ON THE DESIGN OF RELIABLE EFFICIENT INFORMATION SYSTEMS

BY

ABDUR CHOWDHURY

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Advisor

Chicago, Illinois
May 2001

ACKNOWLEDGMENT

I would like to thank the many people who have supported and helped me with this work. First, I would like to thank my wife Ana who provided me the strength to keep going. My parents who instilled the joy of learning and taught me there is always a way. Dr. Ophir Frieder for being the best friend and advisor that one could hope for, without whose advice and support none of this would have been possible.

I would also like to thank Dr. David Grossman for the uncountable help he has provided me over the years. Dr. Peng-Jun Wan for all his help with the probability models. Dr. Catherine McCabe for her help in testing, validating and countless other things that came up through the years. A special thanks to Robert Ellis who believed in me when things were tough. Lastly to all of my friends that have put up with my lack of availability.

aut viam inveniam aut faciam
"I'll either find a way or make one"

ARC

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

This thesis presents computer algorithms for the design of reliable information server clusters. As information systems grow in size the need for reliability, parallelization and speed grow. Provided within are algorithms that address reliability and efficiency. A novel algorithm called DRS (Dynamic Routing System) is presented. The DRS algorithm sustains continuous availability of clusters of information or compute servers even during network failures. We continue with a new duplicate data detection algorithm that is several times faster than the state of the art and more precise. By detecting duplicate data, redundant work is eliminated from indexing and retrieval processing. Additionally higher retrieval accuracy is provided by reducing the amount of redundant information returned to users. Finally, automatic term weighting utilities are examined; results are presented showing that as dynamic collections grow, automatic term weights do not need to be recalculated and still maintain system effectiveness.

CHAPTER I

INTRODUCTION - RELIABLE INFORMATION SYSTEMS

With the ever-increasing amount of information, one of the greatest challenges of the twenty-first century is the organization and retrieval of that information. Currently, the WWW (World Wide Web) has 1-2 billion publicly accessible pages of information [60][*]. Two to five billion pages are estimated to be available via private networks and hidden content that is dynamically generated from databases. Available information has grown at exponential rates for the last few years [60] and no slow down appears to coming soon [1]. This growth of information and the access to it provides ample incentive to develop new algorithms and approaches to organize and retrieve that information.

As greater amounts of information are being stored, indexed and retrieved via a single system, scalability issues are becoming greater and greater problems. Single CPU solutions are not able to keep up with the exponential information growth even with growing CPU power. Parallelization of storage and indexing are current research problems [2, 3] along with distributed retrieval algorithms providing the parallelism for this growing workload [4, 5, 6, 7]. In the words of Grace Hopper:

---

[*] Corresponding to references in the Bibliography

> In pioneer days, they used oxen for heavy pulling, and when one ox couldn't budge a log they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers.

In this thesis, a generalized software architecture for a reliable information system is presented. This thesis addresses essential issues of that architecture namely, network fault tolerance for a distributed information system, efficiency of the system in terms of duplicate data and lastly speed improvements by delaying automatic term weight calculations in a dynamic system.

In the next section, we present a generalized distributed information retrieval architecture, which our new algorithms will support. In Chapter II, we present a distributed fault tolerant routing solution for distributed information server clusters that guarantees server-to-server communication in the event of a network failure. In Chapter III, we then present a duplicate information detection system that uses collection statistics to efficiently determine when duplicate information is inserted into the system. In Chapter IV, automatic term weighting strategies are examined and experimental results show that reduced term weight updates can be applied to dynamic systems without loss of overall effectiveness to the system.

1.1 Generalized Information Retrieval Architecture

In this section, we present a generalized information retrieval architecture. For this architecture, we will make several assumptions:

1. Data/Information are growing at a faster rate than compute power.
2. Parallelism of systems is the most promising solution to the growing needs of retrieval systems.
3. Data/Information are coming from many sources.
4. Any algorithm that helps efficiency or effectiveness is beneficial.

Given the above assumptions, new retrieval systems will be parallel / distributed in nature. Data will be stored on a number of systems, indexing of that data will be done with multiple machines and CPUs. Retrieval of information will be distributed to multiple machines and CPUs. New large-scale information systems will be closely coupled creating a large server array controlled by a single entity. These distributed and parallel information systems will solve complex information needs via many computers. In the next section, we present our contributions in the context of such a distributed large-scale system.

1.2 Thesis Reliability and Effectiveness Issues Addressed

This thesis is comprised of three parts. The first part presents a proactive routing algorithm for distributed systems, the second presents a duplicate data detection algorithm and the third part presents empirical evidence that shows automatic term weight calculations can be delayed in order to reduce maintenance overhead and still provide effective rankings for large dynamic retrieval systems.

As client needs grow, server systems have become more complicated and require additional compute needs. While many techniques were applied to improve the performance of a single machine, these improvements are insufficient to keep up with the growth. To combat the additional demands for computational resources, server systems are distributed among multiple computers to handle the additional computational requirements [8, 9, 10, 11]. As information systems get larger, they too are distributed to provide the additional processing needs. This distributed approach of dividing the problem into either multiple workers for the same problem or multiple workers working

on different client problems all depend on network communication. Since a network connecting the servers is the backbone of the computational server, it must be reliable. We provide data that show that network failures constitute 13% of all hardware failures and are a problem for large systems, thus validating the need for reliable network communication in the presence of network failure. As part of the described effort a proactive network routing protocol that reroutes network communication around failures is presented. This protocol and topology provides a fault tolerant network communications system for server clusters in the presence of failures. This type of system provides the reliability that new Information Retrieval (IR) systems need as they become distributed [46, 12, 13]. We present a quantitative probability model showing the advantages of our dynamic routing approach for tightly coupled server clusters. We also describe the advantages in performance in using a proactive approach as opposed to a reactive routing approach when dealing with server clusters. We show that additional overhead incurred by our approach does not adversely affect network performance and does greatly improve the reliability by 267% over traditional topologies of a distributed server cluster. We validate our hypothesis of network reliability improving the reliability of distributed server clusters with two probability models.

As information retrieval systems collect data, the likelihood of multiple copies of the same data, or near duplicate documents being added to the system, increases [59]. We hypothesize that if the duplicate information is detected and eliminated in a fast efficient manner, the system accuracy and performance is enhanced. We present a duplicate document algorithm based on collection statistics of terms in a given collection. This algorithm selectively chooses what terms represent a document by using term

collection statistics. Once the document's subset of terms is selected, a hash is created of those terms. With a single value representing each document, a collection can be scanned in O(log d) time where d is the number of documents in the collection. We show that this algorithm performs five to nine times faster than comparable algorithms. We also show that the use of this system clusters near duplicate documents better than the other state of the art algorithms. Lastly, we show that by ignoring the detection and removal of duplicates, results are effected, by misjudgments of relevant documents. Furthermore, the same document is presented to the user multiple times forcing end users to filter the same information when looking for new information.

There are many efficiency issues for information retrieval systems. Most of the prior work focuses on efficient data representation to improve the efficiency of the system [14]. We hypothesize that the efficient update of collection statistics improves the overall efficiency of dynamic IR systems by delaying work until necessary. Very little work was applied to the efficiency issues of collection statistics. While collection statistics are used to improve precision and recall for information systems, very little work has gone into efficient update approaches to maintain their currency [15, 16, 17, 18, 19, 74, 77].

One type of common collection statistics used is Inverse Document Frequency (IDF). We hypothesize that the recalculation of idf values for each new document added is not necessary [20]. We present empirical data that show that the recalculation of IDF values, after an initial training set is used, does not improve the overall precision and recall of the information retrieval system. By reducing the time to recalculate IDF values, the overall performance of the system can be improved. Unnecessary additional work, in a dynamic environment, adversely affects the performance. By finding a good training set

of documents, those IDF values may be used by the system for an extended period without having to recompute IDF values for each additional document added to the system. These fundamental issues for reliability and efficiency for information retrieval systems are addressed in this thesis.

CHAPTER II

DISTRIBUTED SYSTEM NETWORK FAULT TOLERANCE

The ever-increasing demands on server applications are resulting in many new server services being implemented using a distributed server cluster architecture where many servers act together providing end user services. Twenty-seven such server clusters were evaluated each containing four to eight servers for a one-year period; thirteen percent of the hardware failures were network related. To provide end-user services, the server clusters must guarantee server-to-server communication in the presence of these network failures. In this thesis, a novel proactive routing algorithm called the Dynamic Routing System (DRS) is presented. The DRS continuously searches for failures via frequent ICMP echo requests. This approach differs from its predecessors in that it is proactive instead of reactive. That is, we continuously search for failures before they affect server-to-server communication. When a failure is detected, an alternative route is identified and used.

With growing compute needs, traditional supercomputers are becoming scarce and distributed compute server clusters are becoming the solution of choice. These smaller computers are coupled by networks to achieve the same objective at a substantially lower cost. The Berkley NOW (Network Of Workstations) project was one of the first projects pushing this solution [21]. PVM (Parallel Virtual Machine) [22] and MPI (Message Passing Interface) [23] libraries provide messaging and synchronization constructs that are needed for distributed parallel computing with NOW solutions. Projects like Beowulf [24] for Linux are continuing the distributed / parallel

approach. Research operating systems, like Spring from Sun [25], focus on distributed computing via the network. All of these approaches have one thing in common, the use of a network as a communication media. While the network is very important, relatively little attention has been focused on providing fault tolerance and redundancy for the network of workstations. Additionally, little work has addressed the issues involved in providing time constraints on detecting and resolving network errors for higher-level applications.

As part of this dissertation, we developed a network routing algorithm to provide fault-tolerance to networks of servers by proactively monitoring network communication links between servers. This is different from reactive routing techniques [26, 27, 28, 29] that wait for a failure to occur and then react by finding an alternative route. The proactive algorithm constantly looks for errors via continuous ICMP echo requests. When a failure is identified, a new route bypassing the failed portion of the network is selected. This new route is often found in the time for a TCP retransmit, so server applications are unaware that a network failure has occurred.

The Dynamic Routing System (DRS) is built on top of existing hardware and a variety of operating systems (Solaris, SunOS and LynxOS), making its use and deployment economical. This algorithm improves reliability via two network interface cards per server to provide an alternate method of physical communications in the case of hardware failure. The DRS works by frequent link checks between all pairs of nodes to determine if the link between pairs of computers is valid. This algorithm uses redundant network links between two nodes to provide multiple communication channels. When

one link fails, the second direct link is checked and used if possible. However, if no link exists, a broadcast is sent to identify whether or not some other node is able to act as a router to create a new path between the sender and the proposed recipient. The algorithm discovers the failure before application performance is affected. The essential goal of our algorithm is to hide network failures from distributed applications.

Based on deployed commercial implementations, we developed an analytical model of the DRS to evaluate its potential use for large networks. Using this model, we computed, for various network sizes, the fault identification times given a percentage of network usage. For a typical 10Mb ethernet, a sixteen-host network has sub-second fault identification when using 10 percent of the total theoretical packet throughput of an ethernet network. Sixteen hosts may seem small given that large corporations often have tens of thousands of workstations all on various LANs. However, the application domain of this solution is distributed server applications running on a separate network. A characteristic of Distributed Server Applications (DSA) is a tightly coupled server host array where clients exist apart from the server network, and the server array appears as a single server handling distributed data and requests. A word processor running off a file server is **not** a DSA.

A Network Survivability Analysis (NSA) of the DRS algorithm shows that the DRS algorithm provides a more resilient solution to network failures than a single network, and a simple dual network solution [30]. Results from the probability model, showing the probability of success of the system as a whole and in terms of the number of network failures are presented. In addition, results from a simulation validating the

probability model are presented.

2.1 Prior Work

There is an abundance of literature on routing algorithms and protocols. We partitioned the prior art into routing algorithms, routing daemons, hardware solutions, and network survivability analysis. We will review each category separately.

One of the most common routing solutions today is the Routing Information Protocol (RIP) [31, 32]. Its popularity stems from the fact that RIP is included in most versions of UNIX. RIP is a dynamic routing protocol that automatically creates and maintains network routes. Although popular, RIP has many shortcomings. One of the major problems of RIP is its reactive nature. When a link has not been heard from for a predetermined amount of time it is considered down and an alternative route is sought. This down time can be in the minutes or longer range, thus disrupting server-to-server communication. Another problem with RIP1 is its inability to work with subnets. RIP2 now can work with subnets but most implementations are still RIP1 [33].

Open Shortest Path First (OSPF) [34, 35, 36, 37] is a routing protocol for IP networks based on the DARPA Internet Protocol (IP) network layer. The basic routing algorithm is called the Shortest Path First Algorithm. OSPF is an Interior Gateway Protocol and is intended to be used within an IP network under common administration, such as a campus, corporate, or regional network. The OSPF approach is a passive approach. Therefore, an OSPF routing daemon does not know that a problem has occurred until a time-out value has been reached before a new route is sought out. OSPF

is not designed or intended for server cluster routing.

The External Gateway Protocol (EGP) [38, 39, 40, 41, 42, 43, 44], sometimes referred to as Border Gateway Protocol (BGP), are network to network routing protocols as opposed to host to host routing protocols. These protocols are used in constructing Wide Area Networks, however they do not provide fault tolerance to small server network clusters [45].

While RIP, OSPF, EGP and BGP are routing solutions to many different routing problems; they do not address the needs of a high availability server cluster environment. Their primary goal is to provide routing updates to other routers on the network to find alternative routes to the same network [45]. The general design goal is based on reactively rerouting when a specified timeout period has been reached. Therefore, if a destination network does not respond to a route query, after some time quantum, it is considered down and a new route is sought after. The DRS algorithm is proactive in that each node of a server cluster is constantly monitored to maintain a communications link. If that link does not work, a redundant route is sought after in a distributed manner [46, 47].

Routed and gated are routing daemons that implement some subset of RIP, OSPF, EGP, and BGP. Given the algorithms that they are based on, however, none of these approaches provides a proactive fault detection schema to protect distributed applications from network failures. Hence, system down time is potentially greater.

For many years the telecommunications industry has been interested in fault

tolerance for their networks of systems. Telecommunications protocols are inherently different from routing protocols in that they focus predominantly on hardware solutions. The most widely used solutions are SONET and DXC [28, 48]. Others include double-loop, forward hop, and fiber. Survivable network architectures for traffic restoration are generally divided into two categories; ring-based dedicated restoration and mesh restoration [47]. Rings with redundant capacity and automatic protection switching are capable of healing by themselves and hence, are called Self-Healing Rings (SHR) [49]. Mesh restoration relies on digital cross-connect systems to reroute traffic around a point of failure [50].

A conventional DXC self-healing network using logical channel protection requires substantial network hardware because for n hosts [29] there are two physical connections among each pair of hosts, or n*(n-1) total connections. This is a large amount of spare capacity for network components. Originally, self-healing meshes used a centralized database to track failures and reconfigure in case of a failure for the entire mesh. This centralization was a bottleneck and was itself prone to failure. Hence, a distributed approach is now used [51]. With a distributed approach, each host determines rerouting patterns and fault detection [52, 53].

A variety of hardware solutions are used by the telecommunications industry to route phone calls. These include SONET, DXC, FDDI, and SHR. Each of these is highly fault tolerant due to the numerous paths that exist between every source and destination host. Fault tolerant routing is provided via hardware mirroring rendering this approach very expensive. Such solutions are not commonly implemented for use with computer

networks using IP protocols. The DRS system works with IP networks unlike some telecommunication approaches using specialized hardware.

Network Survivability Analyses (NSA) [54] are developed to quantitatively evaluate different network topologies. NSA numbers increase with redundancy and decrease with series components. More redundancy with less hardware becomes the design objective instead of the use of redundancy to correct a reliability or survivability deficiency. The DRS system uses redundant hardware to provide alternative routes to network nodes. We provide a NSA type evaluation of the DRS providing several probability evaluation techniques for the DRS. With NSA analysis we quantify the DRS improvements to a compute server cluster system with different network topologies.

2.2 DRS Algorithm

The Dynamic Routing System (DRS) improves fault tolerance via proactive failure recognition and the use of a completely redundant network. In Figure 1, we illustrate a dual network setup for two computers. Each computer has two network interface cards connected to two separate networks. It is the task of the routing daemons to monitor the connections between host A and host B. If a failure occurs, the daemons set up routes to route around the fault before network applications are aware that a problem occurred.

The DRS runs on every host in the server array. Each DRS daemon is configured to monitor every server host on each network and executes a two-phase processing strategy. In the first phase, the communications links between the local host and all other

hosts that it is configured to monitor are checked. These checks are accomplished using the Internet Control Message Protocol (ICMP) echo request [55]. Host A sends an ICMP echo request to host B via the first network. If the echo is returned, the DRS can assume that the hub, wiring, network interface card, device driver, network protocol stack, host kernel, and the DRS daemon are all operational. The DRS then tests all known hosts and all known networks in the above example. The complete DRS network stack is illustrated in Figure 2. Similarly, the second network from host A to host B is checked.

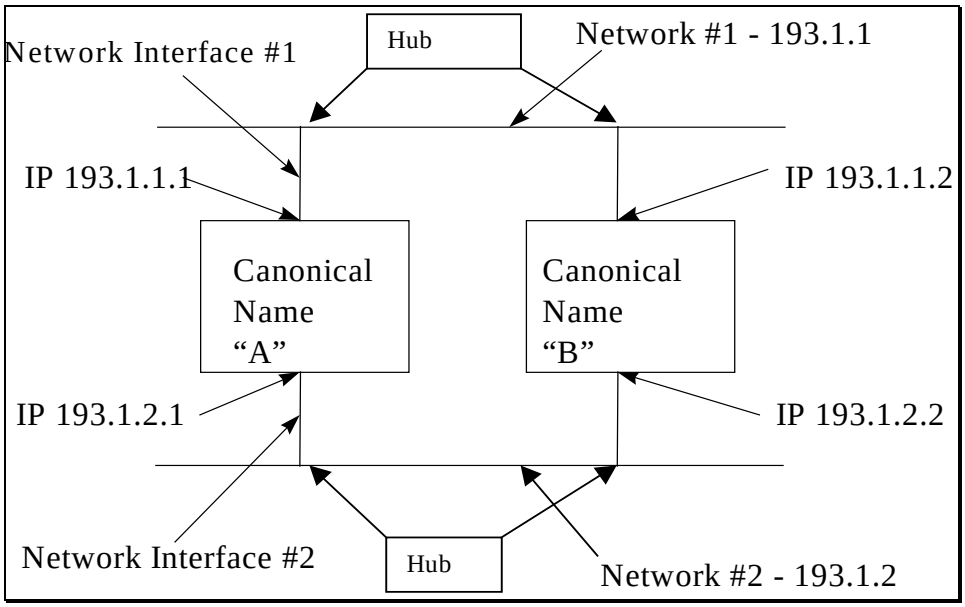


Figure 1. Dual Network Setup Details

Each daemon keeps track of which hosts to monitor and the state that they are in (i.e., "up", "down"). If a failure occurs, the DRS daemon must determine a new route of communication between hosts A and B. The next section describes different failure scenarios and how the DRS attempts to establish a new route. In Figure 3, we overview

the execution flow of the DRS algorithm.

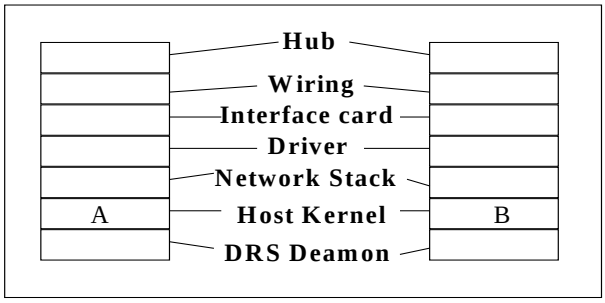


Figure 2. DRS Network Stack

A detailed description of each step in the DRS execution is as follows.

Step 1. System initialization. All DRS system variables are initialized, i.e., read in configuration variables, setup network communication modules, etc.

Step 2. Initial sleep period. Each DRS is dormant at startup. The dormant initial condition prevents false negative results of a ping at startup. Periodically, a system could be powered down during routine maintenance. When the servers are restarted, not all may start at the same time or boot at the same speed. By having the DRS daemon sleep for a predetermined amount of time at startup, false failures caused by start-up time differences are not reported.

Step 3. Monitor incoming requests or discovery messages. The DRS is a routing daemon. Thus, part of its job is to handle requests for information or to add new information to its internal database of network hosts and configurations. A “request for

information" can be an administrator contacting the daemon and requesting a view of its routing tables, or a remote server that is unable to contact another server and is asking all other servers if they are able to communicate to the server in question. "Discovery" messages are used for detection of fixed routes and new servers on the network. This is covered in detail in step 6.

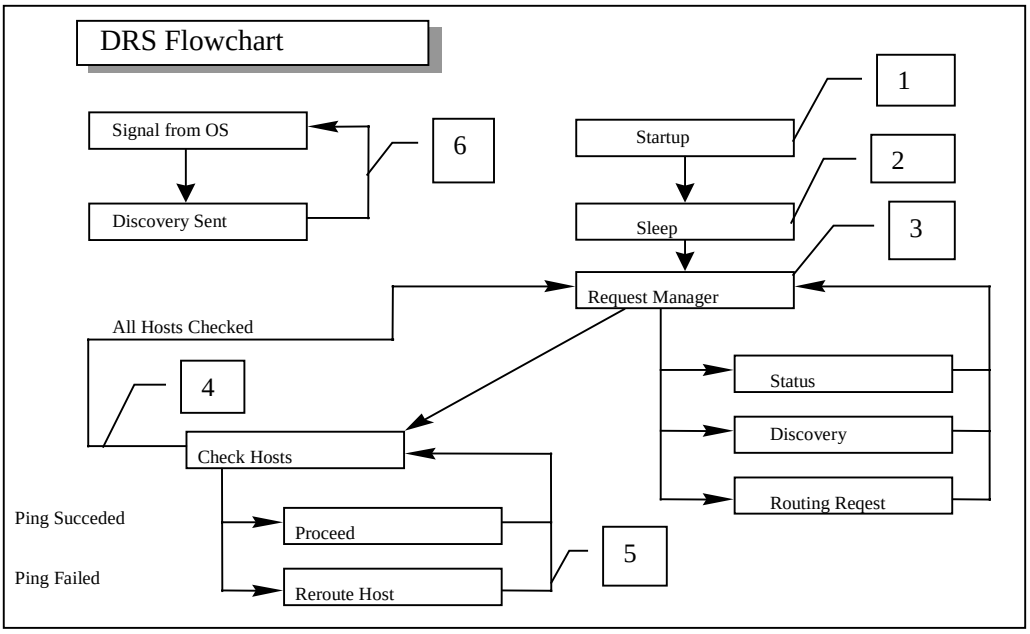


Figure 3. DRS Architecture

Step 4. Determine link status. Link status verification, the key to the DRS, is the proactive monitoring of communication links between each server. This verification enables the DRS to quickly find and fix network failures. The DRS starts with a list of hosts to monitor. This list is known at start time, but may also be added to in the future by a "Discovery" message. The DRS sends an ICMP echo request to the host in question. If the echo is successful, the route is marked as "up"; if it is unsuccessful, several more attempts are made. If none are successful, the route is marked as "down". All

links are continuously monitored. Checks occur every X seconds where X is a configurable setting. Note that X affects the speed an error is detected and effects the amount of network bandwidth used.

Step 5. Fix identified communication errors. In this step, the DRS attempts to fix known "down" routes. Each host has two network interfaces. Once one interface is not responding, the second interface is sent an ICMP echo request to verify that it is working. If that is successful, the DRS modifies its internal routing tables to move all communication to say B network interface #1 to B network interface #2. Note that in step 4, the new route was checked. The second check guarantees that a failure did not occur in-between steps. If the second interface did not respond to the ICMP request, a broadcast is sent on all connected networks. This broadcast asks all other DRS daemons to see if they can communicate with the host or network in question. The first DRS daemon to respond is used as a router to the lost host. If no one responds, the host in question has suffered at least two hardware failures and has become completely separated from the network. If this has happened, the only thing left to do is to send and alarm message to the system administrator notifying him/her of a catastrophic failure.

Step 6. Send "Discovery" messages. This stage runs as a separate thread of execution in the daemon. The DRS sends a broadcast message on all of its network interfaces stating its own server identification and its server's network interfaces addresses. This message is crucial for several reasons, the most significant being that if a network failure did occur and was fixed, the DRS would otherwise not be updated of the fixed status because "down" routes are not checked. By sending this message, the other

DRS daemons become aware that a “down” interface is now working again, and the daemon corrects all rerouted communications of that host to the original routes.

The DRS loops through this six-step cycle monitoring communication links, answering requests, and fixing problems as they occur.

2.3 DRS in the Presence of Network Failures

The DRS handles many different failure situations. We now briefly describe several common failure situations and the solutions the DRS algorithm will compute. Network failures can be categorized into three scenarios:

- *Single network failure*
- *Multiple network failures*
- *Complete network separation failures*

Initially, we focus on the action of the DRS in the presence of a single failure. Upon startup (before the network error occurs), the DRS establishes communication links to each host. Consider a failure to host B interface #1. Since every host on the network is implementing the same algorithm, we only discuss the events as they happen for host A.

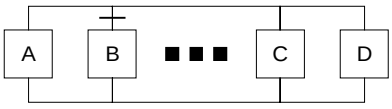


Figure 4. Single Network Failure

Host A sends an ICMP echo request for each host and link in its routing table

(part of step 4). The ICMP echo request is not responded to by host B for network #1 because of the failure. Host A then looks for another route. Note that one does exist because the second interface of host B is operational. Host A now identifies that this is a potential route because it is listed as being in the “up” status. However, this status is not guaranteed to be current so an additional check of the proposed alternative route is made in the later phase of step number four. In this case, an ICMP echo request (or ping) is sent to host B along the alternative route. If this succeeds, the routing table of host A is updated to reflect the newly identified static route that circumvents the failure.

At this point, network communication is not impeded by the failure. However, it is important to identify where the failure exists so it can be repaired. Examining the routing tables of each host isolates the fault. Looking at the local routing at hosts A, C, and D it becomes apparent that each host of these three hosts is able to communicate with all other hosts directly on network #1. However, the routing tables of hosts A, C, and D routing tables now contains an alternative route to host B. For diagnostic purposes, it is reasonable to assume that the routing tables are current enough to isolate the problem.

For hosts A, C, and D, the only failure is the route to B for interface #1, while host B has failure for every interface on the first network. This indicates that the error has occurred with the host’s network interface, ethernet wire, or network hub port of host B. At this point, host B is examined and the exact nature of the problem (i.e., interface card, network cable, hub port, etc.) is determined and repaired.

A single failure is the most common. However, multiple failures do occur, and the DRS is resilient to them. There are two types of multiple failures: multiple failures

that can be viewed as a single fault by the DRS and those failures that require additional processing beyond single failure masking.

When multiple faults simultaneously occur at a **single** host (i.e., within the interface card, network cable, hub port, etc., of a single host) they are treated by the DRS algorithm as a single failure. That is, these failures are handled in the same fashion and appear like the single failure example above. All network communication for the first network is rerouted to the second network via each host's second interface.

Multiple network failures, although unlikely, do occur and are not always as well behaved. The likelihood of multiple simultaneous faults occurring on only either the primary or the secondary network is smaller than the possibility of them spanning over both networks. Thus, the DRS must be able to handle staggered network failures. Figure 5 shows an example of a staggered multiple network failure. A port on the network hub one to host A has failed, and, at the same time, the network interface card for host C has failed.

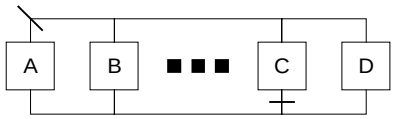


Figure 5. Multiple Network Failure

The problem is that host A and host C cannot directly communicate to each other even with multiple communication networks. Host A cannot communicate via network

#1 and host C cannot communicate via network #2.

For brevity, we only discuss the details of host A and its procedure in correcting the network communication from two simultaneous failures. The same algorithm executes simultaneously at each host.

In step 4, each host's communication link that is in an "up" state is checked. Every host that is on network #1 fails because the problem is with the hub. Host A is then placed in the "down" state. Host C's interface on network #2 also failed and is placed in the "down" state. This reflects the new information that shows many communication paths have failed. At this point, step 5 executes. Host B and host D have direct routes (using interface 2) that appear to be usable. This corrects the communication link failure on network #1 from host A to host B and from host A to host D.

Notice that still there is no means of communicating from host A to host C. The reason for this is that the interface card on network #2 for host C is identified as failed for both interface one and interface two (this is our second failure).

Host A now attempts to find some means of communicating with host C on interface #1. It broadcasts a routing request along both network #1 and network #2. The first "CanYouRoute" broadcast is blocked by the failure on host A interface #1. The second "CanYouRoute" broadcast is sent out as a routing request for host C on the second network. The first host to respond to the plea for help is used as a router for communication to host C. Assume host B is the first host to respond for communications

routing for host C interface #1. A static route-using host B is added to host A's routing table. This scenario is illustrated in Figure 6.

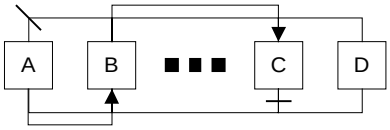


Figure 6. Communication Route Diagram for Host A to Host C via Host B

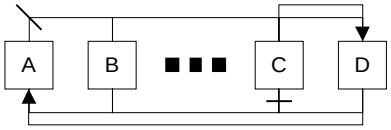


Figure 7. Communication Route Diagram for Host A to Host C via Host D

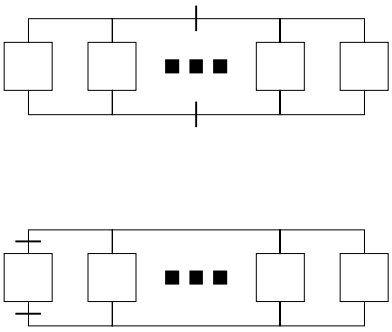


Figure 8. Complete Network Separation

Since two routes must be known for each host, host A attempts to find a route for host C interface #2. The DRS does not distinguish that the different interfaces are connected to the same host in this instance. Again, a broadcast is issued on both

networks. The first broadcast goes unacknowledged. Assume host D answers the second request for help in routing to host C. (See Figure 7). Now all communication routes to host C are restored using a remote host as routers between the hosts.

The final failure scenario is a complete network separation or node failure. A complete node failure cannot be solved by any routing solution. The survivability analysis and routing solution assume that the sender and receiver are working. A complete network separation is a failure scenario where both network interfaces for a single node fail, thus isolating the node from the system. The probability of these cases is addressed in the probability model.

2.4 Detection of Network Repairs

The DRS has the ability to detect the reconnection or repair of a failed network route. In the previous example, assume that the cause of failure for host A was that the port on hub one was accidentally turned off. Furthermore, assume the problem was detected, circumvented, and the DRS daemon sent an alarm.

The system administrators resolve the problem by turning the port back on. Once this is done, they do not need to examine the routing tables of the hosts because the DRS is self-correcting. That is, the DRS uses discovery messages to identify the correction and return the routing tables to their original state (i.e., direct network routes instead of static alternative routes).

The self-correcting recovery occurs because each host periodically broadcasts its own discovery message on all of its network interfaces. When host A receives discovery

messages on interface #1 from hosts B, C, and D, it examines the routing table to determine if any corrections or updates are needed. The DRS identifies that static alternate routes exist for hosts B, C, and D. After receiving the discovery message, the DRS removes the corresponding alternate route entry from the local kernels routing table. This removal restores the original routing table state after a failure has been fixed. The DRS routing table is then updated to reflect the newly repaired communication path.

Notice that host C interface #2 is still down. Host A has not received a status discovery message from that link because the network link is still not functioning correctly. With this approach, when the network failures are corrected, the DRS returns to its original state without manual intervention.

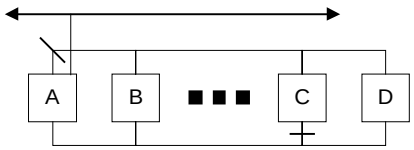


Figure 9. Restored Network Communication

2.5 DRS Performance Results

The DRS checks each “up” link it is attached to (Step 4). Therefore, on average, the fastest failures in a communication link can be determined is the amount of time it takes to check all links divided by two. When scaling the DRS, it is important to examine the time needed to determine that a failure has occurred and the amount of bandwidth the DRS will use to achieve that goal. A trade-off exists between how quickly a failure is discovered and how much network overhead (bandwidth) is introduced.

2.6 DRS Model System

The DRS's proactive monitoring of network links comes at a cost of network bandwidth. To find errors before they effect network communication, the links must be checked frequently. If the links are not checked frequently, the DRS is equivalent to a reactive routing protocol. As the number of nodes increases, the bandwidth required to support the frequent checks likewise increases. In Figure 10 and Figure 11, we present the number of servers in the cluster that the DRS can support given a requirement for error resolution in X time units. Additionally, the percentage of network bandwidth useable by the DRS can be used to determine the speed at which errors are detected. As shown in Figure 11, ninety hosts are supported in less than 1 second with only 10% of the bandwidth usage.

Each ICMP echo request is 64 bytes in length. As the number of nodes increases the number of checks required to maintain link connectivity status increases. Thus, for each node there are 2(n-1) messages and a total of 2n(n-1) messages for the system. For a given number of nodes and a frequency rate of checks the amount of bandwidth used can be calculated. In Figure 10 and Figure 11, we demonstrated that relationship. The production machines did not see any degradation in performance from the added network usage.

As shown in Figure 10, as the number of hosts increases the rate of network monitoring must decrease to maintain a constant network usage. The results presented in Figure 10 and Figure 11 were obtained using the DRS model. A comparison of actual performance predicted by the simulation is presented in Figure 12.

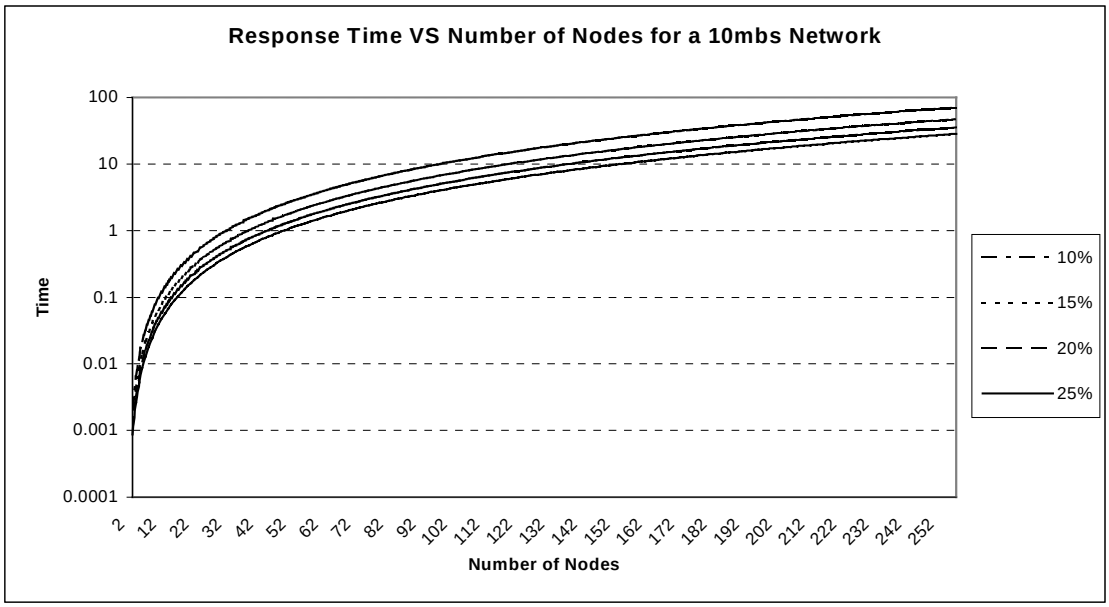


Figure 10. 10Mb Network Performance

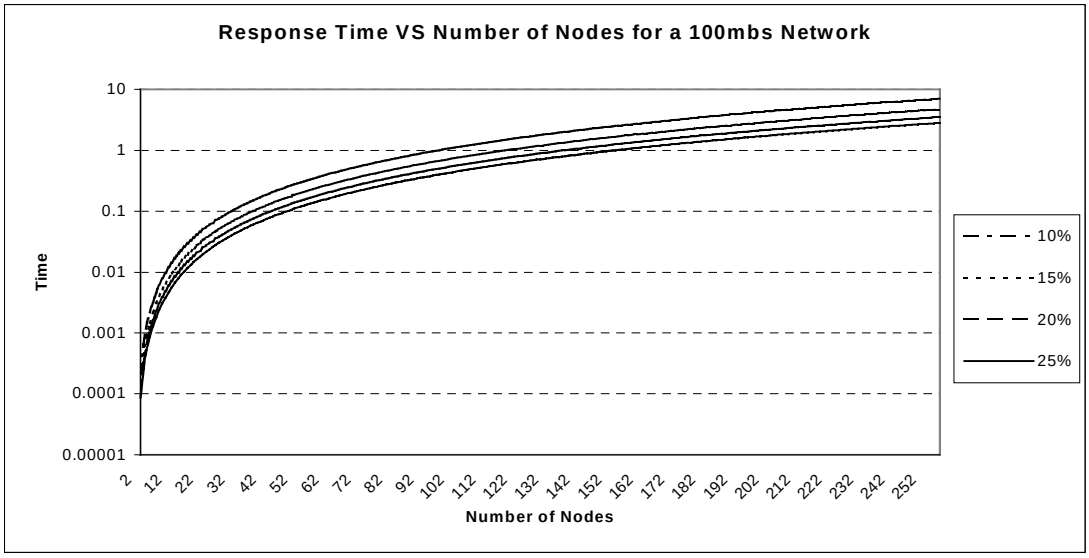


Figure 11. 100Mb Network Performance

Figure 10 shows that on a 10Mb network, 32 servers can coexist while still being able to detect network failures in around one second. If there is a need for more servers,

either a higher network utilization factor is used or a higher bandwidth network is required. In Figure 11, the DRS performance using a 100Mb-ethernet network is shown.

By using a 100Mb network, the performance of the DRS greatly improves. As shown in Figure 11, 254 hosts can be monitored by the DRS with less than 10% network usage, and failures are detected in less than 10 seconds. Ninety hosts are able to detect an error in less than 1 second. The number 254 was chosen because it is the size of a class "C" network.

2.7 DRS Network Measurements

We evaluated the performance of the DRS system and compared the expected network to actual network usage. This was done to (a) find an acceptable level of performance for the system that allowed for sub-second error detection and (b) find a level of network usage that did not adversely affect the other applications performance.

In Figure 10 and Figure 11, we show that as the number of hosts increases, the time between checks decreases to maintain a constant network usage. We tested this hypothesis by transferring files across the network and calculating the percentage of total bandwidth achieved.

We ran this experiment with no network traffic and ran the DRS system on 2, 8, and 16 hosts. We recalculated the available bandwidth at each step. As the number of hosts increased, we reduced the time to check links to maintain a constant network usage. The results given in Figure 12 show that the simulation and actual results correspond. Note that 10% usage is usable by a network of less than 16 hosts and provides a less than

one-second-error detection rate.

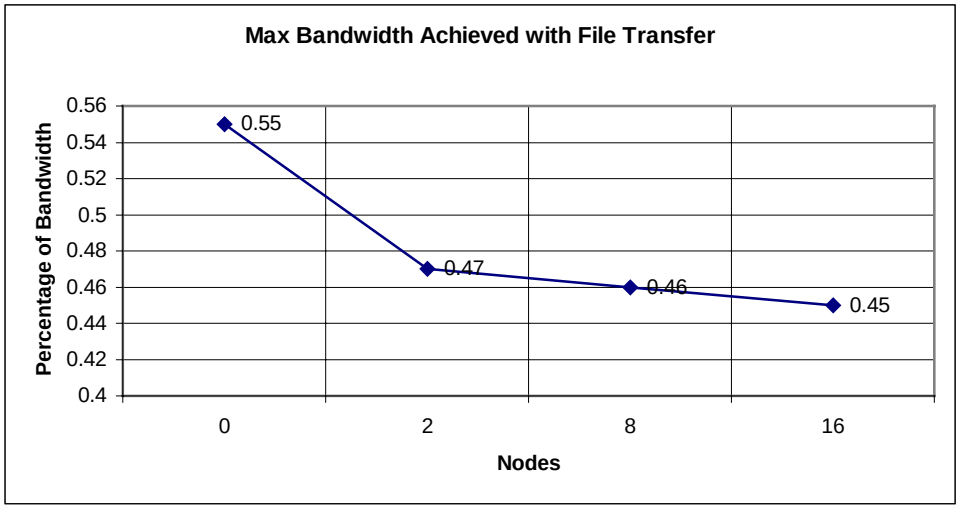


Figure 12. Bandwidth Usage

2.8 Hardware Failure Rates

The DRS was implemented and developed in a commercial system by MCI Worldcom. The primary goal of DRS was to provide fault tolerance for network hardware failures. The commercial system does not have any system administrators on premises. In fact, the closest repair engineer is at least one day away. Because of the mission critical nature of the system, downtime of one day is very expensive.

