CITI-TR-90-6 November 8, 1990

Analysis of X.500 Distributed Directory Refresh Strategies

Guy A. Fasulo
Kevin H. Klinge
Sailesh Makkapati
David W. Bachmann
Michael H. Kamlet
Toby J. Teorey
Michael A. Bauer
J. Michael Bennett

Center for Information Technology Integration (CITI)
The University of Michigan, Ann Arbor, MI 48105-2016

Center for Information
Technology Integration

The University of Michigan
Information Technology Division
535 W. William
Ann Arbor, MI 48103-4943

Analysis of X.500 Distributed Directory Refresh Strategies

Guy A. Fasulo
Kevin H. Klinge
Sailesh Makkapati
David W. Bachmann
Michael H. Kamlet
Professor Toby J. Teorey, Project Director

Michael A. Bauer
J. Michael Bennett
Computer Science Department
University of Western Ontario

Sponsor: IBM Toronto Laboratories (Center for Advanced Studies)

CITI Technical Report 90-6
Center for Information Technology Integration (CITI)
The University of Michigan
535 W. William Street
Ann Arbor, Michigan 48103-4943
313-747-4060

November 1990 (Revised)

Table of Contents

Abstract

Distributed database directory refresh strategies, commonly recommended for the X.500 standard, are defined and analytically modeled for variations on push/pull and total/differential under idealistic asynchronous control conditions. The models are implemented in a HyperCard-based tool called DirMod (for "directory model"). Experimental test results show important elapsed time performance tradeoff among the different strategies, and live test data contribute to the verification of the models.

1. Introduction

Overview

In distributed systems, one major concern is the mapping of names to services. A directory provides this mechanism. The problem of replicating directory information and maintaining the data (updating...) is still being addressed. We choose to study the emerging X.500 standard for directories since Quipu (an implementation of X.500) is readily available [Kille89, KRRT90], and this would be ideal for experimentation to discover the good, the bad and the ugly about directories [BBS89, ISO89a, ISO89b]. The experiments conducted form the basis for the parametric modeling of the refresh strategies.

The X.500 directory system is a set of interconnected open systems which cooperate to provide directory services [Bennettb 89]. X.500 uses attribute-based naming for identification of objects within its name space. The attribute name space is restricted to a hierarchy and one of the primary reasons for doing so is to be able to support a unique (i.e. distinguished) name for each object. X.500 accommodates access to objects using incomplete information by associating a set of attributes with each object and allowing users to browse the attributes at each node in the hierarchy [Bauer 89].

The refresh strategies of the X.500 distributed directory system involve three main parameters:

1) Initiation Source
   - Pull - Shadow
   - Push - Supplier/Master

2) Size - Total vs Differential

3) Consistency - Synch vs Asynch (Synch implies multiple levels of the DSA hierarchy are refreshed. Asynch implies only one level of the hierarchy is refreshed.)

Therefore, the eight strategies are:

1) Pull, Total, Asynch
2) Pull, Diff, Asynch
3) Push, Total, Asynch
4) Push, Diff, Asynch
5) Pull, Total, