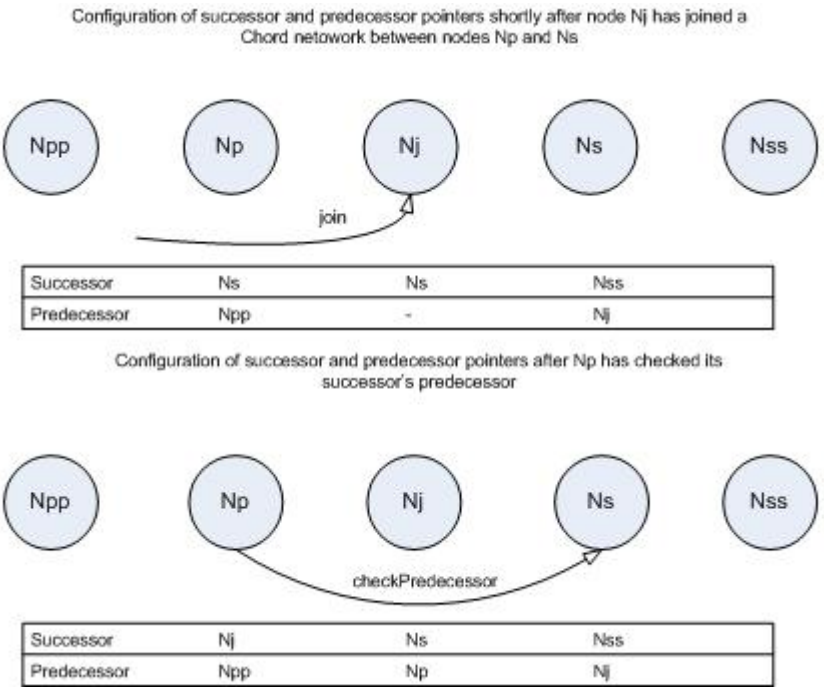


Figure 3.4: Illustration of the Chord stabilisation protocol. Nodes periodically check their successor's predecessor, and if the result is not equal to their own node ID and less than their current successor's node ID, they change their successor accordingly and advise the new node that they might be its predecessor

i^{th} entry in the table for a particular node *N* contains the address of the node that succeeds *N* by at least 2^{i-1} in the identifier circle. Each entry in the finger table defines an area of the circle, and contains the address of a node that messages relating to keys within that range should be directed to. The node contained in the finger table might not be the node ultimately responsible for the key, but it is certain to be topologically closer, and by consulting its own finger table will find a node closer still, until eventually the actual successor node for the key is found.

- For reachability of all keys to be maintained in the network, the successor value needs to be correct at all times. If a node always knows what its successor's address is, a lookup for a particular key can always by succeed just by iteration through successors until the responsible node is found. For lookup performance to be maintained, it is also necessary to keep the finger table current. Chord has a set of operations it performs when nodes join and leave the network, and periodic stabilisation checks to ensure these

properties are satisfied. To assist in the joining process, nodes also keep track of their predecessor, i.e. the node immediately behind them in the idendifier circle. A node joins a network by calling another node already in the network and requesting that it lookup the values for the joining node's successor and finger table entries. Once this is complete, the node notifies other existing nodes that might need to store its details in their own finger table of its arrival. The stabilisation check involves each node contacting its successor and querying it for its predecessor. If a new node C has entered at a point in between node A and node B, node A's successor's predecessor will be Node C rather than node A. Node A would then change its own successor from node B to node C, and notify node C that it thinks it might be C's predecessor (see figure 3.4).

To enable recovery from failure of individual nodes, each node also keeps a list of its *r* nearest successors on the identifier ring (where *r* is a configurable parameter). If the first successor does not respond to a query, it is assumed to have failed and the next successor from the successor list is tried until a successful response is achieved.

3.2.4 Problems To Consider With Distributed Hash Tables

- Malicious nodes. When contemplating the design of a network application, it is prudent to consider potential vulnerabilities to hostile actions from malicious parties. One of the properties relied on to maintain the scalable behaviour of the DHT protocols reviewed is that keys are distributed among nodes in a more-or-less even fashion. This helps ensure a balanced workload across the network and allows users to experience the performance levels they would expect. The nature of encryption algorithms such as MD-5 and SHA-1 is such that, given a random set of values to encrypt, they will produce a set hash values distributed evenly along the potential range of values. A hostile node could compromise this load-balancing effect by deliberately selecting a large number of keys grouped in a small numerical range and performing operations on them.

- A further security consideration arises from the possibility of intermediate nodes divining information about their peers by analysing query traffic being routed through them. It has been shown by O'Donnell et al [19] that, while there is no way to be protect the identity of a node that stores a particular item of data, it is a lot harder than it might at first seem to discover much about who has actually submitted a query to the DHT.

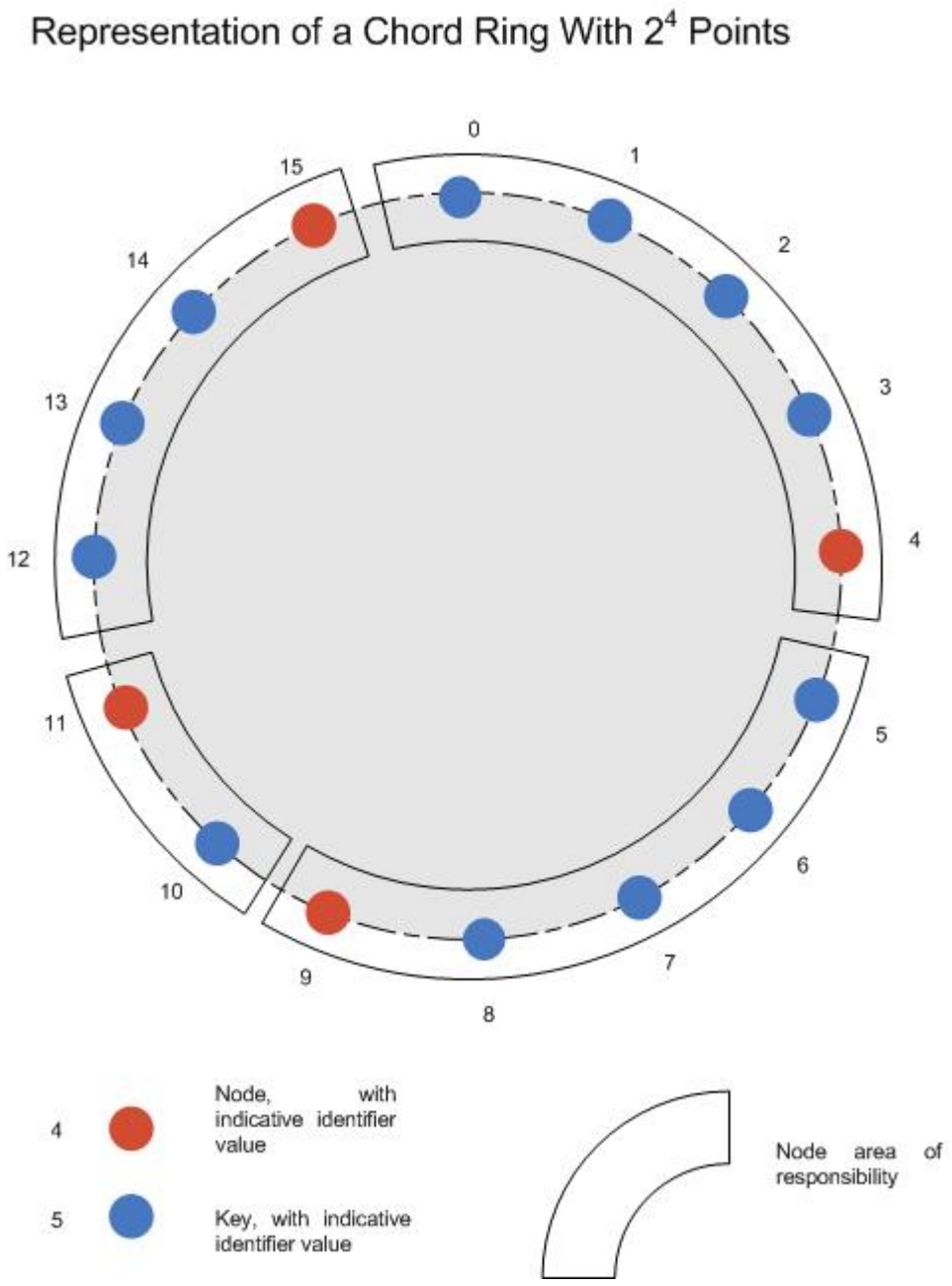
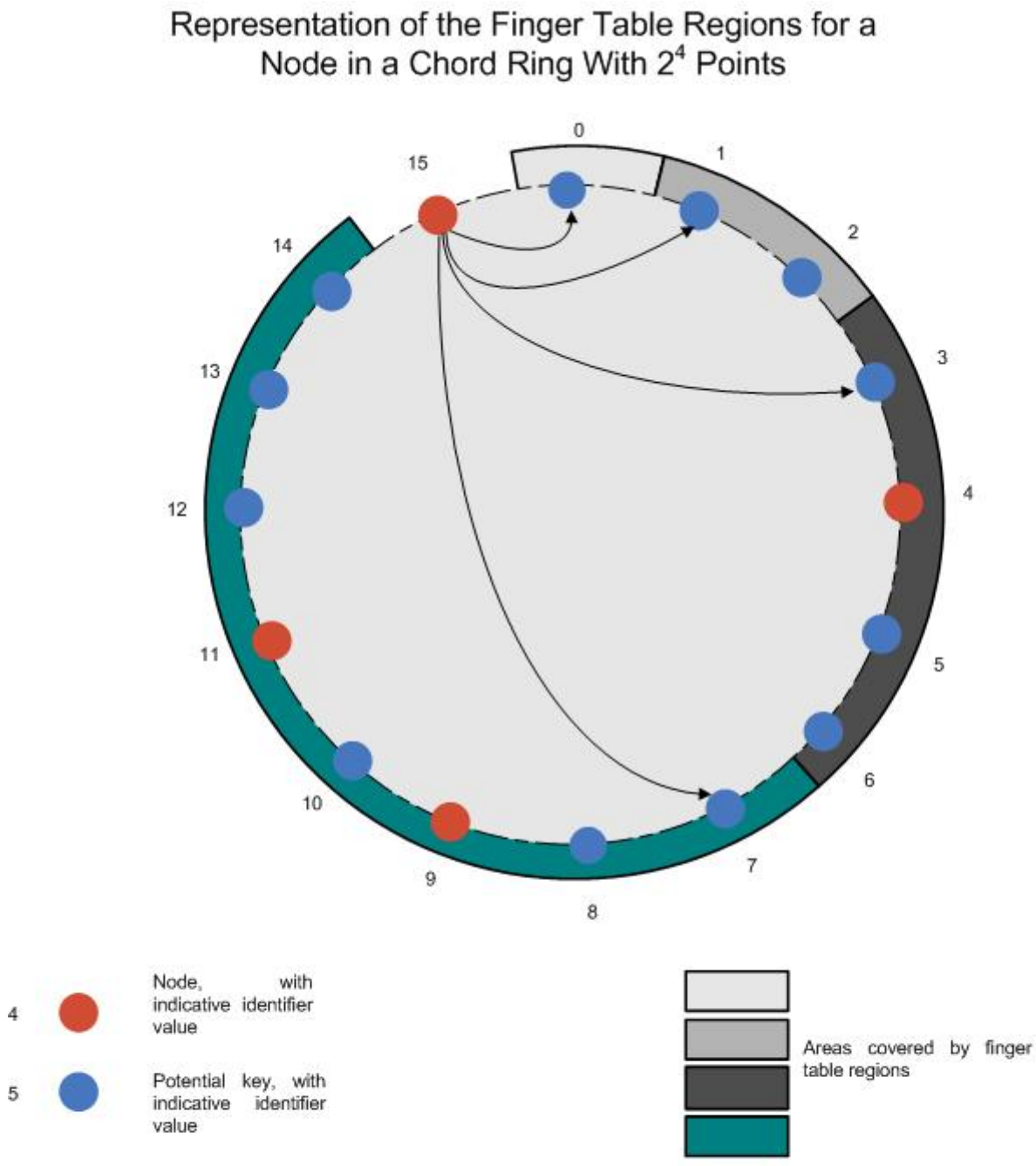


Figure 3.5: A Chord Ring



The finger table for node with identifier 15

Starting Value	Interval	Successor Node
0	[0, 1)	4
1	[1, 3)	4
3	[3, 7)	4
7	[7, 15)	9

Figure 3.6: Successor Regions in a Chord Ring

- In certain circumstances it is possible for a DHT network to be fragmented by forming disjoint sets of nodes. One technique to mitigate against this would be to designate specific keys as reference points, and for all nodes to periodically query these reference points. Failure to locate reference points would indicate the presence of a partition, which could potentially be addressed by consulting a cache of randomly-selected nodes encountered in the past.

- DHTs are efficient at returning specific keys, but users of applications that typically use mass data storage (e.g. RDBMS users) often need to search for a range of values rather than a specific single value. There is no inherent support for such operations in any of the DHT protocols examined; a resolution to this issue would have to come from an application layer above the DHT [20]. In the context of internet search, queries involving wildcard characters are an obvious example of a situation where this type of functionality would be required.

3.2.5 Performance Comparison

- In a DHT it is desirable for the number of hops between nodes a message has to go through to reach a given key to be minimised. For both Pastry and Chord it can be shown that the determining factor for the typical number of hops a message would take is the size of the network, and that in both cases for a network of n nodes the number of hops is $O(log(n))$. CAN message paths are determined by a combination of network size and the number of Cartesian dimensions specified in the identifier scheme at the time of initiating the network. For a network with n nodes and d Cartesian dimensions, the number of hops required to route a message is $O\left(dn^{\frac{1}{d}}\right)$. CAN has the same path size as the other protocols when $d = log\,(n)$ but as the number of Cartesian dimensions in the CAN setup cannot be varied dynamically, whereas the number of nodes can, the performance is in general not the same and the path length is usually longer.

- Each flavour of DHT has to commit resources to maintain a representation of the current state of the network as they see it, i.e. addresses and areas of responsibilities of neighbouring and distant nodes. In CAN's case, this is in proportion to the number of Cartesian dimensions chosen on system setup. In the cases of Pastry and Chord, it is of $O(log(n))$. In all cases, this should be well within the capabilities of modern machines both to store and to keep updated.

- The key to usefulness and robustness for a peer-to-peer system is its ability to compensate when peers disconnect, and to integrate new arrivals promptly. Changes in network membership require nearby peers to adjust their neighbour lists, and distant peers to adjust routing information (i.e. finger tables, routing tables) that allows a message from a distant peer to get be sent somewhere nearer its ultimate destination. Correctness of information in neighbour lists is generally sufficient to satisfy the condition that the message eventually gets to its destination, as the message can just progress from one immediate neighbour to the next repeatedly until gets to where it needs to be. However, for satisfactory response times to be achieved, the network should be able to take advantage of the acceleration of routing gained by using the extended neighbourhood information contained in its routing tables, and for this to happen, the routing data should be usable under conditions where the live node population is volatile. The algorithms proposed for each system all provide evidence that they can cope well in this sort of situation, although CAN perhaps has something of an advantage in that the size of its immediate neighbour set can be made arbitrarily large.

Chapter 4

PageRank

The PageRank algorithm forms the basis of the ordering of search results returned by Google, and hence is an important topic to consider when looking at returning search results. This chapter will review the basic PageRank algorithm, consider modifications that allow it to be used in a distributed environment, look at some properties the distributed algorithm is expected to display, and finish by identifying techniques for enhancing the algorithm to speed up calculation, enhance the relevance of its output and fine tune it for personalised results.

4.1 The Static Algorithm

The PageRank algorithm in its basic form is a relatively straightforward calculation; complexity in its use arises largely from the size of the datasets it is used on and the corresponding scale of computational hardware required to calculate it. The algorithm can be expressed as follows:

$$R = dAR + (1 - d)C \tag{4.1}$$

where

- R is a column vector representing the PageRank value of each page
- A is a square matrix representing the link structure of the web of pages being analysed, with values populated such that:
  - if a link exists from page i to page j then A_{ij} has the value $\frac{1}{d(i)}$ where $d\,(i)$ is the number of outlinks of page i.
  - A_{ij} has the value 0 otherwise.

- d is a scalar term known as the damping factor which influences the amount of weight given to link structure rather than random effects. It also has some bearing over how long the solution takes to compute.

- C is a column vector referred to as the transportation vector. Its purpose is to take account of pages that receive no inbound links from other pages; as well as receiving a PageRank citation from any pages that link to them, each page receives a citation from the transportation vector. It simulates the effect of a 'random surfer' clicking on any page with equal probability; thus the value of a PageRank contribution from the transportation vector is equal to $\frac{1}{n}$ where *n* is the number of pages in the web being analysed. N.B. the transportation vector can also be treated as a 'personalisaton vector', in which case the values at each point in the vector will not all be the same. This models a situation in which an individual prefers certain pages over others, such as a personal homepage, and thus has a greater probability of selecting those pages.

The PageRank algorithm amounts to finding the principal eigenvector of the link matrix of the web, and a numerical solution for it can be calculated using the following iterative method:

$$R_{i+1} = d * AR_i + (1 - d) * C \tag{4.2}$$

where

- R_i and R_{i+1} are PageRank vectors from generated from successive iterations of the algorithm.

- The initial values of the elements of the PageRank vector R_0 are set to 1.

- The equation is said to have converged when the one-norm of $R_{i+1} - R_i$ is less than a specified small value ε. The converged vector constitutes the set of final PageRank values for all of the pages contained in the web graph.

While there are a number of different possible techniques for determining eigenvectors from matrix equations of this form [21], Google themselves are known to use this naive iterative technique, known as the power method. The key benefit of the power method when dealing with internet-scale web graphs is that it does not require modification of the matrix representing the link structure. Alternative techniques reduce the number of iterations necessary before the specified level of accuracy in the eigenvector is attained, but at the cost of incurring resource-intensive copy or modify operations on the link matrix A.

A point to note about the static algorithm is that the total amount of PageRank in the web being analysed does not vary, it merely gets redistributed among pages. Every page starts with the same initial value at the start of the iteration cycle. Once the calculation has converged, some pages will have the minimum possible PageRank $(1 - d)$ and others will have a rank substantially higher than the initial base rank, depending on the number and quality of citations they receive from other pages.

4.2 Computational Requirements

The PageRank calculation undertaken by Google is a significant computational problem because of the size of the dataset involved; it amounts to determining the eigenvector of a matrix of 4 billion * 4 billion elements, and is thought to take several days to calculate. In an iterative computation of this scale, it is important to understand the factors affecting convergence and the impact they can have on the quality of results. Some proposed refinements to the algorithm are discussed later on, but in the basic algorithm previously outlined, the key variables affecting convergence are:

- The damping factor *d*. The original paper from Page & Brin [1] states that this is usually set somewhere between 0.85 and 0.9. Adjusting it has an impact not just on the convergence time of the iterative calculation, but also on the qualitative properties of the results. The larger the damping factor, the more emphasis the final vector places on citations rather than random factors. A higher damping factor should result in a higher level of accuracy in the PageRank value, but at the cost of requiring more iterations of the algorithm to reach convergence.

- The convergence error term ε. There is a threshold at which reducing the size of the error term to produce a more precise value for the PageRank vector results in little to no change in the relative rankings of individual pages, or the composition of particular percentiles. The objective when deciding what level to set the error term at will be to ensure that it identifies pages in the higher percentiles accurately (as it has been shown that search users are rarely interested in more than the first 100 results). A threshold of 1% strikes a reasonable balance between speed of convergence and accuracy of results.

4.3 PageRank in a Distributed Environment

Modifications to the basic PageRank calculation that lend themselves to use in a distributed environment have been proposed by Shi et al [22] and Sankaralingam et al [23]. The distributed versions differ in several respects from the static calculation.

- They are designed to operate asynchronously, i.e. different regions of the calculation will not necessarily converge at the same time. Nodes may enter and leave at any time and messages between nodes may arrive out of sequence.

- They allow for the flow of rank in and out of the systems. There is no requirement that the total sum of PageRank within the system remain constant, as it does with the static algorithm. This loosening is necessary in a distributed environment as otherwise it would be necessary for each peer to know of the existence of every other document in the network; in an environment with fluid membership this is not practical and would conflict with one of the main objectives of a distributed application, that participants should be able to operate effectively within the network with a bare minimum of knowledge of the state of other members.

Two variants of a distributed PageRank algorithm are proposed by Shi et al [22]. The mathematical foundation for their technique involve a principle termed *chaotic* or *asynchronous relaxation* developed by Chazan and Miranker [24, 25]. A detailed discussion of chaotic relaxation is beyond the scope of this report, suffice to say that it provides a justification for solving linear systems in circumstances where operations on the elements of the system are performed in non-deterministic order. The algorithm itself relies on the concept of a Group PageRank (see equation 4.3). This allows for PageRank values to be converged locally within a group (i.e. at a particular node in a peer-to-peer network) and then transmitted out to remote nodes without needing to know more than the an address of another node within another network (as opposed to needing to know the entire link structure of the underlying web graph). The Group PageRank algorithm takes as its inputs the link structure inside the localised group of pages, a constant vector that corresponds to the transportation vector, and the value of any citations from pages outside of the group. It produces as output the PageRank vector of the group and a vector for citations for any pages outside of the group. There is no requirement that the pages referred to in this external group actually need to be present in the wider network, which relieves the local group of the potentially message-intensive task of verifying their existence.

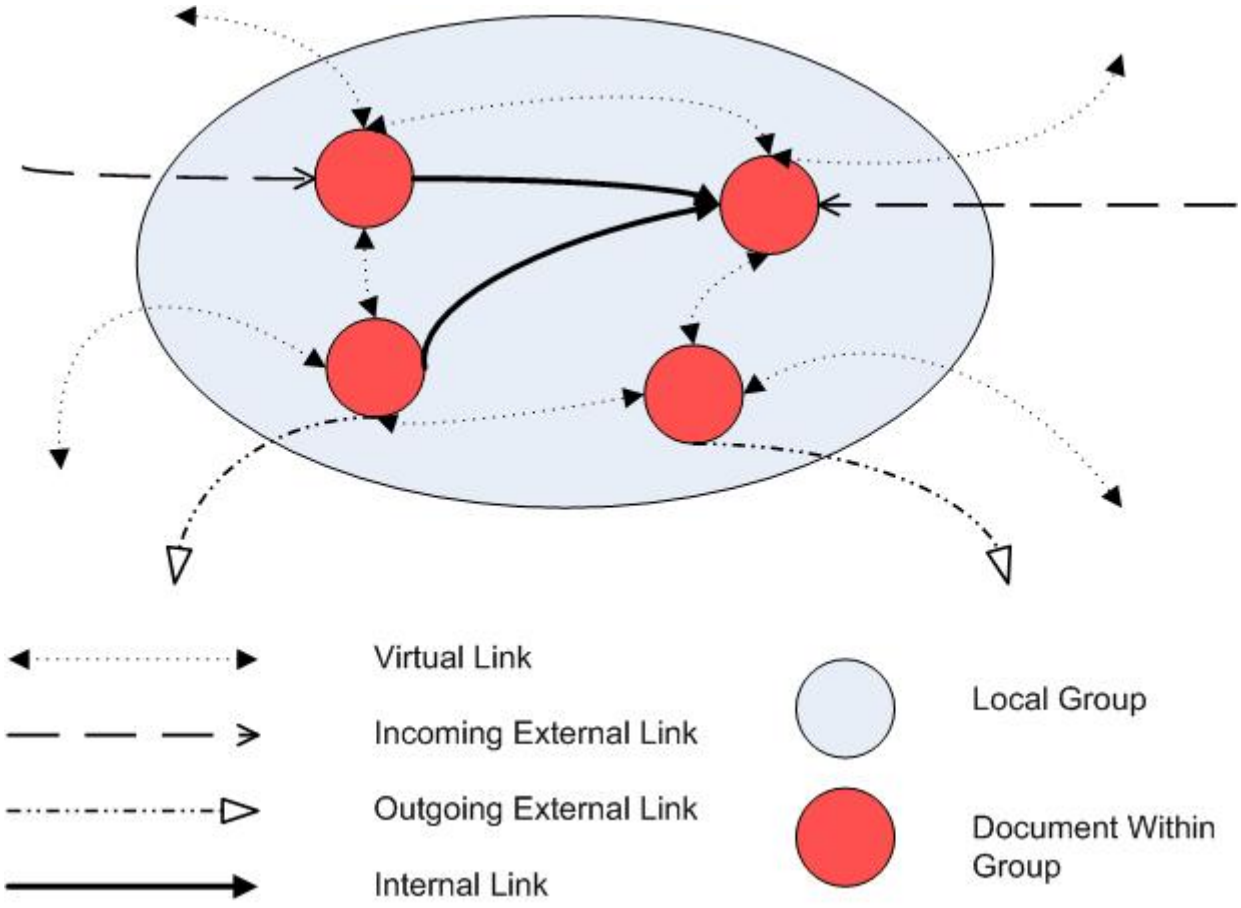


Figure 4.1: Message-passing in a Distributed PageRank Calculation

$$
\begin{aligned}
GroupPageRank =& (R_0, X)\ \{ \\
&\text{repeat} \\
&\quad R_{i+1} = AR_i + \beta E + X \\
&\quad \delta = \|R_{i+1} - R_i\|_1 \\
&\text{until}(\delta < \varepsilon) \\
&\text{return } R_i \\
&\}
\end{aligned}
\tag{4.3}
$$

where:

- A is a square matrix representing the link structure of the group (as in the static PageRank calculation).
- R_0 is the initial value of the PageRank vector. An arbitrary value can be chosen as long as each element of the vector has the same value.
- R_i and R_{i+1} are values of the PageRank vector generated by successive iterations of the calculation.
- ε is a predefined acceptable margin of error.
- X is a vector of citations from pages external to the group. These will be sent to specific pages in the group as and when external groups are ready to send them. The group will store any incoming citation values until such time as it is ready to recalculate its own PageRank vector.
- E corresponds to the transportation vector.
- β is a scalar value corresponding to $(1 - \alpha)$ where α is the damping factor.

The localised Group PageRank algorithm produces as output revised values for the local PageRank vector and a vector of citations Y for pages outside the local group, which should be transmitted once the local PageRank has converged.

$$
\begin{aligned}
&DPR1\{\\
&\quad R_0 = S\\
&\quad X = 0\\
&\quad loop\\
&\qquad X_{i+1} = \text{Refresh} X\\
&\qquad R_{i+1} = \text{GroupPageRank}(R_i, X_{i+1})\\
&\qquad \text{Compute Y}_{i+1} \text{ and send}\\
&\qquad \text{Wait}\\
&\text{while true}\\
&\qquad \}
\end{aligned}
\tag{4.4}
$$

$DPR1$ seeks to converge the PageRank value of local pages at each iteration before transmitting an updated PageRank to external links.

$$
\begin{aligned}
&DPR2\{\\
&\quad R_0 = S\\
&\quad X = 0\\
&\quad loop\\
&\qquad X_{i+1} = \text{Refresh} X\\
&\qquad R_{i+1} = \text{AR}_i + \beta E + X_{i+1}\\
&\qquad \text{Compute Y}_{i+1} \text{ and send}\\
&\qquad \text{Wait}\\
&\text{while true}\\
&\qquad \}
\end{aligned}
\tag{4.5}
$$

$DPR2$ calculates and communicates revised PageRank values out to external groups immediately upon receipt of a citation in from an external group.

4.4 Performance Considerations

One of the benefits of peer-to-peer networks is that it is not necessary for every member to be aware of every other member. In the context of a set of documents connected to form a web graph, it is conceivable that any page could link to any other. If a naive view of this were taken, it could be supposed that each might need to communicate directly with every other peer in the network in order to pass on

PageRank citation messages correctly. In a network of N nodes, this would result on the generation of $O(N^2)$ messages, which is not sustainable with increasing network size. However, the DHT protocols considered all offer some concept of neighbourhood or routing tables. This can be leveraged to reduce the amount of traffic between nodes to a scalable level. The principle is that PageRank messages are generated from a local group, sorted into buckets by destination using the values in the routing table for the local peer and sent to those destinations. On receipt at the remote destination, the citations are either applied to documents held in the remote destination's local group, or re-sorted into buckets for further remote destinations (less remote than the first), and combined with any outgoing messages from the remote group.

4.5 Potential Extensions to the PageRank Algorithm

The PageRank algorithm by itself provides a useful guide for relevance of pages relative to each other, and is especially effective in cases when the search terms supplied are vague. However, a number of additional mechanisms are known to be employed in conjunction with it to enhance the quality of search results. In addition, proposals for further modifications and enhancements to it have emerged from research work.

The papers published while Google was still an academic project [3, 2, 1] refer to the practice of considering the text displayed on link tags and other attributes of the HTML contained on referring pages. They also refer to a process of analysing the proximity of individual search terms to each other on the results. Page and Brin are quite emphatic that the quality of their results owes a lot to their analysis of the non-textual features of documents.

Brin et al [3] explicitly mention PageRank's resilience against spamming; Google's success has provided a great incentive for people to try and compromise that resilience. There are clearly commercial advantages for sites receiving high search engine ratings, and this has led to the growth of a small industry providing 'search engine optimisation' services (at time of writing a Google search for 'search engine optimisation' reports 3,420,000 results, against 7,830,000 for 'George Bush' though presumably only the first couple are worth looking at). Some internet users consider a high PageRank rating on their site to be a matter of personal prestige, and go to great pains to boost it; web log enthusiasts have been identified as culprits of this practice [26]. Both of these examples can result in the artificial inflation of a page's PageRank rating, thereby diluting its usefulness as a measure of a document's relevance. Google themselves certainly have tactics for combating this type of practice, and provide guidance on the sort of things webmasters should avoid doing if they wish to avoid being dropped from the in-

dex (such as including long lists of invisible random text, participating in spurious link rings, etc. [27]). However, Google have not released technical details of how these ploys are identified for obvious commercial reasons, so a distributed citation-based document rating system would need to develop safeguards against artificial manipulation independently from Page and Brin's published research.

Haveliwala [28] proposes adding extra specialisation to PageRank by incorporating the concept of topic-sensitivity. The suggestion is that a page should not be assigned a single relevance rating, but should be assigned 16 different ratings, one for each of the main Open Directory Project [29] topic headings, in each case using a personalisation vector derived from pages found under the respective ODP topic heading. When a query is submitted, each of its terms is evaluated against these topic headings, and the results generated are formed from a linear combination of the corresponding topic results (for example, a search for 'Microsoft' might give a 60% weighting to results from the 'computing' topic and a 40% weighting to results from 'business'). This method also sports the ability to assess *context* either based on analysis of the word content of the location from which the query is launched, or on the search patterns of the user.

The 'transportation vector' contained within the basic PageRank algorithm, as has been said, has the potential to be used as a 'personalisation vector', giving differing weightings to pages based on some preferences of the individual. This idea is explored by Jeh and Widom [30]. The personalisation offered is not completely free, relying on combinations of sets of vectors generated from a set of popular 'hub' pages, but it does seem to present an opportunity to enhance relevance over and above that offered by both the basic PageRank algorithm and the topic-based version [28].

As has been mentioned previously, the power method is thought to be the only feasible technique for computing a dominant eigenvector for a matrix of the size necessary to represent the entire internet; the resource requirements are such that the number of operations on the main link matrix need to be kept to a minimum. However, for smaller matrices representing subsets of the internet (e.g. special interest areas or corporate intranets) the resource demands are lower and alternative techniques that can accelerate the production of a converged PageRank vector can be considered. Andersson et al [21] discuss the Arnoldi method, concluding that it converges in fewer iterations, at the cost of incurring memory-intensive operations. Kamvar et al [31] propose a technique they refer to as Quadratic Extrapolation to accelerate convergence of the basic power method, whereby they periodically use linear combinations of the first three eigenvectors of the current power method iteration to derive a closer approximation of the principal eigenvector, and show that this speeds up the convergence of the PageRank calculation by a factor of 3 without incurring any significant additional computational overhead.

It has been observed that the web graph of the internet displays certain patterns

when considered at different levels. At the macro level, it can be visualised as a bow tie shape [32] where different sections of the bow tie have different connectivity properties; at finer levels of granularity there are recurring patterns in the link structure of individual sites that can be exploited to accelerate the calculation of PageRank values. In particular, Kamvar et al [33] point out that there is frequent clustering of links into local groups based around particular hosts. Intuitively this makes sense when one considers that pages on a particular site will often all contain a link to the homepage, and will commonly have as part of their 'look and feel' a standard set of links to the various main areas of the site. This block structure is used as the basis for accelerating calculation of the global PageRank vector by first evaluating a local PageRank, providing a relative ordering for pages within the same domain, then combining this with a rating for the importance of each domain block. As well as accelerating the calculation, Kamvar et al point out potential applications for this technique in supporting personalisation of ranking based on favoured domains.

Chapter 5

Implementation

To illustrate the feasibility of providing a document ranking service in a distributed environment, it was decided to use a DHT-based peer-to-peer protocol to manage the flow of information necessary between network members. The time available made it impractical to build the DHT implementation from scratch, so the suitability of existing applications was evaluated. A number of public-domain applications were considered, including several Java-based Pastry implementations, and the version of the Chord protocol developed by Chord's originators (written in C) but in the end it was decided to adapt Naanou, a C♯-based implementation built by Clint Heyer [17], as it appeared to be a thoughtfully-constructed version of an intuitively straightforward protocol. Naanou itself is actually a file-sharing application, but its network is maintained using the methods described in the Chord papers [7], and it was felt that this aspect of the application could be usefully extracted and modified to allow the generation and transmission of PageRank update messages between nodes necessary for the study of the convergence properties of a PageRank-like algorithm in a peer-to-peer environment.

5.1 Programming Environment Used

Naanou is written in Microsoft's C♯ language, and as our PageRank code interacts closely with it, the natural choice was to use this language as well. C♯ is a constituent of Microsoft's .NET Framework, and has many similarities to Java. It is fully object-oriented, and offers an extensive range of libraries handling tasks such as managing collections, network communication, database interaction, regular expression parsing and GUI creation. Source code is compiled neither into Java-style bytecode, nor directly into machine code, but into Microsoft Intermediate Language (MSIL) which is then executed by the Common Language Runtime (CLR), another part of the .NET Framework. Among the benefits claimed for this

arrangement are:

- multiple languages can be employed according to preference, as long as a compiler that generates MSIL is used.
- the CLR supports security policies that guarantee type safety of executable code.
- the CLR handles memory management, removing this duty from the programmer. Much like Java, primitive types are passed by value, but all non-primitive types are passed by reference. The CLR keeps track of objects that have gone out of scope and periodically reclaims the memory they have been allocated (the programmer has the freedom to force this if it is felt necessary).
- Programs compiled into MSIL can run on any platform for which a CLR implementation has been written.

5.2 Implementation Design

The main concepts to model in the process of producing a distributed PageRank calculation are:

- Documents.
- Links to other documents.
- The calculation technique for deriving a refreshed PageRank value.
- PageRank update messages.
- Collections of PageRank messages.
- An abstraction representing sets of documents. In the context of our application this grouping occurs at a particular node in the peer-to-peer network rather than at a single host on the internet, which is significant as it means that there is no logical relationship between the pages, only that they happen to have hash values close to each other.
- A text-parsing object to facilitate generation of the Document objects from text files containing web graph data.

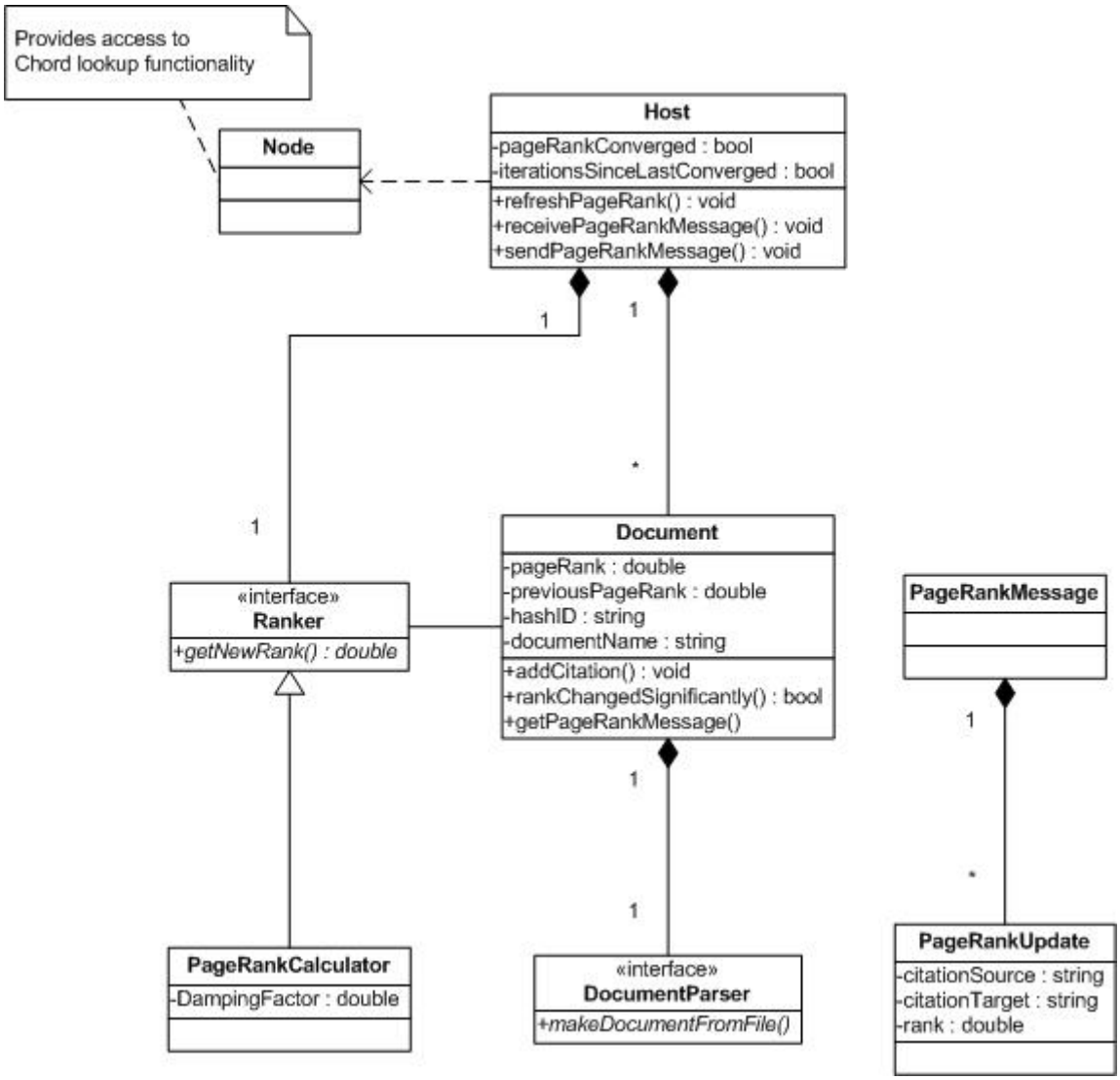


Figure 5.1: UML Diagram of objects required in distributed PageRank calculation

In keeping with the suggestion that Chord be used to provide a storage location service, we include a reference to the main Chord object (Node) in our Host object. This is used to access any of the functionality required to query or modify the DHT contents.

The main scenarios we need to accommodate to facilitate a PageRanking service on top of a Chord layer are as follows:

- Adding document objects to the table
  - Our client generates a document object (consisting principally of an URL and list of outlinks) from externally supplied data.
  - The client calculates the SHA-1 hash of the document's URL and queries the Node object to find the successor node of that particular hash (i.e. the node in the network that is responsible for storing that key).
  - Using the address of the node supplied by the previous operation, our node calls an $insert(hash, Document)$ operation to add the document object to the DHT (n.b. the address returned may happen to be the address of the local node, if it happens to be responsible for the hash of this particular document).
- Generating PageRank messages from documents
  - The host iterates through its collection of documents, producing from each one a list of update messages, each one consisting of a source hash (the document's hash), a destination hash (the hash of one of the document's outlinks) and a PageRank contribution from the document.
  - The collection of PageRank messages from each document are added to a set of pending messages for the Host.
- Sending PageRank messages between nodes
  - Once the host has a set of PageRank messages for documents that are not stored locally, it will send them on to remote nodes.
  - To minimise network traffic, the node will not seek to determine the precise destination of each message by calling the node lookup function. Instead, it queries the local node (i.e. the Chord service) to find the *nearest* node it is aware of for the hash of each message destination. The local node consults its internal finger table and successor list to find the nearest node, thereby avoiding the need for network traffic.

  - The host then uses a remote procedure call $receivePageRankMessage(message)$ on each of the destinations returned from its local tables and clears its store of pending messages.

- Handling receipt of incoming PageRank messages from other nodes
  - When the $receivePageRankMessage(message)$ is called, the node it is called on iterates through each of the messages in $message$ to determine if the message should be applied to a document stored locally.
  - If the document is stored locally, the node adds message to the document's collection of citation messages.
  - Otherwise, the node adds the message to its own collection of pending outgoing messages. N.B. a received message that is not applied to the local node should at least be *closer* to its ultimate destination.

- Recalculating document PageRank values when citations are received
  - To facilitate the use of varying methods of calculating PageRank, we use an interface (*Ranker*) that has a *Document* property and a *getNewRank()* method.
  - A concrete instance of this interface is contained in the appendix.
  - The *getNewRank()* method when called assigns the document a new PageRank value. The document also keeps track of its previous PageRank value, to allow it to check whether the rank has converged or not.

The account of the Chord protocol provided by Stoica et al [7] states that the interface provided to client applications should consist simply of a single operation mapping a key to a node of the form $lookup\,(key)$ returning $keylocation$. The implementation of the protocol in Naanou contains this functionality, but also includes a great deal of other material as well. Much of this proved to be useful but did present an additional level of complexity to the task of passing citations between nodes. Furthermore, changes to the .NET Framework introduced by Microsoft and the fact that Naanou had not been subject to the level of testing that is usual for production applications (and is no longer under active development) meant that it could not be treated as a black box. Large quantities of the application required analysis and debugging to remediate versioning issues with Microsoft's platform, rectify logic errors, and adapt functionality away from the original file-sharing application.To give a feel for some of the issues this presented, without dwelling on specific coding problems, a brief overview of the design of Naanou will be given. The application is divided into four main areas: a graphical front end; file-sharing management; a Common library containing code

used by all sections; and the Lookup service. Of these, the the Common code and Lookup service were extracted and modified to form the base of our distributed PageRank service.

- The main class of interest in the common code section is the Hash class. This acts as a wrapper for the node/key identifier, and provides various operations to support arithmetic and type conversion on these identifiers. The class implementation relies heavily on a BigInteger class that supports operations on integers of arbitrary length. [1]
- Other classes of note within the Common library provide a structure for data to be inserted against a key in the table, a structure for addressing, and a threadpool implementation.
- The majority of the Chord-related functionality is contained within the Lookup library, which has several subsections.
  - The Discovery section contains a set of classes facilitating different techniques for peers to join the network, which include broadcasting, bootstrapping from a purpose-built website and calling a specified port on a specified IP address. For the purposes of calculating PageRank, the latter technique is always used.
  - The Transport section implements a network sink client and server (the peer acts in both capacities depending on what it is doing). The classes enable data to be sent/received in binary format for speed of transmission, and are customised to allow for authentication of the caller at the receiving end. Communication is conducted over a TCP channel. Naanou was coded under version 1.0 of the .NET Framework. This has subsequently been superceded by version 1.1, which has more stringent security requirements. As our PageRank application is coded using the current version of the Framework, a number of changes had to be made to the Transport code in order for it to function correctly.
  - Central to the Chord implementation are the Node and RemoteNode classes. Node holds references to finger tables, successor and predecessor lists and key containers, and has event handlers to cope with changes on all of them. Any lookups for keys contained in the table are initiated from the Node class - if the local node holds the key it returns data from its own key storage, and if not it issues a remote call to

[1] BigInteger is contained in Microsoft's $J\sharp$ library, $J\sharp$ being their proprietary interpretation of Java. As such its interface is the same as that of the Java BigInteger class

find out where in the network the data is held. It also contains procedures to cope with joining and leaving the network. RemoteNode is a RPC stub for Node, facilitating remote calls from one peer to another.

– The WorkDispatch section contains objects supporting various remote methods. Each of these can be assumed to take an indeterminate amount of time, that will certainly be longer than the amount of time a call to a function on the local machine would take; it is therefore advantageous to be able to carry on with other work while waiting for a result, which suggests the use of multithreading is in order. Conversely, creating large numbers of threads carries an overhead that can outweigh any performance gains from using them. Naanou manages this using a threadpool. A fixed maximum number of threads are employed at any time, and specified remote procedures are added to a queue when they are called. As and when a thread finishes execution, it checks the queue for any pending work items; if present, it pops the work item from the top of the queue and calls its $run()$ method; otherwise the thread terminates.

– The two work types of most interest for our purposes are the $GotKeyWorker$ and the $FindSuccessorWorker$. These two types are also interesting in that they illustrate differences in implementation between synchronous and asynchronous calls. A call to FindSuccessorSync blocks until a result is found (the result will either return an address or confirm that a result was definitely not present in the network). A call to GetKey does not block. Once the call is issued, execution continues. Once a result is ready, the result is returned via an asynchronous callback. The calling function passes a pointer to the function that it wants to process the result of the call to the call. Once the result of the remote call is ready, it knows what function to call on the local host to get the result processed in the manner intended. There are similarities in the implementation of both types of function. In both cases, a call to a function results in generation of a call to a worker function. The worker function generates a state object and a worker object. The state object is retained in a collection of pending network requests. The worker function is added to a queue of pending work items.

5.3 Issues encountered

Some difficulties arose in the course of constructing the implementation. A brief account follows.

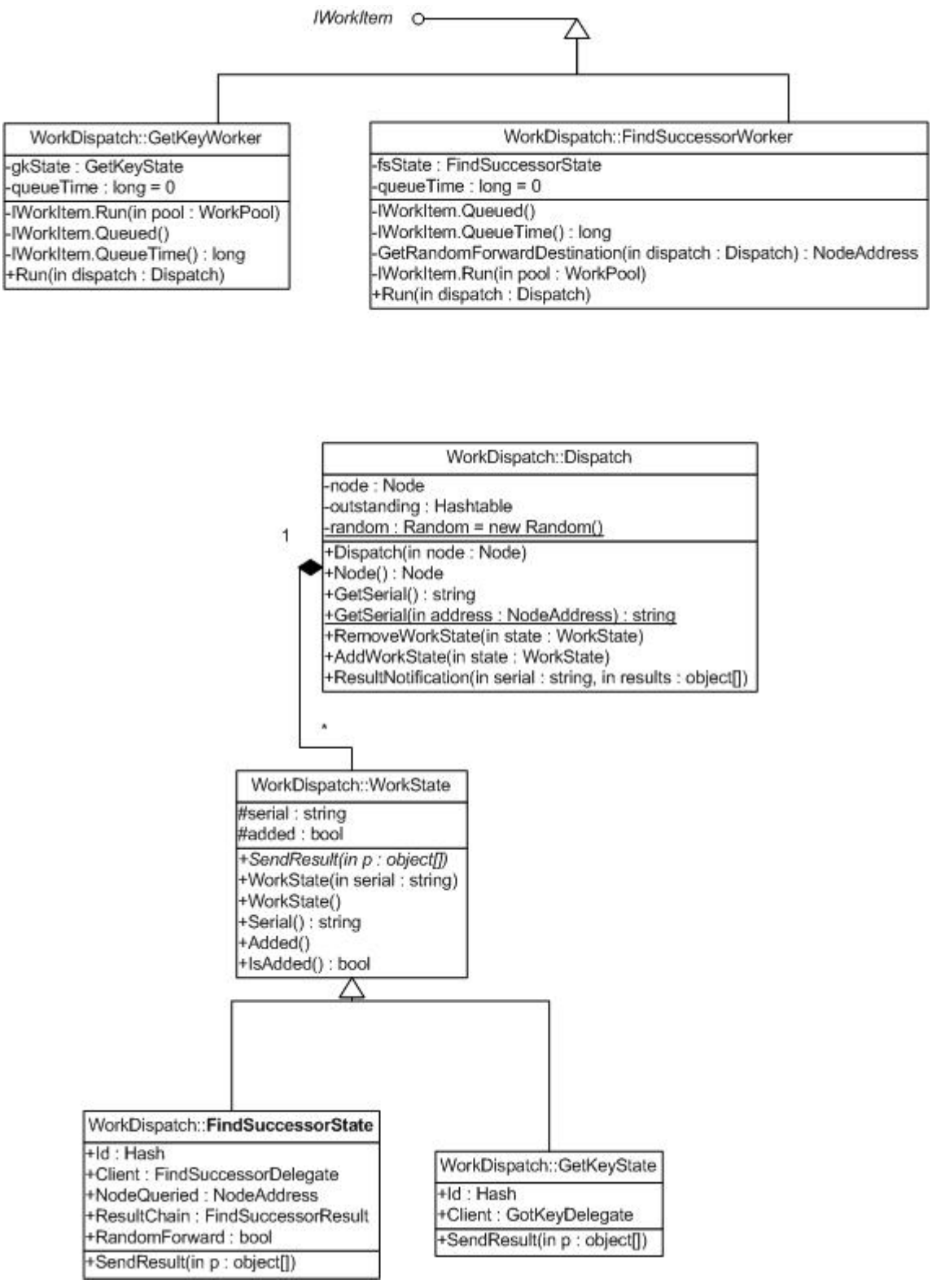


Figure 5.2: UML Diagram of Classes Involved in Key Lookup and Find Successor Operations

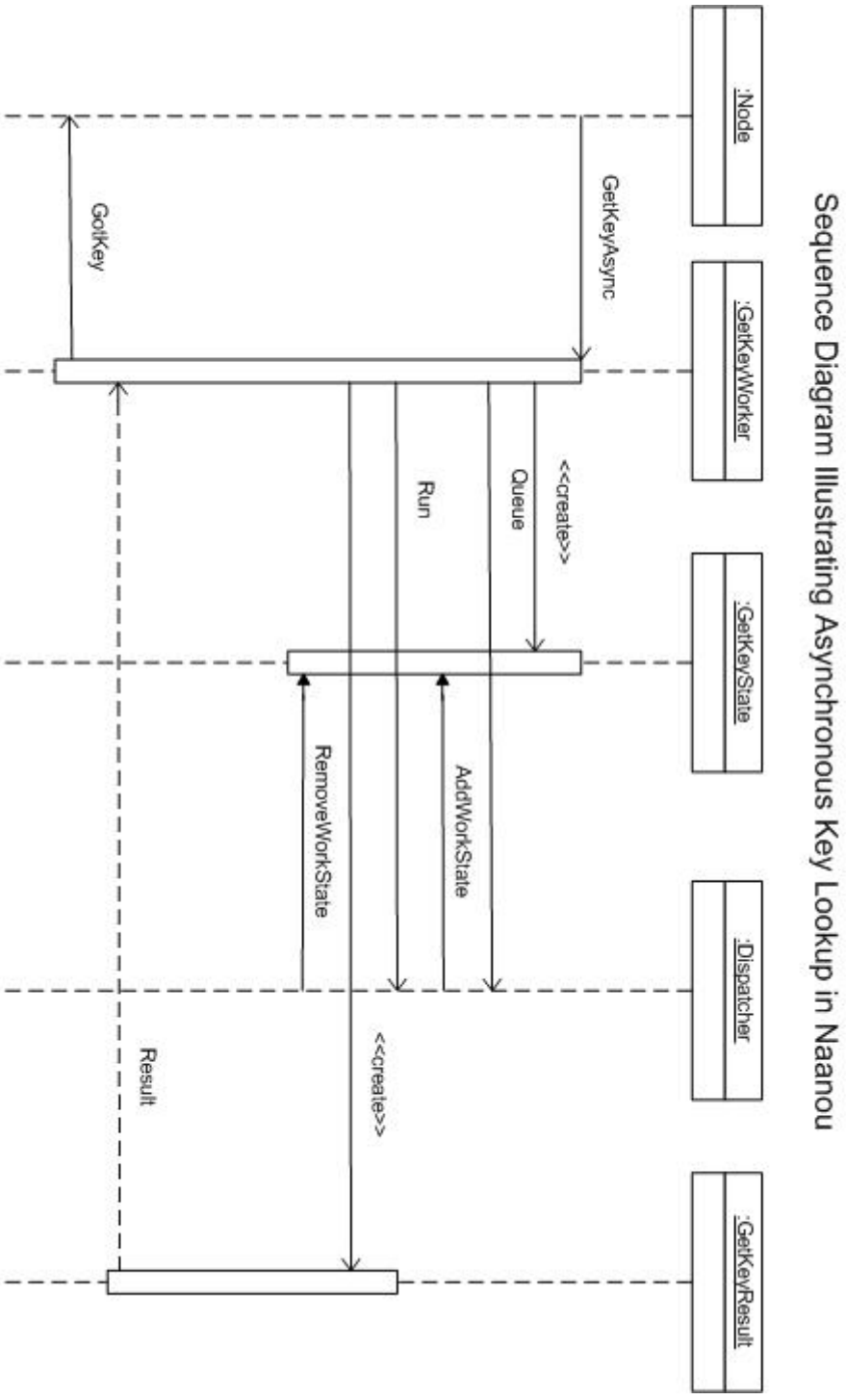


Figure 5.3: UML Sequence Diagram of Interactions Involved in Key Lookup Operation

- The Naanou implementation supports caching of keys where values have been transferred between peers through a network join/leave operation or in the course of a query. This created a problem when performing PageRanking in deciding which of the keys in the host's key storage to apply the updates to. The decision needed to be made whether to apply PageRank updates to any keys held locally, cached or otherwise, or whether the updates should be applied only to those keys that were held in the local key storage because they fell within the range of hash values the local node was currently responsible for. The consequences of the decision would be:

  - In the first case, the chances of getting a faster response to queries would be higher, as the value could be retrieved from an intermediate node nearer to the querying node than the ultimately responsible node.
  - On the other hand, queries may return different PageRank values when initiated from different parts of the network.

  It was decided to select the second option in order to simplify inspection of data for the purposes of comparing distributed PageRank values with centrally-generated values.

- The testing environment was such that the number of nodes present was too small for the proposed gains in routing efficiency from the finger table to become readily apparent. It was not practical to test systems with more than 20 nodes, and inspection of their finger tables showed very little difference in query routing from a linear progression through successors. To observe the performance gains claimed for the finger table, it would be necessary to conduct testing on a network with a higher number of nodes, whether those be genuine physical nodes or a simulation.

Chapter 6

Results

To ascertain whether a distributed PageRank measure is of value, it is necessary to verify whether the values obtained are comparable with those from a centralised calculation (i.e. that the alternative technique remains a valid assessment of a document's relevance). Furthermore, it should be determined whether advantages claimed for a distributed calculation (namely, the ability to support recalculation of rank with incremental addition of documents, rapid convergence, scalability, robustness) can be quantifiably demonstrated. The methods used to do this are described below.

6.1 Distributed Versus Centralised PageRank

To determine whether the distributed PageRank calculation produced comparable rankings to the centralised calculation, results were compared for two datasets. Firstly, as an initial check, an artificial web graph of 21 pages was created, and its PageRank derived by both static calculation and a Chord setup with 3 nodes. Such a dataset is too small for practical purposes, but at least established some basis for comparison. In testing it became clear that links from a page to itself should be discarded, as they had the effect of generating a divergent rank value (as should be mathematically obvious in retrospect). Similarly, outlinks should only be counted once per page regardless of how many occurrences of of a particular link there are on the page. The results of this comparison were broadly favourable and are shown in figure 6.1.

To perform a more realistic comparison, a larger dataset was required with a structure based on that of a genuine website. Some prepared graphs were located online, representing a snapshot of CNN's site, and some further graphs from other sites were obtained using JSpider, an open-source Java based web-crawling tool [34]. The following techniques were employed in generating the respective

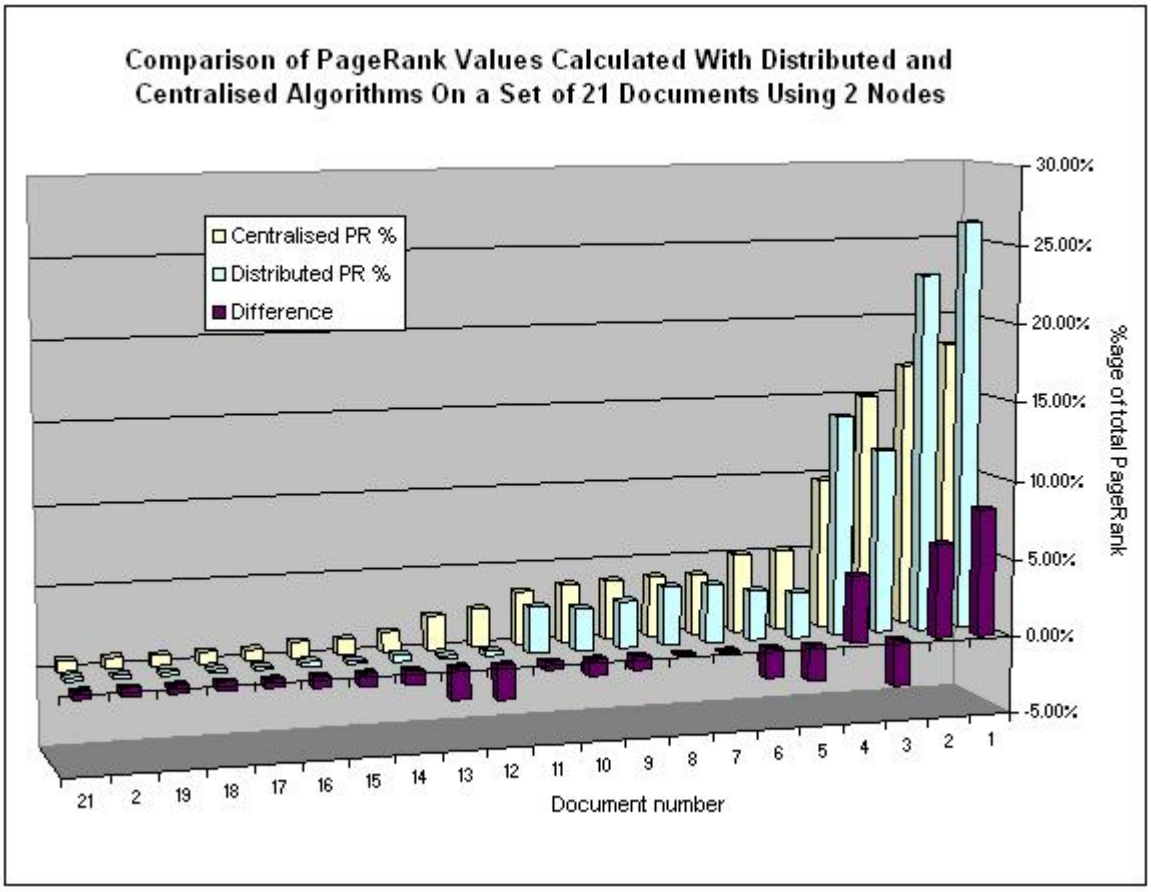


Figure 6.1: Comparison of PageRank values generated for a web graph of 21 pages

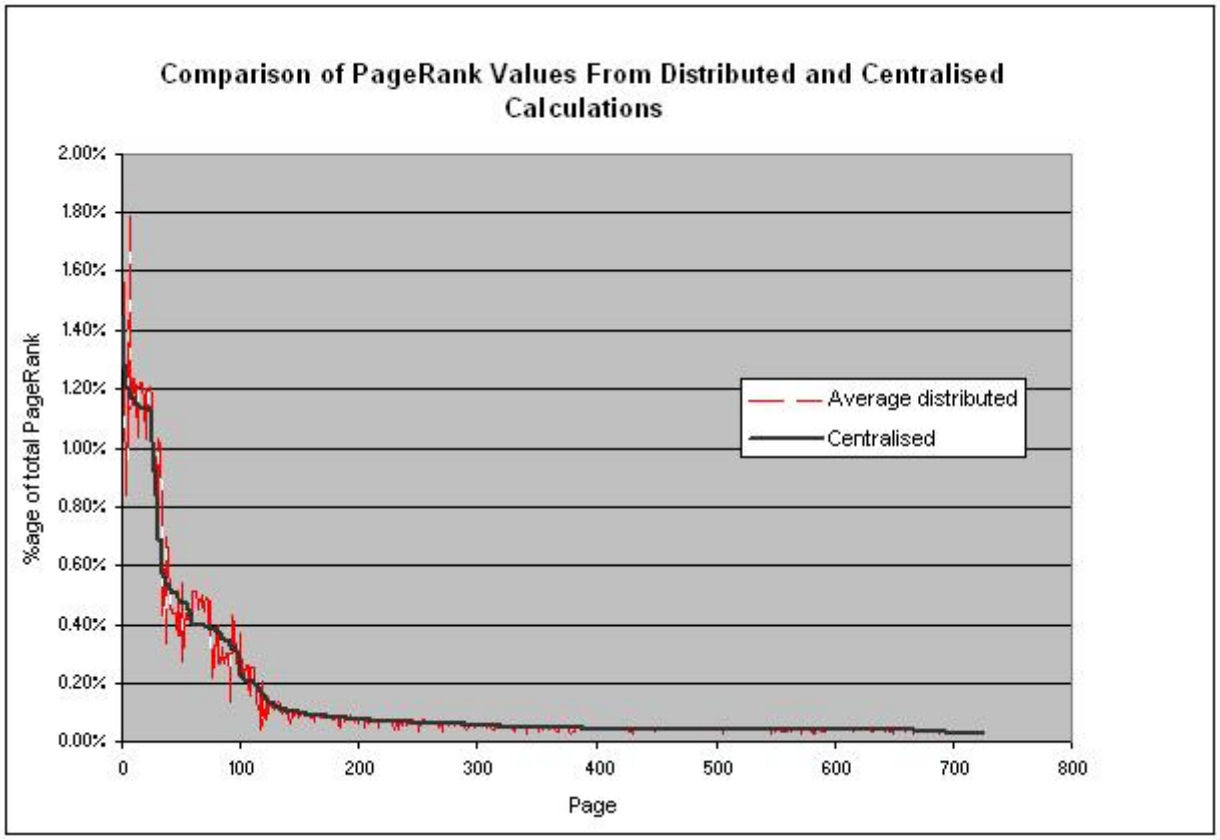


Figure 6.2: Comparison of PageRank values generated for 726 pages taken from CNN's site using distributed and centralised versions of the PageRank algorithm. The distributed value is calculated as the mean of the values obtained from five separate instances of the calculation

PageRank measures from raw web graph data:

- To support the distributed calculation a text file was generated for each page within the web graph, containing the URL of the document and the links within it (unfiltered except for any links to itself, and removal of any duplicated links). Each of these were then parsed using regular expressions to generate a Document object within our application, inserted into the DHT, and the calculation was then initiated. Once each node reported that the PageRank values of all of its documents had converged, the values were recorded.

- To evaluate the corresponding centralised calculation results, a square matrix of n rows and columns was generated from the web graph, where n is the number of individual documents. The elements of the matrix were assigned values as per the technique for populating matrix A in formula 4.2, i.e. links to pages outside of the current collection of documents were not considered to contribute to the PageRank calculation. An iterative calculation was then performed to generate a vector of PageRank values.

The actual values of PageRank derived using these two techniques should be expected to differ due to the differences in calculation method. The distributed technique is an open system using chaotic convergence, whereas the centralised version is a closed system where all operations are performed synchronously; in the open system it is possible for PageRank to leak out, as it is not known at the time citation messages are sent whether the target of the citation is actually present in the collection of documents, whereas in the closed system all citation targets are definitely known to exist. To take account of the different absolute numerical values obtained from each method, the relative rankings of each page within the system were considered (i.e. the percentage of the total rank in the system allocated to each particular page). This test was conducted with stable network conditions, i.e. no nodes joined or left the network while the calculation was in progress, and all members had time to reconcile their routing tables with those of their peers. Results are shown in figure 6.2, and show a promisingly close correspondence between the two measures.

6.2 Convergence Profile of Distributed PageRank

In a centralised environment, determining the time necessary to calculate the PageRank value is a straightforward matter of counting the number of iterations necessary before the given convergence threshold has been reached. This will vary depending on the choice of value for the convergence threshold and the damping

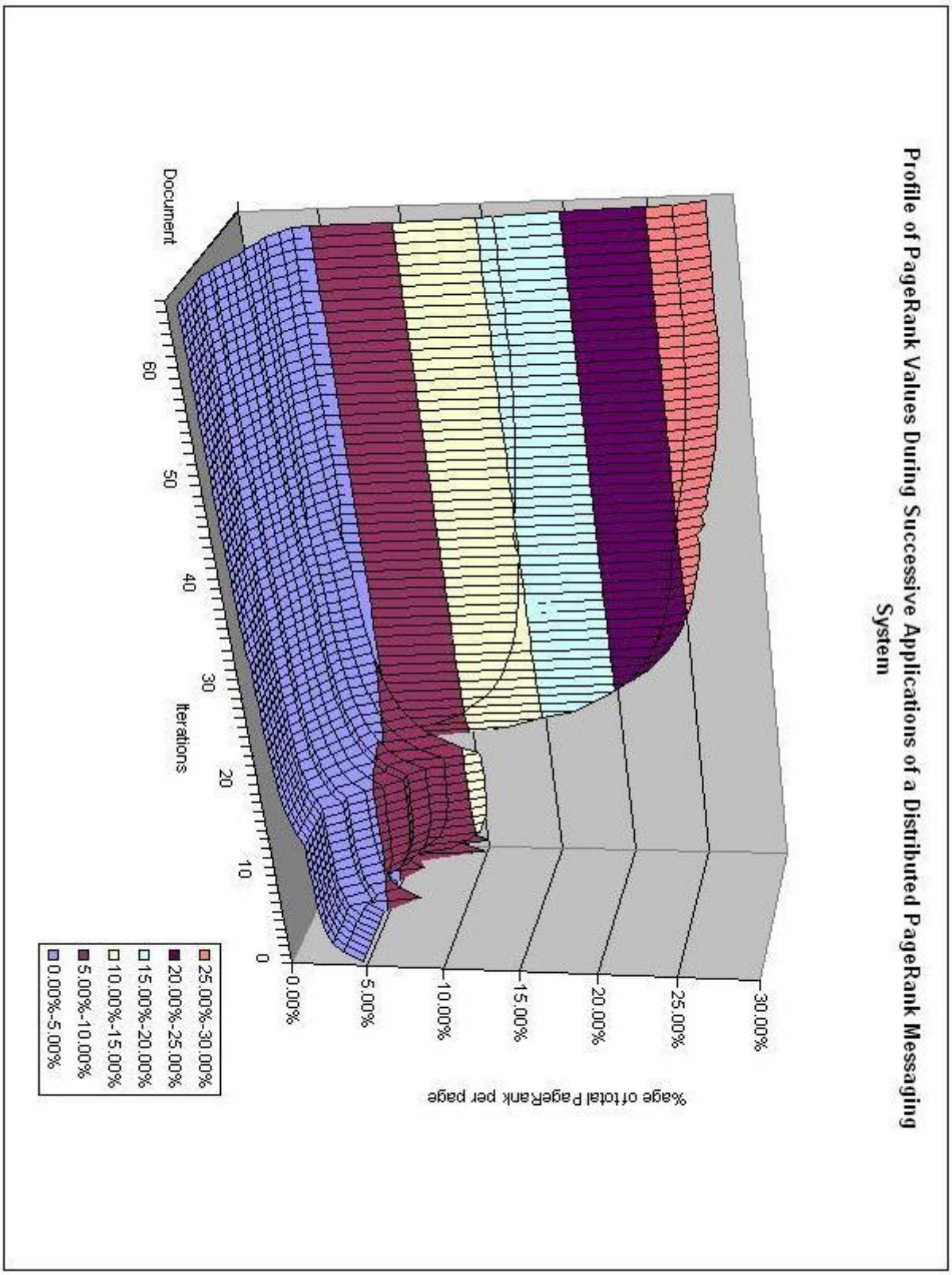


Figure 6.3: Graph displaying the PageRank rating of a sample of 21 pages over successive iterations of the distributed PageRank algorithm

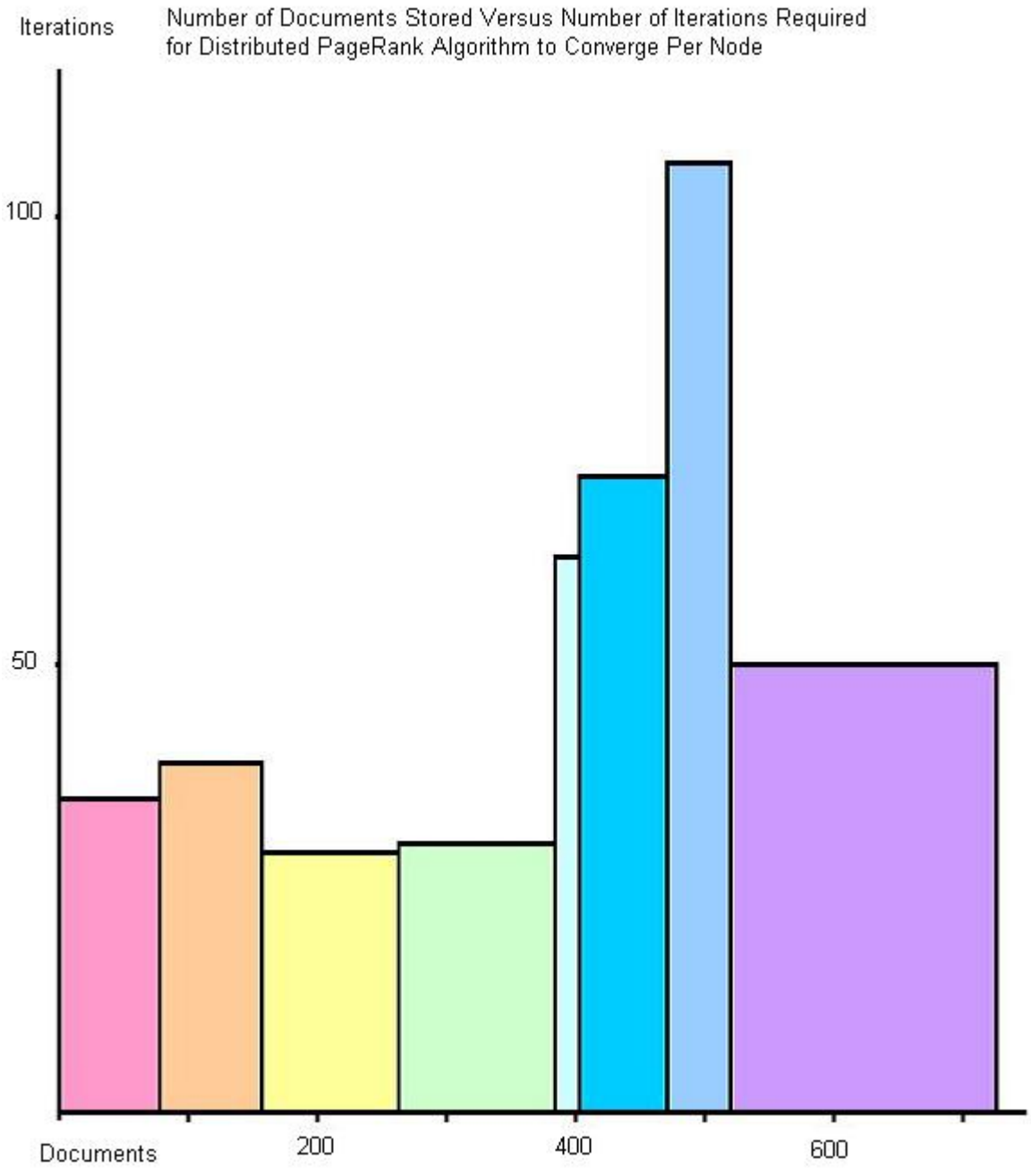


Figure 6.4: Graph illustrating distribution of documents over nodes and asynchronous nature of PageRank convergence

factor. In a distributed environment calculations occur asynchronously, citations for a particular page may arrive at different times from different sources, and as a result the PageRank for a particular page may converge prematurely, and change back into an unconvergent state; it is thus less obvious what metric to use to determine whether PageRank for the system as a whole has converged. Sankaralingam et al [23] claim that the convergence behaviour of their technique is rapid; however, they use the notion of a 'pass' the precise nature of which is not explicitly defined. In the test application each node within the network checks for incoming citations at regular intervals (1 second was chosen arbitrarily), and sends out revised citations if incoming citations have resulted in change to the PageRank value of any of the documents they contain of more than a specified error threshold (0.1% was used in this trial). Citation messages should continue to be transmitted until all documents have reached a stable PageRank value. Not all nodes stop transmitting citation messages at the same time, and some converge quicker than others. To illustrate the behaviour of PageRank convergence its value was tracked every time a set of citations was applied to a document. This corresponds to the notion of an iteration in the centralised calculation in the sense that it shows the values of PageRank for all documents in the set being analysed as a series, with each series corresponding to a specific number of samplings of citations, but making no reference to the time at which those citations were applied.

The conditions of the test were as follows:

- Eight virtual nodes running on 1 machine were used.
- The data used was the web graph of 726 pages from the CNN website, as referred to previously.
- The convergence measure used was that the updated PageRank was less than 0.1% different from the previous PageRank, and that every node reported this condition had been met for every document it held.
- The calculation was initiated and concluded when the network was in a stable state (i.e. every node had reconciled its predecessor and successor values with those of its peers, and no nodes left or joined while the calculation was in progress).
- The graphical representation portrays the PageRank value of each page as a percentage of the total amount of PageRank contained in the system at each iteration.

The results of this test appear to agree with those reported by Shi et al in [22]. Inspection of the graph of results (figure 6.5) reveals that, while the absolute values of PageRank change somewhat with higher numbers of iterations, the overall

ranking of pages relative to each other is quite stable following no more than 10 iterations. This is a promising indicator for the potential of a dynamically updating PageRank measure. In a larger system the likelihood is that documents would be added and removed continuously, such that one would be unlikely to encounter a state in which the PageRank of every document was stable simultaneously; the response to any query would have to be a snapshot of the highest ranking documents at that specific moment. However, if the speed of convergence observed in this test were maintained, all but the most recently added documents should be captured in such a snapshot.

With respect to the scalability of this calculation, there are two aspects to consider. Firstly the distribution of documents among nodes, as has been discussed earlier, should be roughly even when considered in aggregate, by virtue of the nature of the hashing algorithm used to generate key identifiers. A histogram of the specific distribution in our test is shown in figure 6.4, indicating some variation in the number of documents stored per node and in the number of iterations required for the PageRank value to finally converge. The number of iterations necessary before the PageRank value converges influences the number of messages sent between nodes. If one wished to reduce network overhead one could reduce the convergence threshold to a level such that there was sufficient accuracy to distinguish the overall popularity of pages relative to each other while losing a degree of certainty about precisely how much their positions differed. The danger of reducing the convergence threshold is that in larger systems the PageRank values of individual pages may be only subtly different; a small difference in PageRank that decides whether a page appears in the top 5 or the top 10 in a small collection is not that significant, but in a larger collection a subtle change could mean the difference between top 5 and top 500. The decision on where to set the convergence threshold is therefore one which has important consequences, especially as the number of documents contained in the DHT increases. A study of the effect of varying this parameter would clearly make sense in the context of evaluating this calculation with larger datasets and more resources.

6.3 Dynamic Recalculation of PageRank

To examine the capability of the test application to accommodate dynamic addition of further pages, the distributed PageRank values of a set of documents from the Guardian and CNN websites were obtained after they had been inserted into the DHT in a variety of different ways. The results are shown in graph 6.6, and the series represent the following situations:

1. the values obtained when all pages were inserted into the DHT prior to

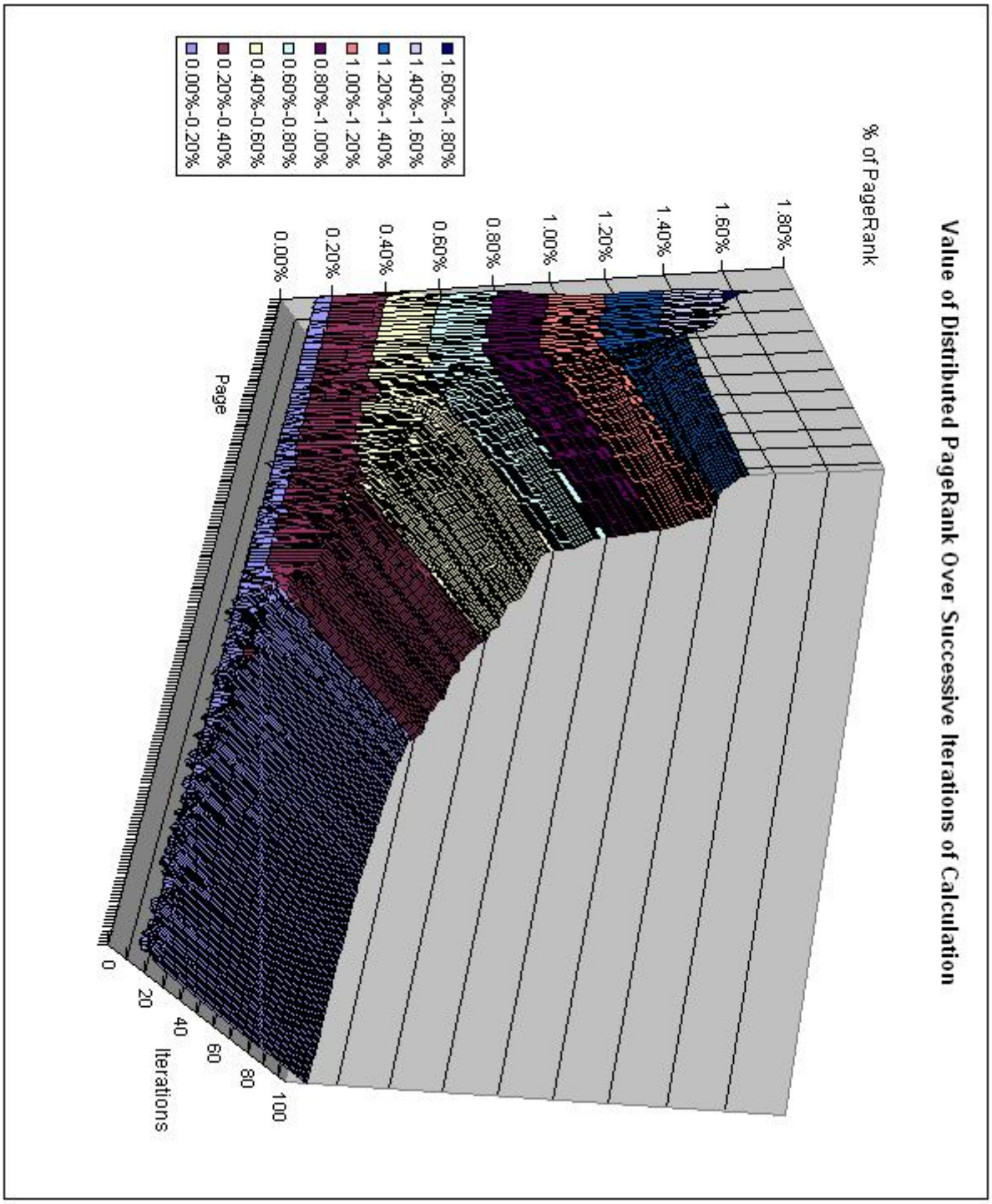


Figure 6.5: Graph displaying the PageRank rating of the top 200 ranked pages out of a sample of 726 pages over successive iterations of the distributed PageRank algorithm

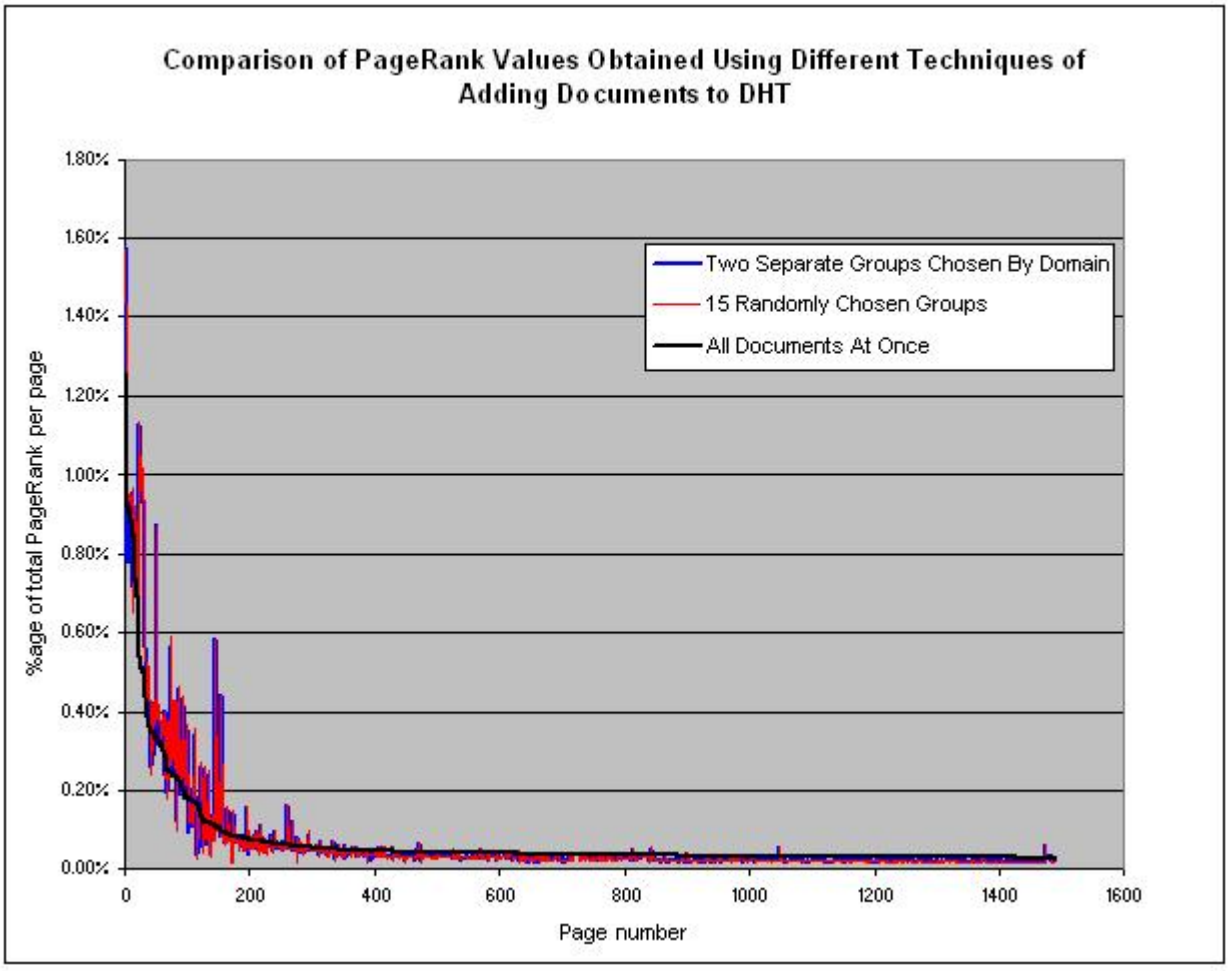


Figure 6.6: Graph displaying the PageRank ratings of 1,439 documents taken from the CNN and Guardian websites.

initiation of the calculation. Results shown are the mean of five separate runs.

2. the values obtained when pages were inserted into the DHT in two batches corresponding to the domains of the each site. The calculation was started after the first batch had been added to the DHT, and continued running while the second batch was inserted until all nodes reported PageRank had converged for all documents they held. Results shown are the mean of five separate runs.

3. the values obtained when pages were inserted in fifteen batches of randomly selected pages. The calculation was started after the first batch had been inserted into the DHT, and continued while the other batches were added until all nodes reported PageRank had converged for all documents they held. Results shown are for a single attempt.

Examination of the results shows a degree of correspondence between the ratings achieved when documents are drip-fed into the DHT versus those achieved when they are all added at the same time, although the amount of differentiation between documents in the middle region between very popular and unpopular looks somewhat erratic. It would be of benefit to continue the study using a larger collection of documents, to determine if this volatility evens out, and to investigate whether any further parameter tuning could reduce any fluctuations.

6.4 Other Performance Considerations

Robustness in the face of dynamic entry and departure of nodes is a design goal of the Chord protocol, as discussed by Stoica et al [8, 7, 9], and their results indicate that this goal is effectively satisfied. Notably, it seems that the neighbourhood data ensures that, even when 20% of the nodes present in a network fail, the network does not degenerate into partitioned subsets but quickly reconfigures itself using its stabilisation protocol. A quantitative assessment of PageRank calculation under changing network membership was not performed, but under inspection the system appeared to continue functioning and generating updated PageRank results if nodes were added and removed while the calculation was running.

N.B. the key storage classes within Naanou contain some caching functionality. This does not officially form part of the Chord protocol, but rather belongs in a separate application layer. The functionality was retained in the test application as it did not impact on any of the results reported, and appeared to preserve key-value data that would otherwise have been lost on node failure. There are a variety of reasons why some form of caching might be useful for a distributed document

ranking application, e.g. to support the production of a document indexing service, or for load-balancing of popular content. A purpose-built facility might well look different to that contained in Naanou, but discussion of the design of this functionality is left for further work.

Chapter 7

Further Work

7.1 Document indexing

A PageRank value on its own is of limited value in providing a search service. For the service to be useful it would be necessary to add an inverted index to the DHT to allow for keyword querying. This would involve:

- Parsing each document for words to generate a dictionary (and cross-referencing against a stop-list of words too frequently used to be useful).
- Storing the position of each occurrence of the word in the inverted index.
- Providing a mechanism for comparing the results of searches for individual words to determine the intersection when performing multiple word searches.

Reynolds [35] suggests some techniques for tackling this. A key problem with this sort of operation in a distributed environment is minimising the amount of network messaging generated. Careful consideration needs to be given to the organisation of data storage. Reynolds proposes two alternatives here:

1. Horizontal storage, whereby keywords are stored along with the document on which they are located, and individual words will therefore probably have occurrences spread across multiple nodes. This reduces messaging overhead at the point of distributing the dictionary, as there is no requirement to communicate with other nodes, however it potentially makes a search very time-consuming as it would be necessary to query every node in the network to determine if any of its documents contained the sought terms. This latter problem means that horizontal storage is unlikely to be a desirable technique.

2. Vertical storage, whereby specific keywords are allocated to specific nodes (based on their hash value). Using this mechanism, once the document has been parsed to generate a dictionary, it would be necessary to perform lookup operations on the words found to determine which node in the DHT should be responsible for them, and then to send an indexing message to each of those nodes advising them of the word found and the locations of its occurrences. High-end estimates of the number of words in the English language suggest around 400,000 possible different words (including other languages, phrases, mis-spellings etc. takes this number into the millions, although the number in common use would be more likely of the order of around 10,000). Possible efficiencies to be realised here would be to parse all documents held on the node at the same time, thereby eliminating the need to lookup nodes for the same word more than once, and pooling indexing messages for specific nodes, thereby removing the need to send more than one message to the same place. The initialisation requirements for vertical index storage are much higher than for horizontal storage, but the benefit shows in query responsiveness, as it is only necessary to query the nodes responsible for the hashes of the search terms sought.