Over a 12-month period, hardware failures were tracked to determine the usefulness of the DRS. Hardware failures were categorized into seven classes.

- *System Boards (mother boards, CPU chips, etc.)*
- *Mass Storage (SCSI controllers, hard drives, tape drives)*
- *Network hardware (hubs, ethernet cards, ethernet wires, etc.)*

- *Terminal Servers*
- *Accessories (specialized hardware)*
- *Memory (RAM)*
- *Power Supplies*

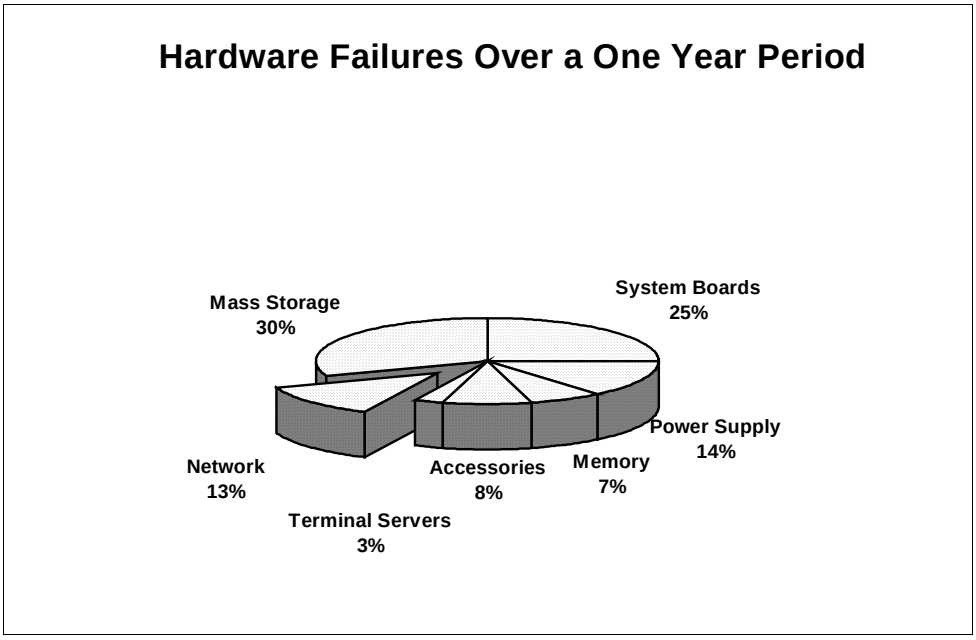


Figure 13. Percentage of Failures by Type

Examining 100 systems, we experienced 70 hardware failures over a 12-month period. Of these failures, 13% were network related.

By using the DRS, network failure did not affect system performance, effectively eliminating this class of failures. A 13% reduction of system down time justifies the added cost of hardware and network bandwidth. It is reasonable to expect the same amount of hardware failures in the future.

2.9 Dynamic Routing Survivability Probability Models

In the prior sections, we described the DRS algorithm, and examined its solution to network failure scenarios. We now present two probability models to quantitatively compare the DRS algorithm to single network approaches, and dual-network approaches. The first model gives a success probability based upon the unconditional failure probability for the system as a whole. The second model provides the probability of a successful connection between any two nodes given a system with N nodes and f network failures.

The following analysis will describe the probability of system failure based upon the success and failure probabilities of each individual component. We assign *p* as the probability that a component will function properly, and *q* as the probability that a component will fail, with *p+q=1*. The formula can be extended trivially if the components have distinct failure probabilities.

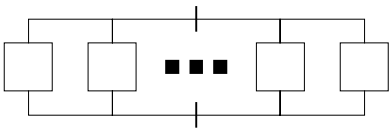


Figure 14. Case 1 - Both Backplanes

Case 1: Both backplanes fail (q^2). In this case, the system fails and the probability is q^2.

Case 2: Exactly one backplane fails (2pq). In this case, the system fails if and only if at least one of the two interfaces of the pairs connecting to the working backplane

fails. So the probability is *2*p*q* (1-p*p) =2pq(1- p^2).*

Case 3: Neither of the backplanes fails (p^2).

Case 3.1: Both interfaces of the source node fail (q^2). So the failure probability is $p^2 * q^2 = p^2 q^2$.

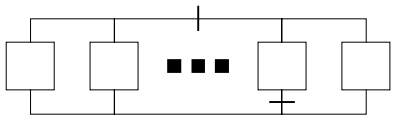


Figure 15. Case 2 - Hub and NIC Failure

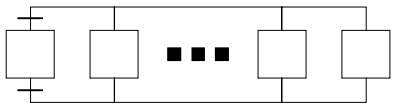


Figure 16. Case 3.1, 3.3 - Both NIC

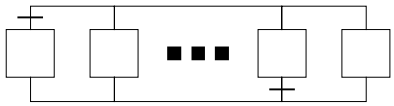


Figure 17. Case 3.2

Case 3.2: Exactly one interface of the source node fails (2pq). In this case, the interface of the destination node at the same side of the working interface of the source node must be down (q).

Case 3.2.1: The other interface of the destination node fails (q). So the failure probability is $p^2*2pq*q*q= 2p^3q^3$.

Case 3.2.2*: The other interface of the destination node works (p). In this case, all other N-2 bridges must be down (1- p^2). Thus the failure probability is $p^2*2pq*q*p*(1-p^2)^{N-2}= 2p^4q^2(1-p^2)^{N-2}$.*

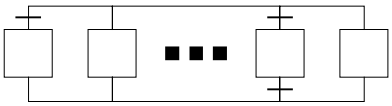


Figure 18. Case 3.2.1

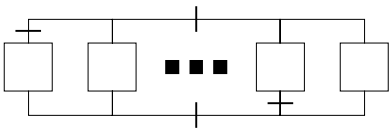


Figure 19. Case 3.2.2

*So the total failure probability in **Case 3.2** is $2p^3q^3+ 2p^4q^2(1-p^2)^{N-2}$.*

Case 3.3: Neither of the interfaces of the source node fails (p^2). (See Figure 16). In this case, both interfaces of the destination node must fail (q^2). So the failure probability is $p^2*p^2*q^2= p^4q^2$.

Therefore, the total failure probability in Case 3 is $p^2q^2+ 2p^3q^3+ 2p^4q^2(1-p^2)^{N-2}+ p^4q^2$. So the total failure probability is $q^2+2pq(1-p^2)+ p^2q^2+ 2p^3q^3+ p^4q^2+ 2p^4q^2(1-p^2)^{N-2}$. Therefore, the probability of success is:

$$P[Success] = 1 - [q^2 + 2pq(1 - p^2) + p^2q^2 + 2p^3q^3 + p^4q^2 + 2p^4q^2(1 - p^2)^{N-2}].$$

Equation 1. DRS Unconditional Failure Probability

To compare results, we also provide equations for a dual network system and a single network system. Using the same methods, the probability of success of a dual network can be written:

$$P[Success] = 1 - [q^2 + 2pq(1 - p^2) + p^2q^2 + 2p^3q^2 + p^4q^2].$$

Equation 2. Dual Network Unconditional Failure Probability

Likewise, the probability of success for a single network system can be written:

$$P[Success] = 1 - [q + pq + p^2q].$$

Equation 3. Single Network Unconditional Failure Probability

The dual and single networks are independent of N because they do not have the re-routing algorithm, while the DRS equation will approach a specific probability as $N \rightarrow \infty$.

In the prior section, we reported about “in the field” deployment and usage statistics. Given the actual usage data, q = 0.13 and p = 0.87. Using Equation 1. DRS Unconditional Failure Probability, we calculate the unconditional failure probability of a given system. Given a server cluster of 20 and a probability of failure of 13% the DRS system yields, an unconditional failure probability 37% greater than a single network topology for 20 clustered server nodes.

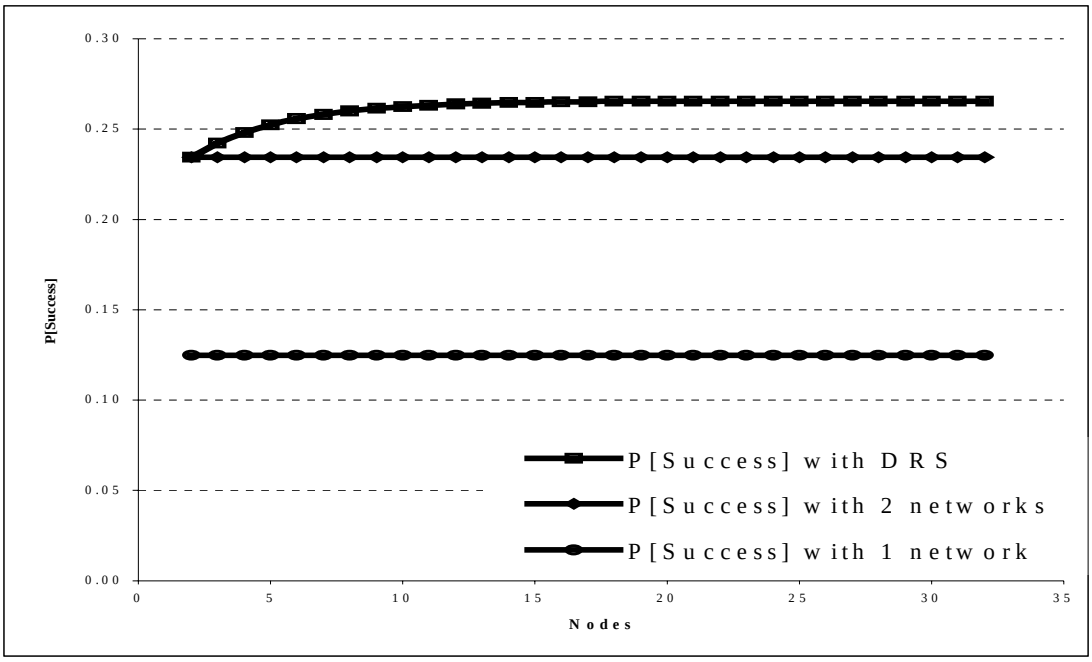


Figure 20. Unconditional Failure Probability

If no prior data are available we can assign an equal probability of success and failure to each node, i.e. *p=q=0.5* and evaluate the probability of unconditional failure at any moment for our system. When evaluating the system's probability of success with Equation 1, the DRS for any given moment is 112% better than a single network topology

and 13% better than a dual-network topology. Figure 20 describes a comparison of the DRS versus dual and single network topologies presented in Equation 1, Equation 2 and Equation 3 for p = q = 0.5 and X nodes, respectively.

The unconditional failure probability gives us a total probability of success model for the entire system. We now present a quantitative probability model to evaluate systems in terms of a given number of network failures occurring at a given instance. In this model, we determine the probability of success, independent of time, of a system with N nodes and *f* failures. We assume that, in a system with N nodes, there are exactly 2N interface connections and two non-meshed back planes, each with equal probability of failure, say q, for $0 \leq q \leq 1$. Therefore, the probability of two failures in any system will be q^2, the probability of three failures will be q^3, and the probability of *f* failures will be q^f. It follows that $\lim_{f \to \infty} q^f = 0$. Consequently, the probability of multiple failures decreases exponentially.

As the number of nodes in a system increases, the probability that a system using the DRS maintains a successful connection between any two nodes at any given time will approach 1 for a fixed number of failures, using the DRS. Since there are 2N+2 total connections that the *f* failures can be distributed among, the total number of combinations of *f* failures in the system is $\binom{2N+2}{f}$. We now count the number of failure combinations that result in the failure of the communication between a specific pair of nodes.

Case 1: Both backplanes fail. In this case, the remaining f-2 failures appear in

the 2N components. The total number of such combinations is $\binom{2N}{f-2}$.

Case 2: Exactly one backplane fails. The total number of combinations of f-failure is $\binom{2N}{f-1}$. To make the specific pair unable to communicate, at least one of the two interfaces of the pairs connecting to the working backplane fails. Therefore, the number of failures that will not result in the failure of communication between the given specific pair of nodes is $\binom{2N-2}{f-1}$. The total number of such combinations is $\binom{2N}{f-1}-\binom{2N-2}{f-1}$.

There are two cases in which exactly one backplane fails, so the total number of combinations is $2\cdot\left[\binom{2N}{f-1}-\binom{2N-2}{f-1}\right]$.

Case 3: Neither of the backplanes fail.

Case 3.1: Both interfaces of the source node fail.

The total number of combinations is $\binom{2N-2}{f-2}$.

Case 3.2: Exactly one interface of the source node fails. In this case, the interface of the destination node at the same side of the working interface of the source node must be down.

Case 3.2.1: The other interface of the destination node fails. So the remaining f-3 failures appear in 2N-4 components. Thus the total number of combinations is $\binom{2N-4}{f-3}$.

Case 3.2.2: The other interface of the destination node works. In this case, all other N-2 bridges must be down. Therefore, the remaining f-2 failures appear in the N-2 bridges, and each bridge must contain at least one failure. Among them (f-2) mod (N-2)=f-N bridges contain two failures, and (N-2)-(f-N)=2N-f-2 bridges contain exactly one failure. There are $\binom{N-2}{f-N}$ choices of the bridges with both fail links. For each such choice, there are 2^{2N-f-2} configurations of the remaining single-failure bridges. Therefore, the total number of combinations is $\binom{N-2}{f-N}\cdot 2^{2N-f-2}$.

There are two cases in which exactly one interface of the source node fails so the total of combinations in **Case 3.2** is $2\cdot\left[\binom{2N-4}{f-3}+\binom{N-2}{f-N}\cdot 2^{2N-f-2}\right]$.

Case 3.3: Neither of the interfaces of the source node fails. In this case, both interfaces of the destination node must fail. Therefore, the total number of failures is $\binom{2N-4}{f-2}$.

The total number of failures in Case 3 is:

$$\binom{2N-2}{f-2}+2\cdot\left[\binom{2N-4}{f-3}+\binom{N-2}{f-N}\cdot 2^{2N-f-2}\right]+\binom{2N-4}{f-2}.$$

The total number of failure combinations is

$$F(N, f) = \binom{2N}{f-2} + 2 \cdot \left[\binom{2N}{f-1} - \binom{2N-2}{f-1} \right] + \binom{2N-2}{f-2} + 2 \cdot \left[\binom{2N-4}{f-3} + \binom{N-2}{f-N} \cdot 2^{2N-f-2} \right] + \binom{2N-4}{f-2},$$

and the probability of success for N nodes and f failures can be written:

$$P[\text{Success}] = \frac{\binom{2N+2}{f} - F(N, f)}{\binom{2N+2}{f}}$$

Equation 4. Probablity of Sucess

Using Equation 4, it is readily apparent that the probability of success converges to 1 as N gets large for fixed values of *f*. More specifically, for *f*=2 the P[S] surpasses 0.99 at 18 nodes. For *f*=3 the P[S] surpasses 0.99 at 32 nodes, and for f=4 the P[S] surpasses 0.99 at 45 nodes. Given that $\lim_{f \to \infty} q^f = 0$ and that $\lim_{N \to \infty} P[S] = 1$, a system implementing the DRS has a high probability of resilience to network failures.

2.10 DRS Simulation

To reinforce the validity of the probability models, we developed a computer simulation of a networking system. We model N nodes with f failures implementing the

DRS. The graph in Figure 21 the convergence of the simulation outputs to the actual equation values for 2 through 10 failures. The y-axis represents the mean absolute difference between the simulation output and the equation value for all values f<N<65. The x-axis represents the number of iterations in $\log_{10}$ scale. The simulation results support the probability model of Equation 4 presented earlier.

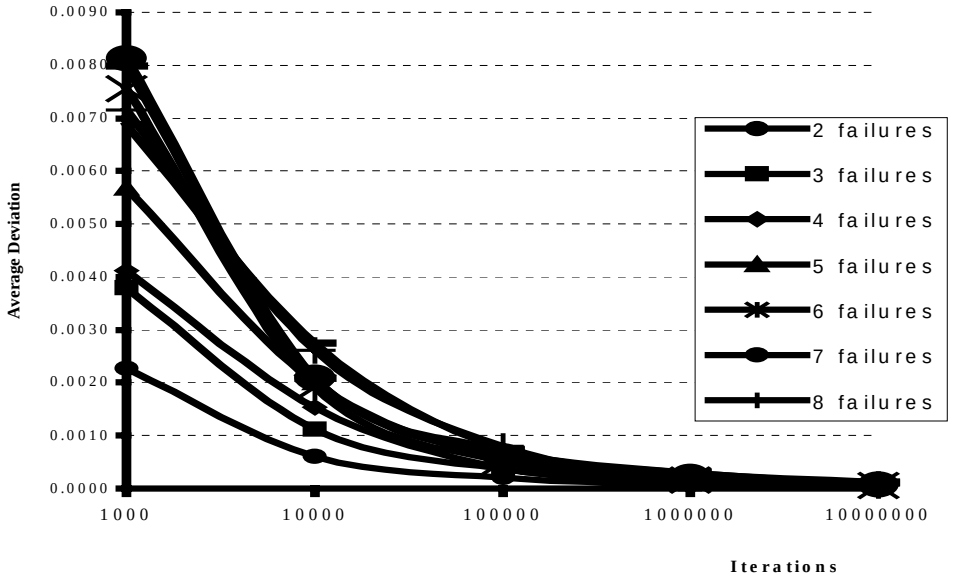


Figure 21. Convergence of Simulation Results to Equation Results

2.11 DRS Model Results

Previously, we provided two probability models of the DRS algorithm. The first was an unconditional failure probability of the entire system. We compared the DRS system to a dual network topology and a single network topology. We showed that the probability of success of the DRS system for any given moment is 112% better than a

single network topology and 13% better than a dual-network topology. We then provided a conditional failure probability model to evaluate the system in terms of number of network failures at a given instance. Those results were later validated via a computer simulation.

We now provide a quantitative comparison of the DRS to other solutions. The first type of solution is a simple one-network topology where all nodes are connected via a shared hub. The second solution is to add a second network to the system to provide a redundant communication path. A simple dual network solution will not work properly as implemented by most operating systems today without routing software; we do not address that issue. We assume that the network operates properly and use the second network if available, which allows the system to *reactively* re-route a connection if a network failure occurs. The third solution is to create a dual network topology and run the DRS on all nodes. With the probability model provided above we can evaluate each of the solutions in terms of probability of success, number of nodes, and *f* number of failures.

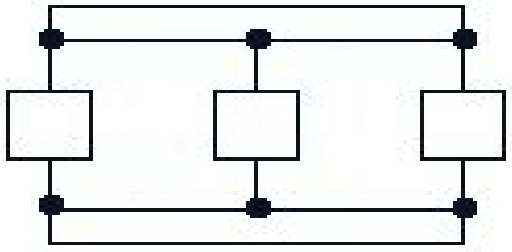
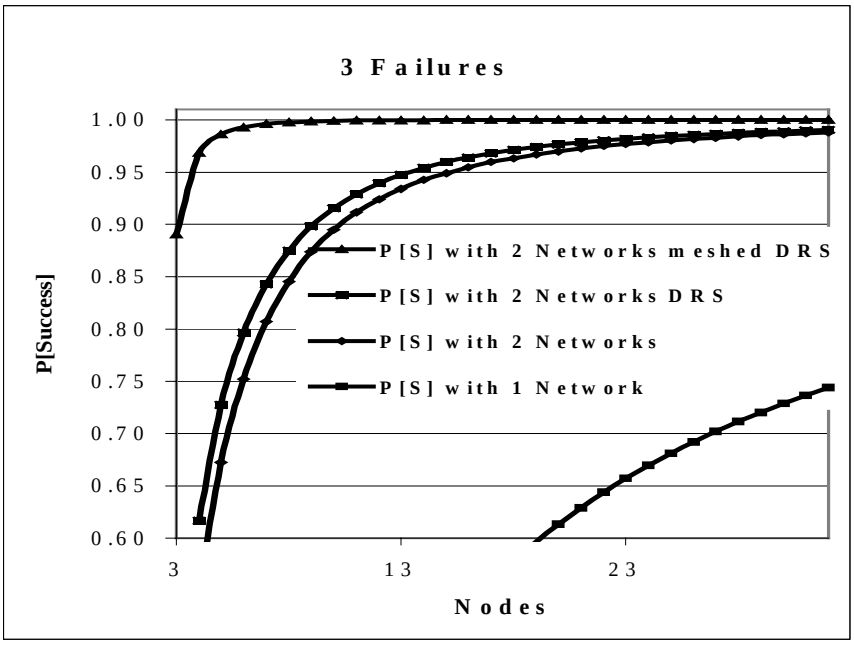


Figure 22. Dual Meshed Network with DRS Algorithm



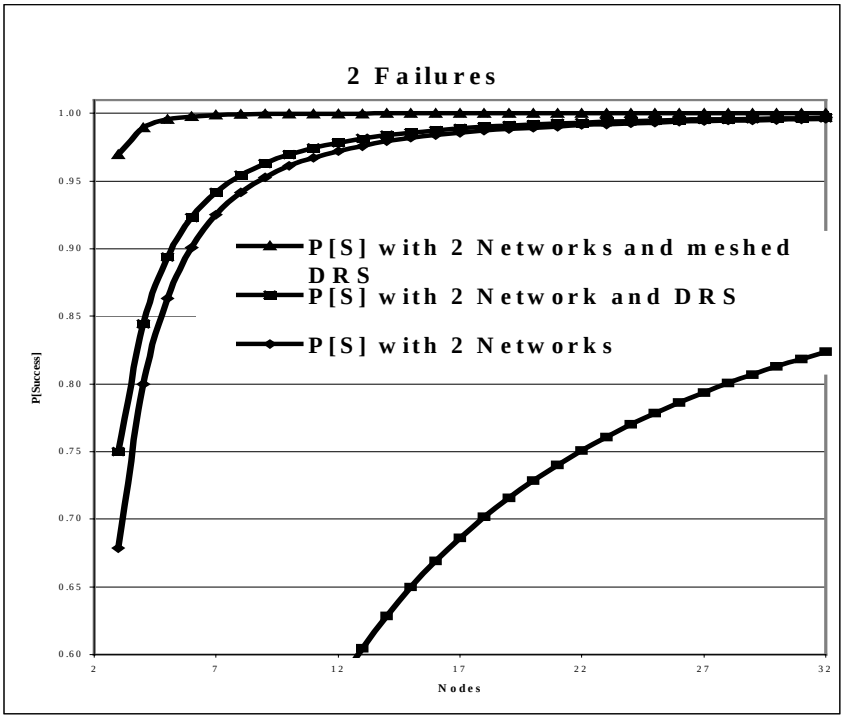


Figure 23. 2-3 Failure Comparison

In Figure 23 a comparison of four network topologies is shown for two and three

network failures. The first is a single network topology. The DRS for an eight-node cluster with three simultaneous failures is 267% more reliable than a single network topology 129% more reliable with only two network failures.

While re-routing without routing software does not work with most operating systems, the DRS only nominally improves on simple dual network topologies when comparing with a fixed number of failures. The reason is that DRS handling of opposite network failure situations is only a small percentage of the total number of network failure scenarios when the communication backplane is shared. Most switches use a fully connected mesh to create more communication channels to avoid channel contention. Therefore, the assumption that the communication link is one link is true for non-switched hub topologies where the hub uses a shared media for all port-to-port communications.

In Figure 23, we show an example of a fully connected mesh system. We provide a comparison of the current results to connected mesh architecture for two and three failures in Figure 24. Our existing models are a lower bound result for most topologies, which in reality are much better. DRS's ability to route around single channel failures in a backplane mesh makes this possible. Therefore, we leave these models out of the network probability model and use our results as a lower bound.

2.12 DRS Conclusions

The DRS algorithm was presented along with failure scenarios and solutions to network failure that the DRS would discover. Additionally, a one-year study was

conducted to determine the failure rates of various system components to demonstrate the benefits that a compute cluster would have by adding a redundant network. Two probability models were provided to quantitatively compare different network topologies and the benefit from the DRS approach. The first model gives an unconditional failure probability of the entire system. We compared the DRS system to a dual network topology and a single network topology. We showed that the DRS systems probability of success for any given moment is 112% better than a single network topology and 54% better than a dual-network topology. We also presented a second probability model to evaluate the system in terms of number of failures. Using Equation 4. Probablity of Sucess, we showed that the probability of success converges to 1 as N gets large for fixed values of *f*. More specifically, for *f*=2 the P[S] surpasses 0.99 at 18 nodes. For *f*=3 the P[S] surpasses 0.99 at 32 nodes, and for f=4 the P[S] surpasses 0.99 at 45 nodes. Given that $\lim_{f \to \infty} q^f = 0$ and that $\lim_{N \to \infty} P[S] = 1$, a system implementing the DRS has a high probability of resilience to network failure.

We also presented a network simulation of the system validating our probability model. We compared the DRS to single network topologies and showed that for an eight-node cluster with three simultaneous failures, the DRS is 267% more reliable than a single network topology and 129% more reliable with only two network failures. We also demonstrated that the DRS results are a lower bound result when compared to dual network topologies for a fixed number of failures and would provide greater resilience to network failures than other topologies.

The DRS algorithm is currently used by MCI Worldcom for enhanced voice services and is deployed to twenty-seven locations/clusters, each containing eight to 12 servers. While the DRS uses network bandwidth to pro-actively monitor network communication links, this added overhead has not effected systems overall performance. We presented this effort to quantitatively analyze the DRS's added benefits in comparison to other topologies or solutions, in terms of fault tolerance added to distributed- server clusters.

The DRS is a proactive routing protocol. It uses existing hardware and networking protocols to provide a fault tolerant network system for distributed applications and operating systems. The DRS, unlike routed and gated approaches, which passively monitor network links, proactively monitors each host and its communication links. The DRS also checks alternate routes before using them to achieve an additional level of fault tolerance without the use of special hardware. This fault tolerance comes at the price of some network bandwidth usage. We found this to be a reasonable trade off given that tightly coupled server arrays tend to be smaller than client server networks.

The production implementation runs on a four-host system and created no network problems. We have calculated that the same system can execute a 32 host server cluster and still achieve sub-second response time to network failures. The DRS is designed for the current trend of distributed computing systems. By using the DRS in a tightly clustered server, system remote clients are unaffected during a network failure. The future of this research will focus on the need for a more efficient means of checking a large number of servers, i.e., lower than n*(n-1) messages. In addition, we will explore

the use of RIP II in conjunction with DRS to add support for hosts outside of the network.

CHAPTER III

FAST DUPLICATE DOCUMENT DETECTION

We present a new algorithm for duplicate document detection that uses collection statistics. We compare our approach with the state-of-the-art approach using multiple collections. These collections include a 30MB 18,577 web document collection developed by Excite@Home and three NIST collections. The first NIST collection consists of 100MB 18,232 LA-Times documents roughly similar in the number of documents to the Excite@Home collection. The other two collections are both 2GB and are the 247,491-web document collection and the TREC disks 4 and 5 - 528,023 document collection. We show that the approach called I-Match, scales in terms of the number of documents and works well for documents of all sizes. We compared the solution to the state of the art and found that in addition to improved accuracy of detection, I-Match executed in roughly one-fifth the time.

Data portals are everywhere. The tremendous growth of the internet has spurred the existence of data portals for nearly every topic. Some of these portals are of general interest; some are highly domain specific. Independent of the focus, the vast majority of the portals obtain data, loosely called documents, from multiple sources. Obtaining data from multiple input sources typically results in duplication. The detection of duplicate documents within a collection has recently become an area of great interest [56, 57] and is the focus of our described effort.

Typically, inverted indexes are used to support efficient query processing in

information search and retrieval engines. Storing duplicate documents affects both the accuracy and efficiency of the search engine. Retrieving duplicate documents in response to a user's query clearly lowers the number of valid responses provided to the user, hence lowering the accuracy of the user's response set. Furthermore, processing duplicates necessitates additional computation without introducing any additional benefit. Hence, the processing efficiency of the user's query is likewise lowered.

A problem introduced by the indexing of duplicate documents is potentially skewed collection statistics. Collection statistics are often used as part of the similarity computation of a query to a document. Hence, the biasing of collection statistics may affect the overall precision of the entire system. Simply put, not only is a given user's performance compromised by the existence of duplicates, but also the overall retrieval accuracy of the engine is likewise jeopardized.

The definition of what constitutes a *duplicate* is unclear. For instance, a duplicate can be defined as the exact syntactic terms, without formatting differences. Throughout our efforts however, we adhere to the definition that duplicate document are two documents with a high percentage of overlapping text as previously defined as a measure of *resemblance* [57, 58]. The general notion is that if a document contains roughly the same semantic content it is a duplicate whether or not it is a precise syntactic match. When searching web documents, one might think that matching URL's would, at least, identify exact matches. However, many web sites use dynamic presentation wherein the content changes depending on the region or other variables. In addition, data providers often create several names for one site in an attempt to attract users with different

interests or perspectives. For instance, www.fox4.com, onsale.channel9.com, and www.realtv.com all point to an advertisement for real TV.

While the previous examples are for web documents, the same holds true for other collections where multiple document sources populate a single document collection. The National Center for Complimentary and Alternative Medicine (NCCAM), part of the National Institutes of Health [59], supports a search engine for medical data whose inputs come from multiple medical data sources. Given the nature of the data, duplicates are common. Since unique document identifiers are not possible across the different sources, the detection of duplicate information is essential in producing non-redundant results.

A previously proposed solution is the Digital Syntactic Clustering (DSC) algorithm and its Super Shingle (DSC-SS) variant [57]. While these algorithms are commonly used, they have efficiency problems. One reported run took ten CPU days to process a thirty million-document collection [57]. Additionally, DSC-SS and DSC are known to perform poorly on small documents. Given that the average size of a document on the web is around 4KB [60, 61], working with small documents is imperative.

The developed algorithm, called IIT-Match or I-Match for short filters documents based on term collection statistics. Results show that I-Match is five to six times faster than the DSC-SS algorithm. Furthermore, we show that I-Match does not ignore small documents and places each document into at most one duplicate set. Hence, I-Match increases accuracy and usability. Other approaches place potentially duplicate documents in multiple clusters. Hence, it is harder for a user to detect the actual duplicates. Finally, the sets of duplicates we detect are usually ‘tighter’ than DSC because we require an

"exact match" for the terms remaining after our filtration process. However, like other approaches, we still identify non-exact duplicates.

3.1 Prior Work

We partition prior work into three categories: shingling techniques, similarity measure calculations, and document images. Shingling techniques, such as COPS [62], KOALA [58], and DSC [57], take a set of contiguous terms or *shingles* of documents and compare the number of matching shingles. The comparison of document subsets allows the algorithms to calculate a percentage of overlap between two documents. This type of approach relies on hash values for each document subsection and filters those hash values to reduce the number of comparisons the algorithm must perform. The filtration, therefore, improves the runtime performance. Note that the simplest filter is strictly a syntactic filter based on simple syntactic rules, and the trivial subset is the entire collection. We illustrate later why such a naive approach is not generally acceptable. In the shingling approaches, rather than comparing documents, subdocuments are compared, and thus, each document may produce many potential duplicates. Returning many potential matches requires vast user involvement to sort out potential duplicates, diluting the potential usefulness of the approach.

To combat the inherent efficiency issues, several optimization techniques were proposed to reduce the number of comparisons made. By either removing frequently occurring shingles [58] or simply retaining only every 25^{th} shingle [57], the computation time is reduced. This reduction, however, does hinder the accuracy. Since no semantic premise is used to reduce the volume of data, a random degree of “fuzziness” is

introduced to the matching process resulting in relatively non-similar documents being identified as potential duplicates.

In terms of computational time complexity, these approaches are *O(kd log(kd))* where *k* is the number of shingles per document and *d* is the number of documents. Even with the performance-improving technique of removing shingles occurring in over 1000 documents and keeping only every 25^{th} shingle, the implementation of the DSC algorithm took 10 CPU days to process 30 million documents [57].

The DSC algorithm has a more efficient alternative, DSC-SS, which uses super shingles. This algorithm takes several shingles and combines them into a super shingle resulting in a document with a few super shingles rather than many shingles. Instead of measuring resemblance as a ratio of matching shingles, resemblance is defined as matching *a single super shingle* in two documents. This is much more efficient because it no longer requires a full counting of all overlaps. The runtime for this approach is still *O(kd log(kd))* but *k* is significantly smaller, and the amount of work to count overlap is eliminated, reducing the overall runtime. The authors, however, noted that DSC-SS does "not work well for short documents" so no runtime results are reported [57].

Approaches that compute document-to-document similarity measures [63] are similar to document clustering work [64] in that they use similarity computations to group potentially duplicate documents. All pairs of documents are compared. A document to document similarity comparison approach is thus computationally prohibitive given the $O(d^2)$ runtime, where *d* is the number of documents. In reality, these approaches use the document terms to search the collection. Therefore, for

document one, each term is used to search the collection, and a final weight is produced for each documents with a matching term, the highest valued document is the most similar. This approach of using the document as a query, thus clustering on those result sets, is computationally infeasible for large collections or dynamic collections since each document must be queried against the entire collection. Finally, approaches using document images are addressed in [65, 66]. These approaches map the duplicate detection problem into the image-processing domain rather than one in the document-processing arena.

3.2 Algorithm

The motivation for I-Match is to provide a duplicate detection algorithm that can scale to the size of the web and handle short documents typically seen on the web. Furthermore, we seek to place each document in only one set of potential duplicates. The degree of similarity supported should be sufficiently loose to identify non-exact matches but tight enough to insure that those true duplicates are defined.

In Figure 24, we illustrate the relative restrictiveness of different algorithms. DSC-SS is the loosest approach because it only requires one super shingle to match. Shingling is tighter because a percentage overlap in the remaining shingles is required. However, shingles and DSC-SS are very sensitive to adjustments in shingle size and thresholds. We have drawn a dotted line to indicate that these may be adjusted in such a way that shingling would be the less restrictive. Syntactic filters are the most restrictive because they leave most of the terms in the document representation. Thus, documents must be very close to an exact match to resemble. The I-Match approach strikes a

balance between parsing and the previously described existing techniques.

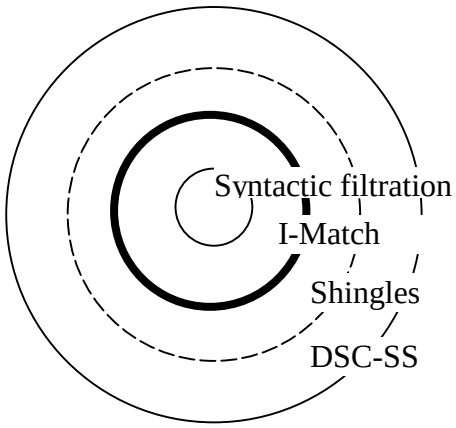


Figure 24. Restrictiveness of Techniques

I-Match does not rely on strict parsing but instead uses collection statistics to identify which terms should be used as the basis for comparison. It was previously shown that terms with high collection frequencies often do not add to the semantic content of the document [67, 68]. The I-Match approach hinges on the premise that removal of very infrequent terms or very common terms results in good document representations for identifying duplicates.

We input a document, filter the terms based on collection statistics (and other simple parsing techniques) and compute a single hash value for the document. All documents resulting in the same hash value are duplicates. We use the SHA1 algorithm [69] for the hash, using the ordered terms in the document as input and getting <docid, hashvalue> tuples as output. The ordering of terms is critical to detect similar documents that have reordered the paragraphs. The SHA1 hash algorithm is used because it is

designed to be very fast and is good for messages of any length. It is designed for text processing and is known for its even distribution of hash values.

SHA1 produces a 20-byte or 160-bit hash value. By using a secure digest algorithm, we reduce the probability of two different token streams creating the same hash value to $P\ (2^{-160})$. We insert each <docid, hashvalue> tuple into a tree requiring the processing time on the order of $(O(d \log d))$. Other efficient storage and retrieval data structures such as a hash table could be used as alternatives. The identification of duplicates is handled through the inserts into the tree. Any collisions of hash values represent duplicates and the document identifiers are stored in that node of the tree. A scan through the tree produces a list of all clusters of duplicates, where a node contains more than one document.

The overall runtime of the I-Match approach is $(O(d \log d))$ where d is the number of documents in the collection. This is comparable to the DSC-SS algorithm, which generates a single super shingle if the super shingle size is large enough to encompass the whole document. Otherwise, it generates k super shingles while we only generate one, $(O(kd \log kd))$ time. Since k is a constant in the DSC-SS timing complexity, the two algorithms are each theoretically equivalent. I-Match, however, is more efficient in practice.

The real benefit of I-Match over DSC-SS however, is not the timing improvement but that small sized documents are not ignored. With DSC-SS, it is quite likely that for sufficiently small documents, no shingles are identified for duplicate detection. Hence, those short documents are not considered, even though they may be duplicated. Given

the wide variety of domains that duplicate document detection may be used, e.g., document declassification, email traffic processing, etc., neglecting short documents is a potentially serious issue.

3.3 I-Match Results

We implemented the DSC, DSC-SS and I-Match algorithms in the IIT Advanced Information Retrieval Engine (AIRE) system [70]. To evaluate I-Match, we implemented a variety of filtering techniques based on various thresholds.

Figure 25 graphically illustrates several I-Match thresholding techniques. In the figure below, the shaded regions are discarded term regions. The next section describes, in detail, the different thresholding techniques. The results are broken into the following sections: experimental layout, syntactic one pass approaches, quality of duplicate sets found, handling short documents, runtime performance and effects on precision recall.

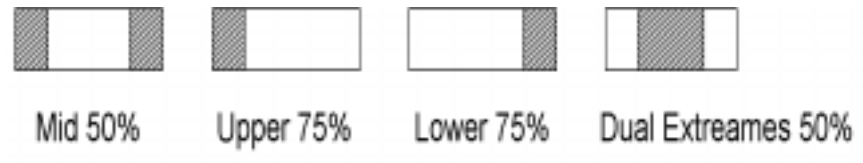


Figure 25. Thresholds for Document Nidf Values

3.4 Experimental Layout

We experimented with two filtration techniques based on collection statistics I-Match-Doc and I-Match-IDF. I-Match-Doc filters the unique terms of a given document by idf value to reach a specified percentage of the original unique terms of the document.

The terms remaining after the filter are used to create the clusters hash value. For instance, 75% of the document might be reached by removing the 25% of the terms with the lowest idf values (most frequent terms). Another example retains 50% of the original unique tokens of the document by removing 25% of the terms with the lowest idf and 25% of the terms with highest idf (least frequent). Thus, a percentage in terms of the number of unique terms of the original document will always remain constant, except for extremely small documents, i.e., a document containing less than four unique tokens would be filtered if we wanted to keep less than 25% of the original document.

The I-Match-IDF filtration technique filters terms based on normalized IDF values. The term IDF values are normalized across the collection so that they fall within a 0 to 1 interval. For each document, an IDF cut-off is used, thus any term above or below a certain idf value is removed from the terms to be used to create the clusters hash.

For each approach, we calculated the number of documents that were completely filtered, i.e., were not evaluated due to the removal of all tokens. We calculated the average distinct terms before and after filtration and the average number of terms in each document pre and post filtration. We counted the number of duplicate clusters found with each approach. We evaluated each duplicate set found and counted how many of documents within the cluster, matched on the evaluation technique, and how many of those did the title or URL match. Therefore, if a document was found to have a duplicate and both documents had either an identical title or URL then it was counted as a duplicate-title, otherwise it was counted just as a duplicate. We evaluated the number of unique documents in our collection, so a document cluster was counted only once.

Lastly, we noted the time to evaluate the collection. We tracked the following for each approach and each collection.

- *Number of documents filtered by the approach*
- *Pre/Post average number of unique terms per document*
- *Pre/Post average number of terms per document (Document size)*
- *Number of document clusters found*
- *Number of duplicates found with same URL/Title*
- *Number of duplicate documents found with just the duplicate approach*
- *Processing time*

We now describe the various thresholding values used. We ran experiments of the I-Match-Doc approach with thresholds of 10, 20, 30, 40, 50, 60, 70, 80, and 90% of the most common terms and the inverse of the least common terms, totaling 18 experiments. We ran the LOW and HIGH filters first, filtering the lowest *X percentage*, and the highest *X percentage* based on *idf* value. Then we filtered the edges of the document – the most frequent and least frequent terms, keeping the middle ones, 20%, 40%, 60% and 80%. Finally, we filtered the middle of the document, keeping only the most frequent and least frequent terms, inner 20%, 40%, 60%, and 80%, 8 more experiments.