Reynolds [35] suggests the use of Bloom filters to accelerate the performance of multiple-word queries. These are a technique for determining the intersection of sets efficiently, and would reduce the amount of network traffic generated from sending intermediate results for documents containing individual terms in the multiple-term search.

Problems arise with the sequence in which indexing and ranking occur. We would like to use our PageRank value to sort results of queries, so only the top n ranked items are returned, and hence it would be useful to have calculated it before indexing occurs and to store a copy of it alongside the document positioning in the index. However, the PageRank value is not static in a distributed environment. We also do not want to be in a position whereby we know the position of all occurrences of a word in our table, and then have to query each of those occurrences to determine what their PageRank values are in order to determine which of the occurrences (of which there could be arbitrarily many) to return to the client. So, to summarise potential choices here:

- Calculate PageRank to convergence, then generate the inverted index and include the PageRank value with document position information. This method is insensitive to subsequent rank change (of the sort that we should expect from, for example, peers joining or leaving the network, thereby removing or adding documents and changing the web graph structure) unless we send new messages to all indexed words every time the rank changes.

- Do not include the PageRank value with document index position information. Query the index entries for their PageRank once the initial list of search results has been generated. This is potentially time-consuming as the list of results could be arbitrarily long.

- Employ a compromise whereby the inverted index contains an *indicative* PageRank based on the document's ranking at the time it was added to the index, which could possibly be refreshed by a message from the document when the PageRank changed by a certain percentage threshold or when a certain time interval had elapsed. Estimates place the average number of different words in a web page at 300. The amount of communication necessary to notify an inverted index of changes in document PageRank would be a function of this essentially constant average word count, the number of documents present in the index and any lookup cost associated with determining the location of the document. If we follow the same principle used in sending PageRank-related messages, and route the index-update information from its source to its destination using information contained in the sources local routing tables rather than performing lookups to determine the exact location of each word, the communication costs can be kept manageable.

It may be the case that some advantage could be gained by attempting to minimise perturbation to the web graph structure, by caching document details within the DHT. It is conceivable that this would smooth out any short-term changes in linkage between documents caused by peers becoming temporarily offline, thereby reducing any requirement to recalculate the PageRank and communicate the new value with the indexing service.

To evaluate the relative merits of these techniques thoroughly it would be necessary to conduct a study of how they performed under a set of conditions representative of those found in a typical peer-to-peer network.

7.2 Modifications to the Rank Calculation

In our implementation we have used the basic PageRank measure proposed by Shi et al [22], which depends solely on the link structure of the web graph. It would be worthwhile to consider including further inputs to this measure to modify the ranking being communicated between peers. Areas worthy of investigation include features such as analysis link text contents and other markup-related document information; source authority or some other form of topic-based weighting; and convergence acceleration techniques derived by using alternative calculation techniques to the basic power method.

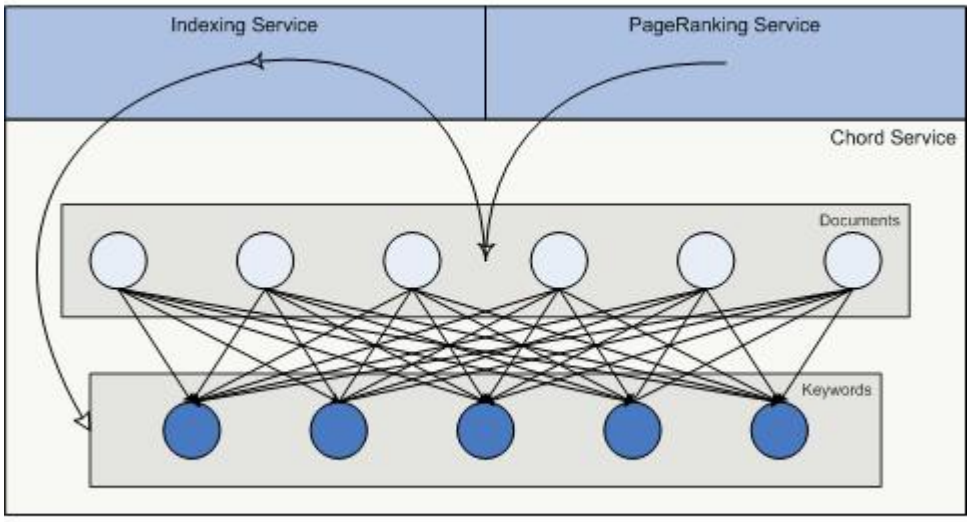


Figure 7.1: Illustration of possible communication flows between a document ranking service, an indexing service and a DHT lookup service in a distributed search engine

7.3 Varying Methods of Distributing Data

Our study of the generation of PageRank values over a Chord-based network has necessarily been conducted using comparatively modest sizes of dataset, with web graphs selected predominantly from a single domain. Use of larger samples of data would be beneficial not only to verify scalability of the calculation, but would also open up the possibility of attempting alternative analysis techniques. The tests conducted for this report have used a single hash value for each document, and have thus distributed the contents of particular domains around various nodes within the network. Given a larger dataset, it would be feasible to construct a system whereby storage in the DHT could be made at the domain level rather than at the document level. This would be useful because:

- It would more closely model the situation where documents with the same domain name are located on the same physical host.
- It would open up the prospect of investigating some of the properties observed in web graphs [33], whereby pages tend to have a concentration of links to other pages on the same host. This is clearly of relevance to the calculation of a PageRank value as it allows for a substantial proportion of the iterative convergence to be performed with reference to pages stored on the same node without any need for network communication.

Potential issues to consider here would be reductions in load balancing efficiency, especially where domains with popular content or large numbers of pages are concerned. This could perhaps be mitigated by using caching or replication of such pages, and by breaking up larger bodies of documents into subsets.

The application we have developed to illustrate the performance of the distributed ranking service has been built such that it uses a subset of data contained in the document (i.e. the link structure) to perform the calculation, and that the actual contents of the document will be retrieved from its regular web host. A possible extension of the notion that keys in our peer-to-peer system might contain the contents of entire domains rather than individual pages is that web servers could act as PageRanking nodes for domains that they themselves host. This would neatly solve the potential load-balancing issue for busy sites; assuming that search traffic is in proportion to the popularity or size of the site, a bigger site would have more powerful servers and therefore be better able to cope with higher volumes of search traffic.

Chapter 8

Conclusion

The results of the experiments conducted suggest promising potential for a version of PageRank calculated in a peer-to-peer environment. The PageRank values generated by the distributed algorithm showed a level of correlation with those of the centralised calculation that indicates the assessment of relevance is sound, while the properties of being able incorporate additional documents and generate relevance ratings for them that approximate closely to their final ranking in a small number of iterations are desirable features not available from the centralised method. The Chord platform underpinning the calculation has demonstrable properties of robustness, and communication capabilities that scale well with increasing peer membership, so while the size of the network and document repository the tests were conducted on was relatively small the elements are in place to suggest it should be possible to extend it to substantially larger groups of documents.

A few applications suggest themselves. Single-site search services, such as those used on a corporate intranet or a university network, are likely to contain sufficient content to warrant a search facility capable of something more sophisticated than a basic keyword-match. The hosts of the content would make a sensible location to place the ranking service; no crawling would be required as each host would be able to consult the documents that it itself hosted to obtain link details. It may be possible to include some form of topic-based personalisation in such a service to further enhance the relevance of results.

One of the applications suggested for DHTs is that of distributed cooperative storage [8]. If such a system were used to store documents that made reference to other documents (e.g. academic reports), a distributed ranking system could be used as part of a search function to help locate the most highly regarded documents.

On a larger scale, one may speculate about the possibility of a distributed full-scale search engine, although this would be a far from trivial endeavour. Se-

curity obviously would be a serious concern; for example great care would need to be taken to prevent the introduction of rogue citation messages from malicious nodes. Distributed ranking would need to be coupled with distributed crawling; intuitively, crawling seems like a function well-suited to the peer-to-peer environment, although clearly it is a task that needs to be approached with sensitivity. Ingenuity would be required to persuade users to contribute resources to such a facility, as search services are universally taken for granted as being available from any browser. The promise of search results containing bang-up-to-date documents tailored to the user-specific interests might just provide compelling enough reason for people to participate.

Bibliography

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," tech. rep., Stanford University, 1998.

[2] L. Page and S. Brin, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," tech. rep., Stanford University, 1998.

[3] S. Brin, R. Motwani, L. Page, and T. Winograd, "What can you do with a web in your pocket?," 1998.

[4] "Gnutella." http://www.gnutella.com.

[5] "Gnutella developers forum." http://groups.yahoo.com/group/the_gdf.

[6] "Napster." http://www.napster.co.uk/.

[7] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 ACMSIG COMM Conference*, pp. 149–160, 2001.

[8] F.Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area Cooperative Storage with CFS," tech. rep., MIT Laboratory for Computer Science, 2001.

[9] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building peer-to-peer systems with chord, a distributed lookup service," tech. rep., MIT, 2001.

[10] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, November 2001.

[11] S. P. Ratnasamy, *A Scalable Content-Addressable Network*. PhD thesis, University of California at Berkeley, 2002.

[12] "Google zeitgeist." http://www.google.com/press/zeitgeist2003.html.

[13] "Google watch." http://www.google-watch.org.

[14] "Is google broken?." http://www.google-watch.org/broken.html.

[15] B. Cohen, "Incentives Build Robustness in BitTorrent." http://bitconjurer.org/BitTorrent/bittorrentecon.pdf, May 2003.

[16] "The free network project." http://freenet.sourceforge.net.

[17] C. Heyer, "Naanou." http://chdesign.cjb.net/, 2003.

[18] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica, "Querying the Internet with PIER," 2003.

[19] C. W. O'Donnell and V. Vaikuntanathan, "Information leak in the chord lookup protocol," tech. rep., MIT Computer Science and Artificial Intelligence Laboratory, 2004.

[20] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in dht-based systems," tech. rep., Carnegie Mellon University, 2004.

[21] E. Andersson and P.-A. Ekstr¨m, "Investigating google's pagerank algorithm," tech. rep., Uppsala University, 2004.

[22] S. Shi, J. Yu, G. Yang, and D. Wang, "Distributed page ranking in structured p2p networks," in *Proceedings of the 2003 International Conference on Parallel Processing*, 2003.

[23] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne, "Distributed pagerank for p2p systems," tech. rep., The University of Texas at Austin, 2003.

[24] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra Applications*, 1969.

[25] J. C. Strikwerda, "A convergence theorem for chaotic asynchronous relaxation," tech. rep., University of Wisconsin-Madison, 1996.

[26] "Google bomb." http://en.wikipedia.org/wiki/Google_bomb.

[27] "Google information for webmasters." http://www.google.com/webmasters/seo.html.

[28] T. H. Haveliwala, "Topic-sensitive pagerank," tech. rep., Stanford University, 2002.

[29] "Open directory project." http://dmoz.org.

[30] G. Jeh and J. Widom, "Scaling personalised web search," tech. rep., Stanford University, 2002.

[31] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub, "Extrapolation Methods for Accelerating PageRank Computations," in *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[32] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "The web as a graph," tech. rep., IBM Almaden Research Center, 2000.

[33] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, "Exploiting the block structure of the web for computing pagerank," tech. rep., Stanford University, 2003.

[34] "Jspider." http://j-spider.sourceforge.net/.

[35] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," tech. rep., Duke University, 2002.

Code Used for the Centralised Calculation

```
Matrix transportationMatrix = new Matrix(transportationVector);
Matrix prMatrix = new Matrix(pr);

Matrix prMatrixNext = prMatrix;
int iterations = 0;

do
{
    prMatrix = prMatrixNext;
    prMatrixNext = linkMatrix.multiply(prMatrix);
    prMatrixNext = prMatrixNext.scalarMultiply(DAMPINGFACTOR);
    prMatrixNext = prMatrixNext.add(transportationMatrix);
    iterations++;
}
    while(!prMatrixNext.converged(prMatrix,CONVERGENCE_RATIO));
```

Figure 1: Code used in calculating centralised PageRank value

Code Used for the Distributed Calculation

```
public double getNewRank()
{
    double rank = 0;
    double newRank = 0;
    double transporter = 1 - DAMPINGFACTOR;
    double newCitations = 0;

    System.Collections.Hashtable citations = doc.NewCitations;
    if (additions.Count > 0)
    {
        System.Collections.ICollection incomingCitations = citations.Values;
        System.Collections.ArrayList arr =
                new System.Collections.ArrayList(incomingCitations);

        foreach (object o in arr)
        {
            double citation = (double)o;
            newCitations += citation  ;
        }
    }
    do
    {
        rank = newRank;
        newRank = (DAMPINGFACTOR * rank) + transporter + newCitations;
    } while (Math.Abs((newRank - rank)/rank)> CONVERGENCE_RATIO);

    return newRank;
}
```

Figure 2: Code used in calculating distributed PageRank value