The I-Match-IDF filters use cut-off thresholds to filter any word above and below certain normalized idf values. For the DSC-SS variant algorithm experiments, we collected document sizes both pre and post filtration and the timing results. Document size information is used to see how sensitive these types of algorithms are to smaller

documents. The DSC-SS runs used super shingle sizes of 2, 4, 5, 10, 15, 20, 25, 50, 75 and 100 shingles where each single was 10 terms. The DSC experiments used thresholds of 0.5, 0.6, 0.7, 0.8 and 0.9. Table 13, contains a detailed description of the I-Match experiments.

We used four document collections, as shown in Table 1. Each collection was chosen to test particular issues involved with duplicate detection. The first is an 18,577-web document collection flagged as duplicates by Excite@Home. The Excite@Home document collection was produced from ten million web documents gathered through crawling the World Wide Web. These documents were then filtered by the Excite@Home engineers to include only those documents thought to be "duplicate". The collection contains 18,577 documents, each of which is suspected of having a duplicate web document within the collection. Many URLs are in the collection repeatedly because of multiple spider inputs. This collection is approximately 30 megabytes in size. The Excite@Home collection is highly duplicated. Thus, as better approaches are used, the greater is the percentage of the collection found as duplicate.

The second is an 18,232 document Los Angles Times collection. A subset of the entire LA Times collection provided by NIST, this subset was selected to roughly mirror the Excite@Home collection in terms of the number of documents but to comprise of significantly longer documents. The LA Times subset collection is used to compare the various techniques by inserting known duplicate documents and analyzing the various approaches, for finding those documents.

The third and fourth collections are likewise from NIST and are the TREC 2GB

web and ad-hoc collections. The NIST web collection is a subset of a backup of the web from 1997 that was used in the TREC web track for TREC 7 and 8. This collection was chosen as a representative of a larger standard web collection to show the scalability of the I-Match algorithm. The NIST Web collection is used to test the run time performance of DSC, DSC-SS and I-Match approaches.

The TREC disks 4-5 are chosen as a second document collection of 2-gigabytes to see what effects duplication has on precision and recall. Since this collection has standard query and judgment results, it is a good collection to see if duplication has an effect on the end result sets. The NIST TREC collection is used to test the effects of duplication on known relevance judgments.

Table 1. Experimental Collections

Collection Name	CollectiOn Size	Number of Documents
Excite@Home Web	30 MB	18,577
NIST LA Times	100 MB	18,232
NIST Web	2 GB	247,491
NIST TREC disks 4 & 5	2 GB	528,023

Unfortunately, there is no available absolute body of truth or a benchmark to evaluate the success of these techniques. Thus, it is difficult to get any type of quantitative comparison of the different algorithms and thresholding techniques. This is not likely to change in the near future. As document collections grow, the likelihood of judgments of duplicates being made is small; therefore, the best that can be hoped for is to provide fast efficient techniques for duplication detection that can be passed on to analysis for further evaluation if desired.

3.5 Syntactic Filtration

The most obvious way to identify duplicate documents is to directly hash the entire contents of a document to a unique value. This type of approach finds exact matches by comparing the calculated hash value with the other document hash values and has a computational complexity of *(O d log (d))* where *d* is the number of documents in the collection. A simple hash of the entire document is not resilient to small document changes, like an additional space added to a document, the addition or deletion of the word "the", a stem change to a term, or the replication of a sentence or paragraph. Because of these reasons, hash values are not commonly used for duplicate document detection. However, they are, used to see if a particular document has changed.

We experimented with various filtration techniques to improve the resilience of the direct hash approach to small document changes. If a simple filtration technique based on strictly syntactic information is successful then fast duplicate and similar document detection could be achieved. We had to evaluate this basic approach prior to considering the use of more sophisticated, collection dependent, hence computationally expensive, filtration techniques.

We experimented with five filtering techniques that removed all white spaces from a document, and created a list of unique tokes to hash.

- *sw - Stop Word Filtration*
- *tg5 - Terms less than 5 characters in length*

- *tl25 - Terms greater than 25 characters in length*
- *nosc - Terms with special characters*
- *stem - Stemming*

All permutations of the filtration techniques were investigated. We used the 571-stop-word list used by many participants of the Text Retrieval Conference and available on the SMART information retrieval site [71]. For word length filters, we removed all the words less than the average word length [72], five, in the *length>5 (tg5)* filter. To filter very long words, we arbitrarily selected 25 as the cutoff for the *length<25 (tl25)* filter. For stemming, we used the Porter stemming algorithm [73].

The effect of filtering tokens on the degree of duplicate document detection is shown in Table 2. We used the Excite@Home collection because the collection is fully duplicated. Therefore, the percentage of duplicates found is an evaluation metric of the effectiveness of the filter. Also shown in the table is the percentage of terms retained after each filtering technique. Generally speaking, as we show in Table 2, the higher the filtration, the greater the degree of detection. While several of the filtration techniques do find 88% of the collection, the duplicates they find are near or exact matches and a maximum number of unique documents of 2038. In contrast, I-Match for this same collection detects 96.2% duplication and a maximum number of unique documents of 568. Clearly the lower the maximum number of unique documents, the better is the detection capability. The simple filtering techniques reduced the list of tokens used to create the hash. By eliminating white spaces and only keeping unique tokens, many small document changes are eliminated. Keeping only unique tokens eliminates movement of paragraph errors, stemming removes errors caused by small token changes, and

stop word removal removes errors caused by adding or removing common irrelevant, in terms of semantics, tokens. We found that removing tokens containing 'special characters' (i.e. /, -, =, etc.) performed the best in terms of removing tokens from documents.

Table 2. Syntactic Experiments

	Percentage of Original Lexicon	Percent Found as Duplicates	Unique documents found in collection
nothing	100.0%	62.2%	7017
sw	99.9%	62.2%	7017
tg5	93.2%	62.4%	6966
sw,tg5	93.2%	62.4%	6966
tl25	60.1%	82.4%	3253
sw,tl25	60.1%	82.4%	3253
tg5,tl25	53.4%	82.7%	3199
sw,tg5,tl25	53.4%	82.7%	3199
nosc	9.5%	87.4%	2214
nosc,sw	9.4%	87.4%	2197
nosc,tg5	7.0%	88.0%	2048
nosc,tg5,sw	6.9%	88.0%	2043
nosc,tl25	9.5%	87.4%	2214
nosc,tl25,sw	9.4%	87.4%	2197
nosc,tl25,tg5	7.0%	88.0%	2048
nosc,tl25,tg5,sw	6.9%	88.0%	2043
stem	80.4%	62.2%	7014
stem,sw	80.4%	62.2%	7014
stem,tg5	78.2%	62.4%	6963
stem,sw,tg5	78.2%	62.4%	6963
stem,tl25	41.2%	82.4%	3248
stem,sw,tl25	41.2%	82.4%	3248
stem,tg5,tl25	39.0%	82.7%	3192
stem,sw,tg5,tl25	39.0%	82.7%	3192
stem,nosc	6.9%	87.4%	2211
stem,nosc,sw	6.9%	87.4%	2194
stem,nosc,tg5	5.2%	88.0%	2045
stem,nosc,tg5,sw	5.2%	88.0%	2039
stem,nosc,tl25	6.9%	87.4%	2211
stem,nosc,tl25,sw	6.9%	87.4%	2194
stem,nosc,tl25,tg5	5.2%	88.0%	2045
stem,nosc,tl25,tg5,sw	5.2%	88.0%	2038

These syntactic filtration techniques are very fast; however, the degree of duplication they detect is limited. Such approaches detect only near or exact duplicates and do not find documents with small differences, like an updated date string or different URL. Therefore, simple filtration techniques such as these do not suffice, and efforts such as DSC, DSC-SS, and I-Match merit further investigation.

3.6 Duplicate Sets

For the task of "remove all duplicates from this collection", it is helpful to get a list of duplicate document sets so that one from each set can be retained and the rest removed. Imagine getting, instead, many different lists of duplicates, where one document may be in many lists. This is essentially what DSC and DSC-SS return. The DSC-SS algorithm creates a duplicate document set for each super shingle that exists in at least two documents. Thus, each document (if it matches more than one super shingle) may appear in multiple document lists. For example, given documents D1, D2, D3 and D4 with super shingles as follows: D1 contains super shingle A. D2 contains super shingle A and B. D3 and D4 contain super shingle B. The resulting sets of duplicates include {D1, D2} (from super shingle A), {D2, D3, D4} (from super shingle B). Now all of the clusters must be scanned to get a list of duplicates for D2. In contrast, I-Match places each document in one and only one duplicate document set.

Consider two documents that match all text except in one small portion as shown in Figure 26. Perhaps a name and an address for a regional contact are changed. It is likely that DSC-SS would identify these two documents as duplicates because the small section that differs may not be represented at all in the selected shingles or a super shingle

exists without a shingle from this section. I-Match will group these together as duplicates only if all terms in the differing section were filtered. This is quite likely with the name and address example because names are generally very infrequent across the collection, the numbers are removed in parsing and state names are generally very common across the collection. On the other hand, if any word in the differing section is kept, the two documents are not matched.

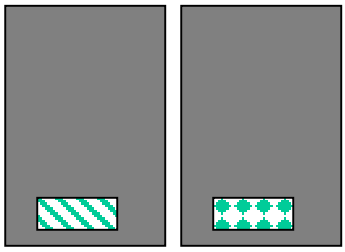


Figure 26. Differing Documents

To find the best performing I-Match approach, we contrived a set of duplicates to test the various approaches with a known test set of duplicate documents inserted into an existing collection. We computed the average document length for the test collection. We then chose ten documents from the collection, that were the average document length. These documents were used to create a test duplicate document collection. Each document is used to create 10 test duplicate documents. This is achieved by randomly removing every i^{th} word from the document. In other words for every i^{th} word, pick a random number from one to ten. If the number is higher than the random threshold (call it alpha) then pick a number from 1 to 3. If the random number chosen is a one then remove the word. If the number is a two then flip it with a word at position i+1. If it is a three, add a word (randomly pick one from the term list). Lastly, these duplicate

documents are now inserted into the collection.

We then ran the I-Match thresholding techniques, DSC, and the DSC-SS with the creation of a super shingle for every 2 and 4 shingles on the LA Times sub-collection, looking for the new test duplicate documents. We found two I-Match filtration techniques to be very effective I-Match (Doc-L-90 and IDF-L-10). Doc-L-90 takes only terms with the highest IDF values, i.e., very infrequent terms, and only the 10% most infrequent terms are used for each document. The second approach (IDF-L-10) uses only the terms with normalized idf values of 0.1 or greater, thus very frequent terms in the collection are removed. In the following tables, we present the data obtained in our evaluation of the different approaches.

Table 3. Documents Found Ratio

Document	DSC	DSC-SS-2	DSC-SS-4	DOC-L-90	IDF-L-10
LA123190-0013	27.3%	0.0%	18.2%	36.4%	63.6%
LA123190-0022	54.5%	63.6%	18.2%	100.0%	100.0%
LA123190-0025	27.3%	0.0%	0.0%	90.9%	100.0%
LA123190-0037	18.2%	18.2%	0.0%	90.9%	100.0%
LA123190-0043	36.4%	0.0%	0.0%	90.9%	90.9%
LA123190-0053	18.2%	45.5%	45.5%	90.9%	100.0%
LA123190-0058	45.5%	18.2%	0.0%	90.9%	81.8%
LA123190-0073	54.5%	0.0%	0.0%	100.0%	100.0%
LA123190-0074	0.0%	0.0%	0.0%	90.9%	100.0%
LA123190-0080	27.3%	18.2%	0.0%	54.5%	63.6%
Average	30.9%	16.4%	8.2%	83.6%	90.0%

As shown, both I-Match approaches yield a significantly higher percentage of detection than either DSC or either of the DSC super single approaches. Furthermore, as expected, the super single approaches declined in the percentage detected as the super

single size increased. The DSC performance was better than both super single approaches.

The most effective I-Match techniques are retaining the highest idf valued terms from a document either as a percentage or as a normalized value. We produced 10 duplicate documents for 10 test documents, thus creating 11 known duplicate documents for each cluster. In Table 3, we show the percentage of the total document duplication found for each approach. Both I-Match approaches find a greater duplication percentage for all test cases.

Table 4. Document Clusters Formed

Document	DSC	DSC-SS-2	DSC-SS-4	DOC-L-90	IDF-L-10
LA123190-0013	9	11	9	9	7
LA123190-0022	6	7	9	3	2
LA123190-0025	9	11	11	4	3
LA123190-0037	10	10	11	4	1
LA123190-0043	8	11	11	2	2
LA123190-0053	10	9	9	3	2
LA123190-0058	7	10	11	3	3
LA123190-0073	6	11	11	3	3
LA123190-0074	11	11	11	2	1
LA123190-0080	9	10	11	8	9
Average	8.5	10.1	10.4	4.1	3.3

In Table 4, we illustrate that the I-Match techniques yield a smaller number of document clusters than any of the shingling techniques. That is, we know, by design, that for each document the actual number of clusters to be formed should ideally be one since besides the original document, the other ten copies are simply slight modifications of the original. Therefore, a perfect similar document detection algorithm would generate one

cluster per document. As shown, the I-Match configurations result in an average number of clusters per document of approximately 3 to 4. DSC and the super shingling variants are significantly worse ranging from 8 to 10 clusters.

3.7 Short Documents

While DSC-SS is more efficient than DSC, it has known deficiencies with short documents. To evaluate how often DSC-SS completely ignores a document, we ran the algorithm against the Excite@Home duplicate document collection and the NIST LA Times collection. As presented in Figure 27, for the Excite@Home document collection, DSC-SS ignored over 6,500 documents for a super shingle size of two. As for the LA Times collection, DSC-SS ignored over 1,200 documents. In comparison, DSC ignored 5052 and 636 documents, respectively. The high number of filtered documents for the Excite@Home collection is caused by the filtration of common shingles produced by this contrived collection where the 636 documents filtered from the LA Times collection is probably a better representation of the algorithms performance. I-Match, in the worst case, ignored only four documents.

In Figure 27, we illustrate the increase in the number of documents ignored as the number of shingles used to create a single super shingle increases. The more shingles used to make a super shingle, the more documents are ignored. We then ran the DSC-SS algorithm against the 2GB NIST collection with super shingle sizes of 100, 75, 50, 25, 20, 15, 10, 5, 4 and 2 shingles. In Table 5, we once again show that the greater the super shingle size the more documents ignored, thus validating our prior results using the LA Times and Excite@Home collections. In Table 5, we also illustrate the percentage of the

collection filtered.

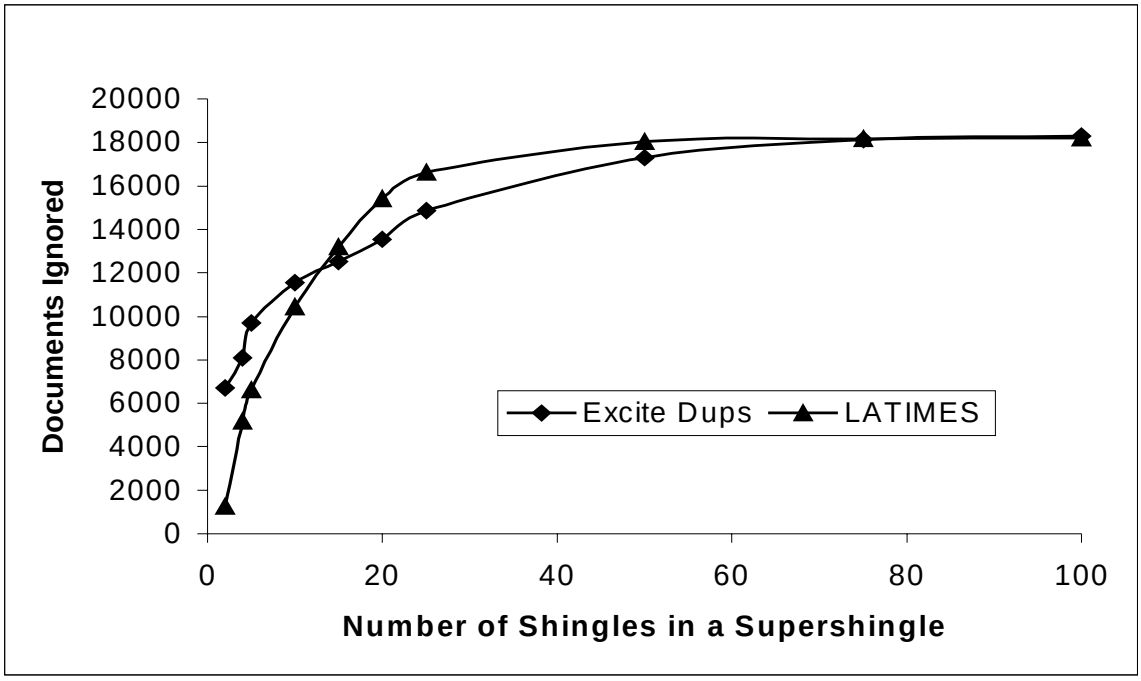


Figure 27. Super Shingle Size vs. Documents Dropped

Table 5. DSC-SS Short Document Filtration

Super Shingle Size	Documents Ignored	% Filtered
100	220073	88.92%
75	209928	84.82%
50	189071	76.40%
25	133614	53.99%
20	112703	45.54%
15	86288	34.87%
10	54212	21.90%
5	22257	8.99%
4	16805	6.79%
2	6528	2.64%

The I-Match algorithm uses various term filtration techniques based on collection

statistics to filter terms. We conducted 52 different filtration experiments. For most I-Match runs, only about 150 documents were filtered or less than .06% of the collection. Since our best filtration techniques take only a percentage of the document, only documents with a couple unique terms are ignored. The only I-Match thresholding technique to filter a substantial percentage of documents filters based on IDF values retaining only a normalized IDF value of 0.9 or greater. This technique keeps less than 50% of the collection, similar to a DSC-SS of 50. In spite the degree of filtering, no I-Match thresholding technique dropped any significant number of documents. That is, the greatest number of documents dropped was 143 out of 247,491.

Table 6. Post Average Document Size

Super Shingle Size	Post Avg Doc Size
100	9860
75	8123
50	6109
25	3833
20	3389
15	2963
10	2575
5	2272
4	2225
2	2140

As the super shingle size increases, the average size of a document that survives the filtration process increases. In Table 6, we present the average number of tokens per document retained after super shingling. The average token size is about six characters in length. The sizes of the documents are presented in terms of the number of terms. Thus, multiplying by six [72] estimates the average size of a document. This sub-collection of

the web has slightly higher document sizes than the 4K sizes reported in [60]. This shows us that the DUP-SS algorithm performs poorly on web sized documents.

3.8 Runtime Performance

We divide the runtime performance experiments and results into two sections. The first set of experiments compares the I-Match and the DSC-SS algorithms on the Excite@Home test document collection. The second set of experiments compares the I-Match, DSC-SS and DSC algorithms using the 2-gigabyte NIST web collection. All experiments were run on a SUN ES-450; each process ran with about 200MB for all algorithms.

I-Match was approximately five times faster than DSC-SS for the Excite@Home collection. The pure Syntactic Filtration technique ran in less than 5 seconds, but as discussed previously, only exact matches are found with this technique. Varying the threshold for super shingle sizes does not significantly influence the runtime since the same amount of work must occur. Our best performing I-Match techniques ran in 142 (Doc-L-90) and 134 (IDF-L-10) seconds.

The DSC and DSC-SS timings for the Excite@Home collection are comparable since the third and fourth steps of the DSC algorithm are I/O bound in nature but are relatively negligible for a small collection. The third and fourth steps in the DSC approach become a greater percentage of the cost as the collection grows as seen in Table 8 for the 2GB NIST collection.

We compared the run time of I-Match to the DSC and DSC-SS algorithms

running against the NIST 2 gigabyte Web collection. As with the Excite@Home experiments, the parsing/indexing system builds shingle data and relevance feedback data structures when indexing a collection. Thus, preprocessing the text and creating shingle and token data times are not contained in our timing results, just the specific clustering or duplication algorithm.

Table 7. Duplicate Processing Time

Algorithm	Mean Time	Std Deviation	Median Time
DSC	595.4	4.3	593.5
DSC-SS	587.6	18.5	587.1
I-Match	96.9	33.4	82.6
Syntactic	5	N/A	N/A

As shown in Table 8, I-Match was approximately six times faster than DSC-SS and almost 9 times faster than the DSC algorithm. The faster speed of the I-Match algorithm suite is due to the processing of fewer tokens. The average distinct tokens per document for the NIST 2 gigabyte collection is approximately 475, while the average document size is over 2000 terms long. Since a sliding window creating shingles produces about the same number of shingles as the size of the document, the added amount of processing is proportional. This is true for all small window sizes proportional to the total document size. If a large window size for super shingles is used, the DSC-SS approach is just a hash approach and will not match on similar documents. This ratio of distinct terms to document size is consistent to our TREC collection statistics.

The DSC algorithm has several additional steps, which are I/O bound in nature, and contributes to its additional run time. Table 8 contains an average of timing

results for each of the given techniques. DSC had five experiments using a threshold of 50%, 60%, 70%, 80% and 90%. DSC-SS had ten experiments enumerated in Table 5. I-Match results are from 52 experiments described above. Lastly, a syntactic filtration comparison is given. Detailed experimental results are presented as an appendix and are listed in Table 13 and Table 15 with the legend describing the experimentation present in Table 14.

Table 8. Processing Time for 2GB NIST Web Collection

Algorithm	Mean Time	Std Deviation	Median Time
DSC	31838.22	807.9	30862.5
DSC-SS	24514.7	1042.1	24475.5
I-Match	3815.8	975.8	3598.8
Syntactic	65	N/A	N/A

3.9 Duplication Effecting Accuracy

We examined the effects of duplication on result sets. We used the I-Match algorithm on the TREC disks 4-5, which were used for TREC 6-8. We used the NIST relevance judgments to flag documents judged by NIST. If a duplicate was found and a positive judgment was made for a given query, we checked to make sure that no false judgments were made on its duplicates. A dozen inconsistencies were found for TREC 6. Eight inconsistencies were found for TREC 7. Seventeen inconsistencies were detected in TREC 8 and 65 inconsistencies were noted for the web track of TREC 8. Examining these inconsistencies, we found documents that were identical were judged differently. TREC topic 301, judged document FBIS3-58055 relevant and FBIS3-58055 not relevant; they are the same document except for the document number. Another example

for topic 301 is that document FBIS3-41305 was judged relevant and document FBIS3-41305 was not judged as relevant although these documents are identical except for the title and the document number. Similar examples were found for TREC 7 and 8 and the web track of TREC 8. While this does not diminish the usefulness of the TREC judgments, it does show that duplicate document detection is important from both a research point of view and an end user's point of view.

3.10 Conclusions and Future Work

Algorithms for detecting similar documents are critical in applications where data are obtained from multiple sources. The removal of similar documents is necessary not only to reduce runtime but also to improve search accuracy.

We proposed a new similar document detection algorithm called I-Match, and evaluated its performance using multiple data collections. The document collections used varied in size, degree of expected document duplication, and document lengths. The data collections were obtained from NIST and from Excite@Home.

I-Match relies on collection statistics to select the best terms to represent the document. I-Match was developed to support web document collections. Thus, unlike many of its predecessors, I-Match efficiently processes large collections and does not neglect small documents. In comparison to the prior state-of-the-art, I-Match ran five times faster than DSC-SS against the Excite@Home test collection and six times faster against the NIST 2GB collection. Furthermore, unlike the efficient version of the prior art, I-Match did not skip the processing of small documents.

In terms of human usability, no similar document detection approach is perfect. The ultimate determination of how similar a document must be to be considered a duplicate relies on human judgment. Therefore, any solution must be easy to use. To support ease of use, all potential duplicates should be uniquely grouped together. Shingling approaches, including the DSC and DSC-SS approaches, however, group potential duplicate documents according to shingle matches. Therefore, any match in even a single shingle results in a potential duplicate match indication. This results in the scattering of potential duplicates across many groupings and many false positive potential matches. I-Match, in contrast, treats a document in its entirety and maps all potential duplicates into a single grouping. This reduces the processing demands on the user.

CHAPTER IV

DELAYED IDF UPDATES

Information Retrieval (IR) is directed towards finding relevant data in the form of documents, in response to user requests (commonly referred to as queries). Computerized or automatic information retrieval has been a topic of both commercial development and research for many decades. To improve accuracy, many text search systems automatically assign weights to terms [77]. The idea is to weigh infrequent terms high and frequent terms low. Hence, a search for "Telecommunications stock" will weigh the term "stock" substantially lower than "Telecommunications" because "stock" occurs much more frequently and is essentially noise (especially in articles found in a newswire). The inverse document frequency is a well-accepted, automatically assigned weight that is computed as log(N/df) where N is the number of documents in the collection and df is the number of documents that contain the term in question. If the term appears in all N documents, it is clearly noise and is weighted as log(N/N) = 0. Similarly, if the term appears in only one document, it is considered quite significant and is weighted as the log(N). Notice that N changes with the addition of each new document to the collection.

For an information retrieval system that uses constantly, changing data (i.e., a system that indexes a user's e-mail) the cost of updating the value of N with each new document can be quite prohibitive. Most work on information retrieval systems focused on static data; therefore, this has not been widely addressed. Viles and French [74] showed that the inverse document frequencies did not have to be updated very

often to assure good accuracy, but this work was done on a very small document collection. Experimentation on small document collections was shown to be non-indicative for larger collections [75].

We tested the hypothesis that a sufficient amount of "training" data is sufficient to assign the term weights and once the weights are assigned, they need not be updated frequently [19, 20]. We tested a variety of different training set sizes (10 MB, 20 MB, 40MB, 80MB, and 160 MB) for a 320MB collection. We also tested different update intervals for these training set sizes. We found that the size of the training set was not nearly as important as the number of unique terms identified in a training set. (Clearly, up to a point, this number of distinct terms increases with the size of the training set.)

We found that a training set size of 50% of the document collection did not require any updating of the inverse document frequencies to maintain accuracy in retrieval. For a training set size of 10% and 20%, accuracy degraded as compared to that of 40%. Hence, for a collection similar to the one used, the "right" training set size is around 20-40 percent of the document collection. Note that these estimates are based on the assumption that the number of new terms appears at the same rate as observed in our test collection. A safer means of computing training set size, might be to use the number of distinct terms.

The decision of when to update the inverse document frequencies should be driven by the user requirements. If the requirement is for the theoretically highest achievable accuracy then the inverse document frequency should be updated constantly; however, if there is some flexibility in this requirement, more infrequent updates will

certainly improve system performance by reducing processing overhead without noticeably affecting accuracy. We try to answer an important efficiency question. What is a reasonable update frequency for the inverse document frequency used in the vector space model? We start with a training collection of 40, 80, and 160 MB, we observed no significant change in the accuracy of our retrievals when the inverse document frequencies were updated frequently or infrequently. Accuracy was affected when the training size was only 10 or 20 MB. Since the order in which new documents appear could affect accuracy, we tested two different orderings and the results were comparable for each ordering.

4.1 Vector Space Model

To evaluate the relevance of each document, for each query-document pair, a measure of relevance is computed. Accordingly, the documents within the collection are then ranked based on this measure. A popular means of computing a similarity measure is the vector space model. This model defines a vector that represents each document, and a vector that represents the query [76]. Once the vectors are constructed, the distance between the vectors, or the size of the angle between the vectors, is used to compute a similarity coefficient.

There is one component in each vector for every distinct term that occurs in the document collection. Consider a document collection with only two distinct terms, α and β. All vectors contain only two components. The first component represents occurrences of α, and the second represents occurrences of β. The simplest means of constructing a vector is to place a one in the corresponding vector component if the term

appears, and a zero, if the term does not appear. Consider a document, D_1, that contains two occurrences of term α and zero occurrences of term ß. The vector, *<1,0>*, represents this document using a binary representation. This binary representation can be used to produce a similarity coefficient, but it does not take into account the frequency of a term within a document. By extending the representation to include a count of the number of occurrences of the terms in each component, these frequencies can be considered. In the example, the vector would now appear as *<2,0>*.

Early work in the field used manually assigned weights. Similarity coefficients that employed automatically assigned weights were compared to manually assigned weights [77, 78]. Repeatedly, it was shown that automatically assigned weights would perform at least as well as manually assigned weights [77, 78].

Unfortunately, the above approach does not include the relative weight of the term across the entire collection. The utility of including a collection-wide based weight was studied in the 1970's, and the conclusion was that relevance rankings, the ordering of documents with respect to their relevance to the user query, improved if this weight was included. Although relatively small document collections were used to conduct the experiments, the authors still determined that "in so far as anything can be called a solid result in information retrieval research, this is" [79].

To construct a vector that corresponds to each document, consider the following definitions:

- *n = number of distinct terms in the document collection*

- tf_{ij} = *number of occurrences of term* t_j *in document* D_I
- df_j = *number of documents which contain* t_j
- $idf_j = \log d/df_j$ *where* d *is the total number of documents*

The vector for each document is of size *n* and contains an entry for each distinct term in the entire document collection. The components in the vector are filled with weights that are computed for each term in the document collection. The terms in each document are automatically assigned weights based on how frequently they occur in the entire document collection and how often a term appears in a particular document. The weight of a term in a document increases the more often the term appears in a document and the less often it appears in all other documents.

The weights computed for each term in the document collection are non-zero only if the term appears in the document. For a large document collection consisting of numerous small documents, the document vectors are likely to contain mostly zeros. For example, a document collection with 10,000 distinct terms results in a vector of size 10,000 for each document. A given document may have only 100 distinct terms. Hence, 9,900 components of the vector contain a zero.

The calculation of the weighting factor (*w*) for a term in a document is formally defined as a combination of term frequency (*tf*), document frequency (*df*), and inverse document frequency (*idf*). To compute the value of the *j*th entry in the vector corresponding to document *i*, the following equation is used:

$$D_{ij} = (tf_{ij}) (idf_j)$$

Consider a document collection that contains a document, D_1, with ten occurrences of the term *green* and a document, D_2 , with only five occurrences of the term *green*. If ***green*** is the only term found in the query, document D_1 is ranked higher than D_2.

The inverse document frequency can best be examined when term frequency is not a factor. Returning to our earlier example, for the query containing the terms "*Telecommunications stock"* it is assumed that *stock* occurs substantially more frequently that the term "*Telecommunications"*. For a document collection in which document D_1 contains one occurrence of "*stock"* and document D_2 contains only one occurrence of the term "*Telecommunications"*, document D_2 will be ranked higher than D_1, and "*Telecommunications"* will have a higher inverse document frequency than "*stock"*.

When a document retrieval system is used to query a collection of documents with *t* terms, the system computes a vector D of size *t* for each document. The vectors are filled with term weights as described above. Similarly, a vector Q is constructed for the terms found in the query.

A simple Similarity Coefficient (SC) between a query Q and an *i*th document D_i is defined as the Euclidean distance between the two vectors $SC(Q, D_i) = \sum_{j=1}^{t} q_j * d_{ij}$

where q_j is the *j*th term in the query and d_{ij} is the *j*th term in the *i*th document.

Consider a case insensitive query and document collection with a query Q and the sample document D given below.

Q: “nikkei stock exchange or american stock exchange.”

D:

<DOC>
<DOCNO> AP881214-0028 </DOCNO>
<FILEID>AP-NR-12-14-88 0117EST</FILEID>
<FIRST>u i BC-Japan-Stocks 12-14 0027</FIRST>
<SECOND>BC-Japan-Stocks,0026</SECOND>
<HEAD>Stocks Up In Tokyo</HEAD>
<DATELINE>TOKYO (AP) </DATELINE>
<TEXT>
The Nikkei Stock Average closed at 29,754.73 points,
up 156.92 points, on the Tokyo Stock Exchange Wednesday.
</TEXT>
</DOC>

Table 9. Document Table

Component	Term	DF	IDF	TF	Weight
1	american	6401	0.60	0	0.00
2	average	2265	1.08	1	1.08
3	closed	2208	1.08	1	1.08
4	exchange	2790	1.00	1	1.00
5	nikkei	234	2.07	1	2.07
6	points	1627	1.23	2	2.46
7	stock	2674	1.00	2	2.00
8	tokyo	725	1.58	1	1.58
9	up	12746	0.30	1	0.30
10	wednesda y	6417	0.60	1	0.60

The Component, Term, Document Frequency, Inverse Document Frequency, Term Frequency and Weight (*tf* * *idf*) values for the terms in the document are given in Table 9. Note the term “american” does not appear in the document, but since it does appear in the query, it is presented here for completeness.

The document vectors can now be constructed using the term weights given in Table 9. Since ten terms appear in the document collection, a ten-dimensional document vector is constructed. The alphabetical ordering given above is used to construct the document vector. The weight for term *i* in vector *j* is computed as the $(idf_i)(tf_{ij})$

The document vectors are given below:

D = <0.00, 1.08, 1.08, 1.00, 2.07, 2.46, 2.00, 1.58, 0.30, 0.60>

Consider a query that requests all documents about "nikkei stock exchange or american stock exchange."

Q = <0.6, 0, 0, 2, 2.07, 0, 2, 0, 0, 0>

The SC(Q, D) would be computed as Q x D or SC(Q,D) = (2)(1.00) + (2.07)(2.07) + (2)(2.00) = 10.2849. First proposed in 1975, the vector space model is still a popular means of computing a measure of similarity between a query and a document [80].

In 1988, several experiments tried to improve the basic combination of *tf-idf* weights [81]. Many variations were studied, and the following weight function was identified as a good performer, where the literals are defined previously

$$w_{ik} = \frac{\log(tf_{ik} + 1.0) * idf_k}{\sum_{j=1}^{t} ((\log(tf_{ik} + 1.0) * idf_k)^2}.$$

Several different means of comparing the query vector with the document vector were implemented. These are well documented, the most common of these is the cosine

measure where the cosine of the angle between the query and document vector is given:

The cosine coefficient is defined as:

$$sim(Q, D_1) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sqrt{\sum_{j=1}^{t} (d_{ij})^2 \sum_{j=1}^{t} (w_{qj})^2}}$$

Note that the cosine measure "normalizes"' the result by considering the length of the document. With the inner product measure, a longer document may result in a higher score simply because it is longer, and thus, has a higher chance of containing terms that match the query not necessarily because it is relevant. The cosine measure levels the playing field by dividing the computation by the length of the document. We note that Singhal, et al, found that the field may have been leveled too much [82] as a study of recent results showed that long, relevant documents were often excluded simply because they are long.

4.2 IDF Experimentation

The vector space model can incorporate either automatic or manually assigned term weights. As we discussed in the prior sections, automatically assigned term weights were shown to perform well. Many weighting schemes were investigated, but the term frequency inverse document frequency (*tf-idf*) scheme remains popular. The term frequency (*tf*) is computed once for a given document and is relatively easy to add to an inverted index.

The inverse document frequency is based on the total number of documents. Hence, for a system with 1,000,000 distinct terms, the addition of a single document requires a computation for each of the terms in the document to compute the new idf based on the increase in the size of the document collection. For new documents that have many distinct terms, this requires a substantial amount of resources.

Since the *idf* is simply an estimate of significance, it is reasonable to expect that it does not need to be updated with every new document. This premise was tested by Viles and French et al., for a small document collection, but it has not been tested on a more realistically sized data set like the TIPSTER collection [74].

Viles and French demonstrated that the *idf* did not have to be updated frequently. Since the updates of the *idf* could prohibit widespread deployment of a commercial system (due to the reduced ability to add or delete documents), we investigated the frequency by which one needed to update the *idf*.

Our hypothesis was that it is not necessary to update the idf for every new document. We attempted to find the optimal update interval by using a *training collection* of 50% of the document collection. Once this was developed, sequences of text were added to the collection, and *idfs* were updated for different intervals of text.

Consider a collection with one document that contains terms *apple, boy,* and *cat.* Assume that the *idf's* for each of these terms are computed. A new document with *apple, boy,* and *dog* requires an update to the *idfs.* However, it may not be necessary to update the *idf's* if they are not significantly changed by the new document. This would be fine

except a term such as *dog* would now effectively not be in the inverted index although its document has been added to the system. A user who searches for *dog* would not find this document although it was just added. Hence, even if it is not necessary to update *idf's* very often, the risk is that a unique term could appear between the updates of the *idf's* and that term could be extremely useful to obtain accurate results.

4.3 Results

We tested our hypothesis on a 320MB subset of the TREC collection. For our initial results, a training set size of 160 MB (roughly half of the document collection) was used. That is, files numbered 1-160 were used as our training set, and files numbered 161-320 were incrementally added to the collection. The *idf's* were initially computed using a training set. After training, *idf's* were only updated every u MB, where u = 10, 20, 40, 80, and 160MB. (160MB is equivalent to no updates of *idf* except for the training set.) The effect on average precision was measured every 10 MB to determine the impact of *not* updating the *idf*.

Figure 28 illustrates the average precision-recall for various update frequencies. The five different lines shown are not significantly different from each other. To test the impact of the input order, we reversed the training set and the document collection. In Figure 29, we present the results obtained when reversing the order. That is, files numbered 161-320 were used as a training set, and files 1-160 were incrementally added to the collection. Somewhat surprisingly, the order did not significantly affect average precision-recall measurements. Hence, for a relatively large training set, we conclude that

the update of *idf's* has a negligible effect on accuracy.

Table 10 below summarizes the average precision-recall for each update interval. It can be seen that the average precision-recall does not vary by more than 0.3 percent.

Having noticed no significant degradation in accuracy due to delayed idf updating but being aware that the order of appearance of new terms (order of document insertion into the collection) could significantly affect the results (i.e., order). We investigated an additional 160MB training set. For clarity of motivation, consider a case where a term that was *not* seen in the training set appears between *idf* updates and occurs in a relevant document. Hence, changing the order of the input files could result in such a term appearing in the training set and subsequently being found in a relevant document.

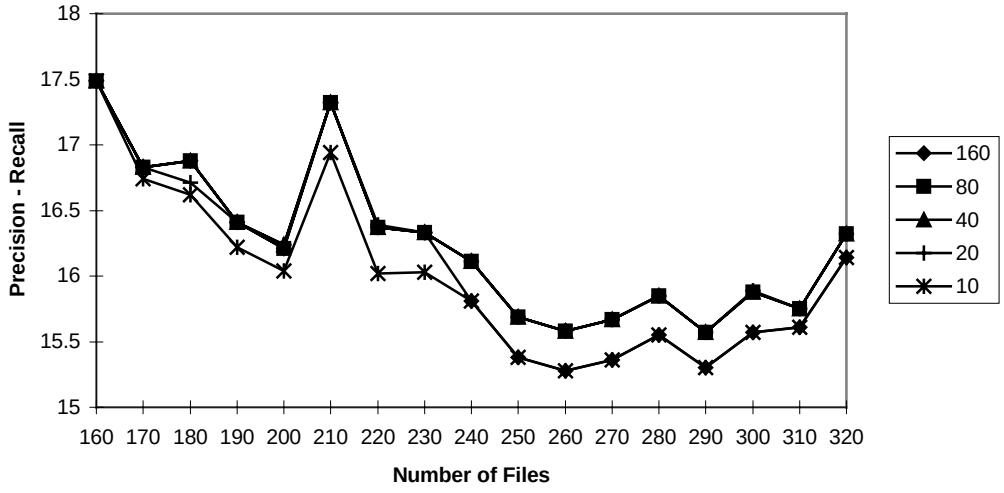


Figure 28. Precision - Recall for Different Update Frequencies (1-160)

To test the impact of the input order, we reversed the training set and the

document collection. In Figure 29, we present the results obtained when reversing the order. That is, files numbered 161-320 were used as a training set, and files 1-160 were incrementally added to the collection. Somewhat surprisingly, the order did not significantly affect average precision-recall measurements. Hence, for a relatively large training set, we conclude that the update of *idf's* has a negligible effect on accuracy.

Table 10. Average Precision / Recall for Training Set (1-160)

	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300	310	320
160	17.49	16.83	16.88	16.41	16.21	17.32	16.37	16.33	15.81	15.38	15.28	15.36	15.55	15.3	15.57	15.61	16.14
80	17.49	16.83	16.88	16.41	16.21	17.32	16.37	16.33	16.11	15.69	15.58	15.67	15.85	15.57	15.88	15.75	16.32
40	17.49	16.83	16.88	16.41	16.24	17.32	16.37	16.33	16.11	15.69	15.58	15.67	15.85	15.57	15.88	15.75	16.32
20	17.49	16.83	16.71	16.41	16.24	17.32	16.39	16.33	16.11	15.69	15.58	15.67	15.85	15.57	15.89	15.75	16.32
10	17.49	16.74	16.62	16.22	16.04	16.94	16.02	16.03	15.81	15.38	15.28	15.36	15.55	15.3	15.57	15.61	16.14

Table 11. Average Precision / Recall for Training Set (161-320)

	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300	310	320
160	20.71	20.34	20.04	19.55	18.72	18.31	17.99	17.67	17.56	17.69	17.69	17.32	17.27	17.04	16.59	16.47	16.30
80	20.71	20.34	20.04	19.55	18.72	18.31	17.99	17.67	17.44	17.59	17.6	17.24	17.15	16.96	16.53	16.42	16.30
40	20.71	20.34	20.04	19.55	18.8	18.38	18.07	17.76	17.44	17.59	17.6	17.24	17.14	16.95	16.48	16.36	16.30
20	20.71	20.34	20.06	19.59	18.8	18.38	17.95	17.62	17.44	17.59	17.58	17.23	17.14	16.95	16.52	16.40	16.30
10	20.71	20.37	20.06	19.62	18.8	18.27	17.95	17.63	17.44	17.56	17.58	17.21	17.14	16.98	16.52	16.41	16.30

(Note the reader should not compare the curves across Figure 28 and Figure 29 since the documents used in the training sets differ. Instead, what should be noticed is that in both sets of results the frequency of the update intervals of the *idf's* does not significantly affect the average precision-recall as the precision-recall numbers of the different update frequencies are roughly equivalent.)

Having concluded that given a sufficiently large training set, the order of

document insertion had little significance, we investigated the effects of the size of the training set. The training set size for the results presented in Figure 28 and Figure 29 encompassed half of the document collection. We hypothesized that the reason for our failure to detect any effect on average precision-recall due to *idf* update frequency was that this training set was relatively large. To measure the effect of training set size on performance, we tested training collections of t = 80, 40, 20, and 10 MB.

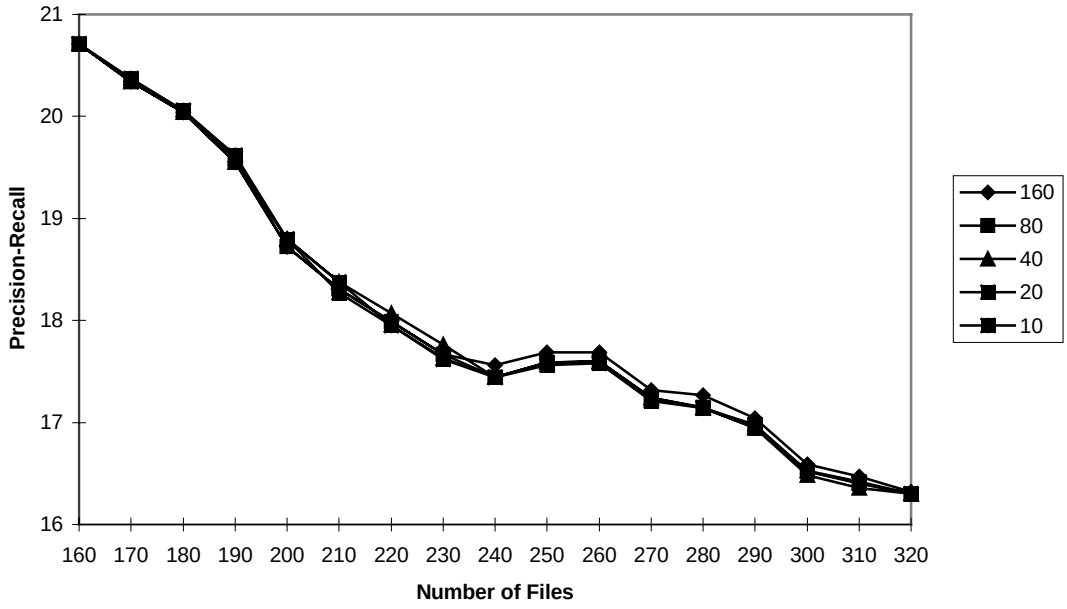


Figure 29. Precision - Recall for Different Update Frequencies (161-320)

In Figure 30, we present our results for these smaller training sets. In this study, we used the first x files, x = 80, 40, 20, and 10 as our training set. A significant difference in the average precision-recall measurements occurs when the training set drops below 20MB. At roughly 7% of our document collection size, the training set is small enough to illustrate the impact of longer *idf* update intervals. One reason for this

effect is that there are far fewer distinct terms in a training set of size 10MB than one of 160MB. In Table 13, we indicate the number of distinct terms for different training set sizes. For a training size of 10MB, only 34,953 distinct terms had been observed. Our training set sizes of 160MB had nearly tripled the number of distinct terms, thereby dramatically, increasing the chance that a query term will appear in the training set.

Table 12. Training Sets and Number of Unique Terms

Training Set Size	Number of Unique Terms Elements
160	140078
80	97408
40	69651
20	48730
10	34953

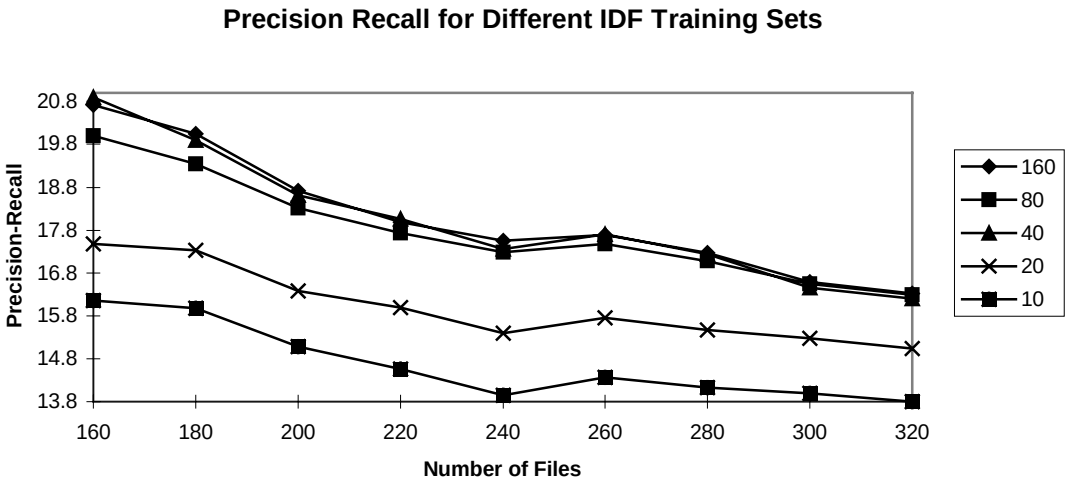


Figure 30. Precision - Recall for Different Training Sets

In summary, as demonstrated by our experimentation, given a sufficiently large (rich in the number of unique terms) training collection, the *idf's* need not be

updated frequently to support high average precision-recall measures. Thus, information retrieval systems can reduce processing overhead by initially collecting a sufficiently rich set of terms, and computing their idf's prior to future document insertion, and then, only infrequently updating the collection *idf's*. It remains an open question as to how to determine what is a sufficient set of terms to constitute as the basis for the collection *idf's*.

4.5 IDF Update Conclusions

We investigated the effect of infrequent updates to the inverse document frequency. Most research systems simply update the inverse document frequencies with the addition of each new document. Avoiding the need to update *idf's* saves computational resources by reducing computational overhead.

In our investigation, given a sufficiently rich collection of terms used in a training set to derive the *idfs*, we found that the update frequency did not significantly affect performance. This matches a prior result in the literature, but that work was done using a very small document collection. Even with our collection, which was substantially larger, *we did not notice significant impact caused by infrequent idf updates*. Since this result is somewhat counterintuitive, we investigated the impact of different training set sizes and found that for a very small training set (only 10MB) failure to update the *idf's* did adversely impact average precision-recall. Hence, a small training set is insufficient to compute the *idf's*. Given an appropriate training set, in a real system, frequent updates to the *idf* are unnecessary.

We found that a training set size of 50% of the document collection does not require any idf updates. For a training set size of 10% and 20% accuracy started to degrade as compared to 40%. Note that these estimates are based on the assumption that the number of new terms appears at the same rate as observed in our test collection.

The decision of when to update the idf's should be driven by user requirements. If the requirement is for the highest achievable accuracy, the *idf's* should be updated constantly; however, if there is some flexibility in this requirement, updates that are more infrequent yield statistically equivalent results.

It is critical that this assertion be tested against a large collection. Initial results in this area used a very small collection and did not report any significant impact on average precision-recall due to training set size. *Using a 320 MB collection, we observed some degradation in average precision-recall due to infrequent idf updates when the number of unique terms used in the training set did not accurately represent the document collection.* This result was not uncovered by prior work because the test collection was too small. To protect against any other incomplete results, we need to expand our work to the TReC (10 GB) web collection. We now have a process in place to implement these various algorithms so it should be possible to scale up and measure our results for an even larger collection.

We have reproduced the somewhat counterintuitive result that updates to idf's do not significantly impact effectiveness. Additionally, efforts are required that study means to determine the number of unique terms that are required to accurately represent the entire document collection. This term set is likely to be based on the size of the total

document collection and the rate of introduction of new terms per document. Additionally, it is also likely to be the case that a training set should include different sample sizes based on domains.

CHAPTER V

CONCLUSIONS AND FUTURE DIRECTIONS

Computer Science is providing algorithms, theoretical models and powerful computational machines all for one common goal; to improve the quality of our everyday lives. While this is an altruistic goal for all scientists, in reality, funding from governmental agencies and businesses has guided the goals of research. Regardless of the motivation, our world has just started to see the improvements that science has to offer. Today's growth in productivity and efficiency can be directly correlated with the computer age. This thesis provides algorithms and approaches to solve some of the growing problems that computer systems are encountering as they try to tackle larger and larger problems.

In addition to computer scientists, other professionals are providing fundamental research and improvements in their respective fields: doctors, physicists, sociologists, etc. The common thread for this accelerated development of new ideas and technologies has been our ability to improve the methods for sharing information. Without the ability to share and store information, our capacity to piggyback on the work of others is greatly hindered. It is this essential ability that has stressed the importance of information storage and retrieval. The growth of information being stored and shared is exponential and without new approaches for the scalability of information storage and retrieval systems; computer science and other disciplines' future development will be held back.

The importance of information systems to not only computer scientists but for all

disciplines is apparent. What is not known is the future direction these systems will proceed to solve current and future growth patterns. This thesis made several generalized assumptions about the future of information systems. These assumptions were made to provide a framework based on current trends in which new algorithms can be presented.

1. Data/Information is growing at a faster rate than compute power.
2. Parallelism of systems is the most promising solution to the growing needs of retrieval systems.
3. Data/Information are coming from many sources.
4. Any algorithm that helps efficiency or effectiveness is beneficial.

Given the above assumptions about future information systems, new retrieval systems will be parallel/distributed in nature. Data will be stored on a number of systems, indexing of that data will be done with multiple machines and CPUs. Retrieval of information will be distributed to multiple machines and CPUs. New large-scale information systems will be closely coupled creating a large server array controlled by a single entity. These distributed and parallel information systems will solve complex information needs via many computers. Algorithms that allow these new systems to scale, improve efficiency and effectiveness are of fundamental importance and are imperative for future systems.

Given the above motivation and framework, this thesis presented computer algorithms for the design of reliable information server clusters. As information systems grow in size the need for reliability, parallelization and speed grows. A novel algorithm called DRS (Dynamic Routing System) was presented. The DRS algorithm sustains continuous availability of clusters of information or compute servers even during network failures. Additionally a new duplicate data detection algorithm called I-Match was

presented that is several times faster than the state of the art and more precise. By detecting duplicate data, redundant work is eliminated from indexing and retrieval processing. Additionally higher retrieval accuracy is provided by reducing the amount of redundant information returned to users. Finally, automatic term weighting utilities are examined; results are presented showing that as dynamic collections grow, automatic term weights do not need to be recalculated and still maintain system effectiveness. In the next sections, we examine the contributions of the various algorithms and examine future research efforts.

5.1 Reliability, Efficiency and Effectiveness

This thesis examined three issues for the scalability of future information systems. The first is a time sensitive proactive routing algorithm for distributed systems; the second is a duplicate data detection algorithm; the third is empirical evidence that automatic term weight calculations can be delayed without degrading accuracy but reducing maintenance overhead.

As client needs grow, server systems have become more complicated and require additional compute needs. While many techniques were applied to improve the performance of a single machine, these improvements are insufficient to keep up with the current growth of information. To combat the additional demands for computational resources, server systems are distributed among multiple computers to handle the additional computational requirements. As information systems get larger, they too are distributed to provide the additional processing needs. This distributed approach of dividing the problem into either multiple workers for the same problem or multiple

workers working on different client problems all depend on network communication. Since a network connecting the servers is the backbone of the computational server, it must be reliable. We provided data showing network failures constitute 13% of all hardware failures in a one-year study of over a hundred servers. Therefore, any fault tolerant server cluster must address network reliability in a manner that is aware of the time critical issues of server clusters. Our study of hardware failures validates the need for reliable network communication.

As part of the described effort, a proactive network routing protocol that reroutes network communication around failures was presented. This protocol and topology provides a fault tolerant network communications system for server clusters in the presence of failures. The algorithm is time sensitive in which failures are detected and addressed before they affect applications. This type of system provides the reliability that new Information Retrieval (IR) systems need as they become distributed. The contribution of this type of dynamic routing is its application time sensitivity where proactive network monitoring is applied as opposed to the traditional reactive routing approach when dealing with server clusters.

Our algorithm creates a fully connected graph from the existing server cluster in which each server may route network traffic. By doing so, failures to any single point in the graph can be circumvented and allow the cluster to continue working despite network failures. While any single point of failure can be dealt with, the algorithm also allows for multiple network failures, although not all multiple failures are recoverable. Additionally, the proactive monitoring of communication links enables time sensitivity of

applications to be addressed.

Since our algorithm is proactive, additional network traffic is incurred. We show that the additional overhead incurred by our approach does not adversely affect network performance and does greatly improve the network reliability. We examined the overall improvement to availability of this type of approach in comparison to other topologies. Our detailed analysis is provided via several probability models that allow the various topologies to be compared in terms of number of network failures and overall reliability. Our probability models show a 267% improvement to network availability over traditional topologies for a distributed server cluster. Additionally, we showed improvements on all levels from single to multiple failures. By providing fundamental reliability to the communication channels between distributed servers, issues of effectiveness and efficiency can be addressed.

We examined issues related to multiple information sources. As information retrieval systems collect data from multiple sources, the likelihood of multiple copies of the same data, or near duplicate documents being added to the system, increases. We hypothesize that if the duplicate information is detected and eliminated in a fast efficient manner, the system accuracy and performance is enhanced. We presented a duplicate document algorithm called I-Match based on collection statistics of terms in a given collection. Additionally, we showed that this algorithm preformed five to nine times faster than comparable algorithms. Lastly, we showed that the I-Match algorithm is more effective for finding duplicate documents than shingling approaches.

The I-Match algorithm selectively chooses what terms represent a document by

using term collection statistics. Once the document's representative subset of terms is selected, a hash is created of those terms. By reducing the number of representative terms based on collection statistics fuzzy matches can be determined. With a single value representing each document, every document in a collection can be scanned and evaluated in O(log d) time where d is the number of documents in the collection. Additionally, this algorithm is founded in information retrieval techniques unlike the prior approaches of shingling. Shingling approaches take a set of terms and create a hash value, such that the first term t_1 to the n^{th} term t_n are used to create a hash value. Then the second term t_2 to the $(n+1)^{-st}$ term t_{n+1} are used to create the next hash value and so on. N-1 shingles represent the document. The shingles are used to determine the percentage of overlap between documents. Shingling approaches have a complexity of $O(n^2)$ (where n is the number of terms in the collection) to examine the entire collection and thus must apply random filtration techniques to perform in reasonable amounts of time. Since each document can occur in multiple clusters based on a shingle, analysis of the collection in terms of fuzzy duplicates is difficult. We provided experimental results showing I-Match clustering near duplicate documents is better and faster than the other state of the art shingling algorithms.

There are many efficiency issues for information retrieval systems. Most of the prior efforts focus on data representation to improve the efficiency of the system. We hypothesize that batched updates of collection statistics improves the overall efficiency of dynamic IR systems by delaying work until necessary. While collection statistics are used to improve precision and recall for information systems, very little work has gone

into efficient update approaches to maintain their currency.

One common type of collection statistics used is Inverse Document Frequency (IDF). We hypothesize that the recalculation of idf values for each new document added is not necessary. We present empirical data that show the recalculation of IDF values after an initial training set does not improve the overall precision and recall of the information retrieval system. By reducing the time to recalculate IDF values, the overall performance of the system can be improved. Unnecessary additional work, in a dynamic environment, adversely affects the performance. By finding a good training set of documents, those IDF values may be used by the system for an extended period without having to recompute IDF values for each additional document added to the system. These fundamental issues for reliability and efficiency for information retrieval systems were addressed in this thesis.

5.2 Future Direction

The future directions of this research are divided into the categories enumerated above, the time sensitive dynamic routing approaches, duplicate data detection and collection statistics update frequencies heuristics. The DRS is a time sensitive proactive routing protocol, unlike routed and gated approaches, which passively monitor network links, DRS proactively monitors each host and its communication links. The DRS checks alternate routes before using them to achieve an additional level of fault tolerance without the use of special hardware. Since this fault tolerance comes at the price of network bandwidth usage, future research for this system will optimize network bandwidth usage either via intelligent multicast status checks or other more efficient means of checking a

large number of servers, i.e., lower than n*(n-1) messages. In addition, we will explore the use of RIP II and gated in conjunction with DRS to add support for hosts outside of the network.

We presented a new algorithm called I-Match for the task of identifying duplicate documents for web documents and other types of text documents. I-Match uses collection statistics to select the best terms to represent the document. For future work, further analysis and testing of our duplicate detection scheme against a collection with known duplicate document sets (relevance judgments of duplicates) is desirable. We do not know of such a test collection but plan to continue testing against such collections as available. In addition, currently we select terms-only, statistical or noun phrases have been shown to be important in information retrieval and could make a better document representation than the simple term-based analysis. Finally, experimentation on larger document collections, such as the 10GB and 100GB collections of web documents from TReC are planned.

We investigated the effect of infrequent updates to the inverse document frequency retrieval strategies. Most research systems simply update the inverse document frequencies with the addition of each new document. Avoiding the need to update *idf's* saves enormous computational resources and *significantly* reduces resources needed to deploy a relational search engine or dynamic collection search engine. We demonstrated the somewhat counterintuitive result that updates to *idf's* do not significantly impact effectiveness on moderately sized document collections. Efforts are required that study means to determine the number of unique terms that are required to accurately represent

the entire document collection. This term set is likely to be based on the size of the total document collection and the rate of introduction of new terms per document. It is also likely to be the case that a training set should include different sample sizes based on domains. The TReC collection contains several different sources, and we are interested in evaluating the effect of different training sets of different sources.

APPENDIX

Table 13. I-Match WT2G Experiments (page 1 of 2)

Experiment	Post Doc Num	Filtered	Pre Avg Dist Terms	Post Avg Dist Terms	Pre Doc Size	Post Doc Size	Dup Hash	Dup URL	Dup Total	Dup clusters	Max Cluster	Time
Baseline	247491	0	471	471	2085	2085	60	0	60	46	5	62
Doc-h-f1	247348	143	471	471	2085	2085	53535	0	53535	22019	782	6869
Doc-h-f2	247348	143	471	471	2085	2085	54398	0	54398	22243	782	6086
Doc-h-f3	247348	143	471	471	2085	2085	55578	0	55578	22484	782	5707
Doc-h-f4	247348	143	471	471	2085	2085	56756	0	56756	22710	782	5376
Doc-h-f5	247348	143	471	471	2085	2085	58235	0	58235	22975	782	5287
Doc-h-f6	247348	143	471	471	2085	2085	60905	0	60905	23520	782	3701
Doc-h-f7	247348	143	471	471	2085	2085	65183	0	65183	23926	782	3415
Doc-h-f8	247348	143	471	471	2085	2085	75169	0	75169	24286	1805	3259
Doc-h-f9	247348	143	471	471	2085	2085	108157	0	108157	22957	4288	3104
Doc-l-f1	247348	143	471	471	2085	2085	52280	0	52280	21747	782	4311
Doc-l-f2	247348	143	471	471	2085	2085	52331	0	52331	21753	782	4100
Doc-l-f3	247348	143	471	471	2085	2085	52389	0	52389	21776	782	3810
Doc-l-f4	247348	143	471	471	2085	2085	52498	0	52498	21831	782	3710
Doc-l-f5	247348	143	471	471	2085	2085	52712	0	52712	21928	782	3568
Doc-l-f6	247348	143	471	471	2085	2085	53196	0	53196	22067	782	5439
Doc-l-f7	247348	143	471	471	2085	2085	54079	0	54079	22306	782	4668
Doc-l-f8	247348	143	471	471	2085	2085	55788	0	55788	22801	782	3317
Doc-l-f9	247348	143	471	471	2085	2085	60053	0	60053	24268	1200	3174
IDF-h-f1	244348	3143	471	476	2085	2107	126710	0	126710	31065	782	2473
IDF-h-f2	247265	226	471	472	2085	2086	60858	0	60858	23636	782	3012
IDF-h-f3	247317	174	471	471	2085	2085	56616	0	56616	22914	782	3551
IDF-h-f4	247339	152	471	471	2085	2085	54864	0	54864	22576	782	3924
IDF-h-f5	247341	150	471	471	2085	2085	54130	0	54130	22358	782	4081
IDF-h-f6	247343	148	471	471	2085	2085	53466	0	53466	22012	782	4115
IDF-h-f7	247344	147	471	471	2085	2085	53107	0	53107	21925	782	4266
IDF-h-f8	247345	146	471	471	2085	2085	52816	0	52816	21900	782	4383
IDF-h-f9	247345	146	471	471	2085	2085	52574	0	52574	21829	782	4472

Table 13. I-Match WT2G Experiments (page 2 of 2)

Experiment	Post Doc Num	Filtered	Pre Avg Dist Terms	Post Avg Dist Terms	Pre Doc Size	Post Doc Size	Dup Hash	Dup URL	Dup Total	Dup clusters	Max Cluster	Time
IDF-I-f1	247318	173	471	471	2085	2086	52391	0	52391	21813	782	4399
IDF-I-f2	246870	621	471	472	2085	2088	52732	0	52732	22022	782	3773
IDF-I-f3	245724	1767	471	474	2085	2097	53372	0	53372	22437	423	3133
IDF-I-f4	244098	3393	471	475	2085	2101	55271	0	55271	22994	238	2897
IDF-I-f5	240678	6813	471	479	2085	2122	58963	0	58963	24275	238	2645
IDF-I-f6	227196	20295	471	492	2085	2171	57403	0	57403	24995	130	2696
IDF-I-f7	203845	43646	471	519	2085	2285	50840	0	50840	24132	30	2535
IDF-I-f8	165898	81593	471	568	2085	2497	35423	0	35423	20139	10	2430
IDF-I-f9	114434	133057	471	616	2085	2724	13242	0	13242	11696	2	2608
DocR-O-f1f9	247348	143	471	471	2085	2085	57316	0	57316	23226	782	4050
DocR-O-f2f8	247348	143	471	471	2085	2085	54248	0	54248	22293	782	3576
DocR-O-f3f7	247348	143	471	471	2085	2085	53201	0	53201	22046	782	3869
DocR-O-f4f6	247348	143	471	471	2085	2085	52556	0	52556	21849	782	4211
DocR-I-f1f9	247200	291	471	472	2085	2086	53726	0	53726	22087	782	5267
DocR-I-f2f8	247200	291	471	472	2085	2086	54743	0	54743	22294	782	4043
DocR-I-f3f7	246907	584	471	472	2085	2087	55840	0	55840	22525	782	3517
DocR-I-f4f6	245740	1751	471	474	2085	2090	57865	0	57865	22963	798	3250
IDFR-O-f1f9	244743	2748	471	476	2085	2106	88641	0	88641	25354	676	2622
IDFR-O-f2f8	247291	200	471	472	2085	2086	57349	0	57349	22981	782	2991
IDFR-O-f3f7	247334	157	471	471	2085	2085	54415	0	54415	22472	782	3577
IDFR-O-f4f6	247346	145	471	471	2085	2085	53030	0	53030	22173	782	4088
IDFR-I-f1f9	247293	198	471	472	2085	2086	52703	0	52703	21905	782	4112
IDFR-I-f2f8	246759	732	471	473	2085	2088	53395	0	53395	22245	782	3419
IDFR-I-f3f7	245457	2034	471	475	2085	2099	55295	0	55295	22997	423	2921
IDFR-I-f4f6	241515	5976	471	479	2085	2117	62341	0	62341	24360	238	2615

Table 14. I-Match Experiment Legend

I-Match Experiment	Description
Baseline	Syntactic one-pass hash approach, stemming and removable of special character terms.
Doc (% Doc approach)	Takes the X percent of the document based on idf values of the terms. **l** = Highest on the left side of tree. So the terms withthe X highest idf values are used. **h** = Lowest on the left side of tree. So the terms with the X lowest idf values are used.
IDF (IDF approach)	Filters terms that don't meet the normalized idf value threshold are removed. **l** = Terms with idf value is greater than the filter value, the term are kept. **h** = Terms with idf value is lower than the filter value, the term are kept.
DocR (%Doc Range approach)	Takes the X percent of the document based on idf values of the terms. The range takes either the middle X percent or the outer X percent based on the l or h value. **I** = The inner X percent of terms based on idf values are kept. **O** = The outer X percent of terms based on idf values are kept.
IDFR (IDF range approach)	Filters terms based on normalized idf values. Thus if the term is in the range of idf values it is kept for the final hash. **I** = Keeps terms with idf values between the two values **O** = Keeps terms with idf values greater and less than the 2 filter values.

Table 15. WT2G DSC-SS Experiments

	Post Num Docs	Filtered	Pre Dist Shingles	Post Dist Shingles	Pre Doc Size	Post Doc Size	Dup Clusters	Time
DSC-SS-2	239886	7605	470	484	2079	2140	41001	23465
DSC-SS-4	229609	17882	470	503	2079	2225	35638	23441
DSC-SS-5	224157	23334	470	513	2079	2272	33771	26064
DSC-SS-10	192202	55289	470	578	2079	2575	27224	24984
DSC-SS-15	160126	87365	470	659	2079	2963	22667	23609
DSC-SS-20	133711	113780	470	746	2079	3389	19183	24318
DSC-SS-25	112800	134691	470	833	2079	3833	16249	23669
DSC-SS-50	57343	190148	470	1255	2079	6109	7546	25366
DSC-SS-75	36486	211005	470	1626	2079	8123	4948	24147
DSC-SS-100	26341	221150	470	1926	2079	9860	3673	26084

BIBLIOGRAPHY

[1] Morrison P., Morrison P., "The Sum of Human Knowledge?", Scientific America, Vol. 279, No. 1, http://www.sciam.com/1998/0798issue/0798wonders.html, July 1998.

[2] Lu Z., McKinley K., "Partial Replica Selection Based on Relevance for Information Retrieval", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 97-104, August 1999.

[3] Ribeiro-Neto B., Ziviani N., Moura E., Neuber M., "Efficient Distributed Algorithms to Build Inverted Files", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 105-112, August 1999.

[4] French J., Powell A., Callan J., Viles C., Emmitt T., Prey K., Mou Y., "Comparing the Performance of Database Selection Algorithms", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 238-245, August 1999.

[5] Baumgarten C., "A Probabilistic Solution to the Selection and Fusion Problem in Distributed Information Retrieval", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 246-253, August 1999.

[6] Xu J., Croft B., "Cluster-based Language Models For Distributed Retrieval", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 254-261, August 1999.

[7] Kretser O., Moffat A., "Efficient Document Presentation with a Locality-Based Similarity Heuristic", Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 113-120, August 1999.

[8] Nau D., Ball M., Baras J., Chowdhury A., Lin E., Meyer J., Rajamani R., Splain J., Trichur V., "Generating and Evaluating Designs and Plans for Microwave Modules." AI in Engineering Design and Manufacturing, 2000. To appear.

[9] Chowdhury A., Nicklas L., Setia S., White L., "Supporting Dynamic Space-sharing on Clusters of Non-dedicated Workstations", ACM 17th Int. Conference on Distributed Computing Systems, 1997.

[10] Setia S., Chowdhury A., Nicklas L., White E., "Supporting Dynamic Reconfiguration of Parallel Applications on Clusters of Non-dedicated Workstations", Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.

[11] Chowdhury A., "Dynamic Reconfiguration: Checkpointing Code Generation", IEEE 5th International Symposium on Assessment of Software Tools and Technologies (SAST97), 1997.

[12] Chowdhury A., Luse P., Frieder O., Wan P., "Network Survivability Simulation of the Commercially Deployed Dynamic Routing System Protocol", IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, May 2000.

[13] Chowdhury A., Burger E., Grossman D., Frieder O., "DRS: A Fault Tolerant Network Routing Protocol over IP", IEEE-IC3N Sixth International Conference On Computer Communications and Networks, 1997.

[14] Goharian N., Chowdhury A., Grossman D., Ghazawi T., "Efficiency Enhancements for Information Retrieval using a Sparse Matrix Approach" Parallel and Distributed Processing Techniques and Applications, June 2000.

[15] Holmes D., McCabe C., Chowdhury A., Alford K., Grossman D., Frieder O., "Improved Manual Query Processing and Using Stemming Equivalence Classes as a Basis for Relevance Feedback", NIST Text Retrieval Conference, November 1999.

[16] Holmes D., McCabe C., Grossman D., Chowdhury A., Frieder O., "Use of Query Concepts and Information Extraction to Improve Information Retrieval Effectiveness", NIST Text Retrieval Conference, November 1998.

[17] Grossman D., Lundquist C., Reichart J., Holmes D., Frieder O., Chowdhury A., "Overview of the Fifth Text REtrieval Conference", NIST Text Retrieval Conference, December 1996.

[18] Chowdhury A., McCabe C., "Performance Improvements to Vector Space Information Retrieval Systems with POS", TR-98-48, Institute of Systems Engineering UMD, June 1998.

[19] Frieder O., Grossman D., Chowdhury A., Frieder G., "Phrase Processing with the Relational Model and IDF Update Effects on Precision/Recall", IBM Technical Report 05.500, April 1997, (IBM Confidential until May 2002)

[20] Frieder O., Grossman D., Chowdhury A., Frieder G., "Efficiency Considerations in Very Large Information Retrieval Servers," Journal of Digital Information, December 1999.

[21] Anderson T., Culler D., Patterson D., "A Case for NOW (Networks of Workstations)." IEEE Micro 15(1):54-64, February 1995.

[22] Casas J., Konuru R., Otto S., Prouty R., Walpole J.. "Adaptive Load Migration Systems for PVM". In Proceedings of Supercomputing 94, Pages 390-399, Washington D.C., November 1994.

[23] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard", International Journal of Supercomputer Applications and High Performance Computing, 8(3/4), 1994. Special issue on MPI. Also available electronically, the url is ftp://www.netlib.org/mpi/mpi-report.ps.

[24] Reschke C., Sterling T., Ridge D., Savarese D., Becker D., Merkey P., ‘‘A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation,”, Proceedings of the Fifth IEEE Symposium on High Performance Distributed Computing, 1996

[25] Hamilton, Kougiouris, "The Spring Nucleus: A Microkernel for Objects", Sun Technical Reports, TR-93-14 (April 1993)

[26] Dees W., Smith R., "Performance of Interconnection Rip-Up and Reroute Strategies," Proceedings 18th Design Automation Conference, June 1981, pp. 382-390.

[27] Bahk S., Zarki M., "Dynamic Multi-Path Routing and How it Compares with Other Dynamic Routing Algorithms for High Speed Wide Area Networks", Proc. SIGCOMM ‘92, pp. 53-64, 1992.

[28] Ash G. R., Dynamic Routing in Telecommunications Networks, McGraw Hill, New York, NY, 1998.

[29] Krishnan K. R. Doverspike R. D. Pack C. D. "Improved survivability with multilayer dynamic routing," IEEE Communications Magazine, pp. 62--68, July 1995.

[30] Hurley B. R. Seidl C. J. R. Sewell W. F. "A Survey of Dynamic Routing Methods for Circuit-Switched Traffic". IEEE Communications Magazine, 25(9), September 1991.

[31] Hedrick C. Request For Comment 1058, "Routing Information Protocol", 06/01/1988, http://ds.internic.net/ds/dspg2intdoc.html

[32] Malkin, "RIP Version 2 Carrying Additional Information", RFC-1388, 01/06/1993, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[33] Dees W., Smith R., "Performance of Interconnection Rip-Up and Reroute Strategies," Proceedings 18th Design Automation Conference, pp. 382-390, June 1981.

[34] Moy J., Request For Comment 1583, "OSPF Version 2", 03/23/1994, http://ds.internic.net/ds/dspg2intdoc.html

[35] Internet Architecture Board, "Applicability Statement for OSPF", RFC-1370, 10/23/1992, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[36] Moy J., "Experience with the OSPF Protocol:, RFC-1246, 8/8/1991, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[37] Baker, Coltun, "OSPF Version 2 Management Information Base", RFC-1253, 08/30/1991, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[38] Mills, "Exterior Gateway Protocol EGP", RFC-904, 10/01/1982, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[39] Lougheed, Rekhter, "Border Gateway Protocol (BGP)", RFC-1105, 06/01/1989, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[40] Lougheed, Rekhter, "A Border Gateway Protocol (BGP)", RFC-1163, 06/20/1990, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[41] Varadhan, "BGP OSPF Interaction", RFC-1503, 01/14/1993, http://ds.internic.net/ds/dspg2intdoc.html

[42] Honig, Katz, Mathis, Rekhter, Yu, "Application of the Border Gateway Protocol in the Internet", RFC-1164, 06/20/1990, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[43] Rekhter, "BGP Protocol Analysis", RFC-1265, 10/28/1991, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[44] Lougheed, Rekhter, "A Border Gateway Protocol 3 (BGP-3)", RFC-1267, 10/25/1991, http://info.internet.isi.edu/in-notes/rfc/files, July 17, 1998

[45] Low S., Varaiya P., "Stability of a class of dynamic routing protocols (IGRP)". In IEEE Proceedings of the INFOCOM, volume 2, pages 610--616, March 1993.

[46] Chowdhury A., Frieder O., Burger E., Grossman D., Makki K., “Dynamic Routing System (DRS): Fault tolerance in network routing”, Computer Networks and Isdn Systems (31) 1-2 (1999) pp. 87-97.

[47] Key P. B., Cope G. A.. "Distributed dynamic routing schemes". IEEE Communications Magazine, vol. 28:54-64, October 1990.

[48] Hurley B. R., Seidl C. J. R., Sewell W. F.. "A Survey of Dynamic Routing Methods for Circuit-Switched Traffic". IEEE Communications Magazine, 25(9), September 1991.

[49] Peha, Tobagi, "Analyzing the Fault Tolerance of Double-Loop Networks", IEEE/ACM Transactions on Networking, Vol. 2, No. 4, August 1994.

[50] Wu, "A Passive Protected Self-Healing Mesh Network Architecture and Applications", IEEE/ACM Transactions on Networking, Vol. 2, No. 1, February 1994.

[51] Kelly F.P., "Bounds on the Performance of Dynamic Routing Schemes for Highly Connected Networks". Mathematics of Operations Research, 19:1--20, 1994.

[52] Bahk S., Zarki M. E.. "Dynamic Multi-Path Routing and how it Compares with other Dynamic Routing Algorithms for High Speed Wide Area Networks". In Proceedings of ACM Sigcomm, August 1992.

[53] Gibbens R. J., Kelly F. P., "Dynamic Routing in Fully Connected Networks", IMA Journal of Math. Control & Information. 1990, 7, 77-111.

[54] Talbott R., “Network Survivability Analysis”, Fiber and Integrated Optics, Volume 8, pp. 13-43, 1988.

[55] Postel J., "Internet Control Message Protocol (ICMP)", RFC 792, 1981

[56] Shivakumar N., Garca-Molina H., "Finding near-replicas of documents on the web", In Proceedings of Workshop on Web Databases (WebDB'98), March 1998.

[57] Broder A., Glassman S., Manasse M., Zweig G., "Syntactic Clustering of the Web", Sixth International World Wide Web Conference, April, 1997.

[58] Heintze N., "Scalable Document Fingerprinting", Proceedings of the Second USENIX Workshop on Electronic Commerce, 1996.

[59] NCCAM, http://nccam.nih.gov/, The National Institutes of Health (NIH), National Center for Complementary and Alternative Medicine (NCCAM), April 12, 2000.

[60] Giles L., Lawrence S., "Accessibility and Distribution of Information on the Web". Nature vol 400, 1999.

[61] Lawrence S., Giles L., "Searching the World Wide Web", Science, Volume 280, Number 5360, p. 98, 1998.

[62] Brin S., Davis J., Garcia-Molina H., "Copy Detection Mechanisms for Digital Documents", Proceeding of SIGMOD '95, 1995.

[63] Buckley C., Cardie C., Mardis S., Mitra M., Pierce D., Wagstaff K., Walz J., "The Smart/Empire TIPSTER IR System", TIPSTER Phase III Proceedings, Morgan Kaufmann, San Francisco, CA, 2000.

[64] Salton G., Yang C.S., Wong A.. "A Vector-Space Model for Information Retrieval", Comm. of the ACM, 18, 1975.

[65] Kjell B., Woods A, Frieder O., "Discrimination of Authorship Using Visualization", Information Processing and Management, Pergamon Press, 30(1), pp. 141-150, January 1994.

[66] Scotti R. Lilly C., George Washington University Declassification Productivity Research Center, http://dprc.seas.gwu.edu, July 31, 1999.

[67] Grossman D., Holmes D., Frieder O., "A DBMS Approach to IR in TREC-4", Text REtrieval Conference (TREC-4), November 1995.

[68] Smeaton A., Kelledy F., Quinn G., "Ad Hoc Retrieval Using Thresholds, WSTs for French Monolingual Retrieval, Document-at-a-Glance for High Precision and Triphone Windows for Spoken Documents", Proceedings of the Sixth Text Retrieval Conference (TREC-6), 1997, p.461.

[69] NIST, SECURE HASH STANDARD, U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, FIPS PUB 180-1, 1995 April 17.

[70] Chowdhury A., Holmes D., McCabe C., Grossman D., Frieder O., "Improved Query Precision using a Unified Fusion Model", NIST Text Retrieval Conference, November 2000.

[71] SMART FTP site: "ftp://ftp.cs.cornell.edu/pub/smart/", January 19, 2000.

[72] Baeza-Yates R., Ribeiro-Neto B., Modern Information Retrieval, Addison Wesley, Reading, MA 1999.

[73] Porter M., "An Algorithm for Suffix Stripping," Program, 14(3):130-137, 1980.

[74] Viles C., French J., "On the Update of Term Weights in Dynamic Information Retrieval Systems", Conference on Information and Knowledge Management (CIKM95), Baltimore MD, Nov 29 - Dec 2, 1995.

[75] Blair D., Maron M., "An Evaluation of Retrieval Effectiveness for a Full-Text Document- Retrieval System", Communications of the ACM, 28(3):289-299.

[76] Salton G., Wong A., Yang C.S., "A Vector Space Model for Automatic Indexing", Communications of the ACM, pp. 613-620.

[77] Salton G., "A Comparison Between Manual and Automatic Indexing Methods", Journal of American Documentation, 1969, 20(1):61-71.

[78] Salton G., "Automatic Text Analysis", Science, 1970, 168(3929):335-342.

[79] Robertson S., Spark Jones K., "Relevance weighting of search terms". Journal of the American Society for Information Science, 27:129--146, 1976.

[80] Salton G., Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[81] Salton G., Buckley C., "Term Weighting Approaches in Automatic Text Retrieval", Information Processing and Management 24(5), pages 513--523, 1988.

[82] Singhal A., Buckley C., Mitra M., “Pivoted Document Length Normalization,” Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, August 18-22, 1996.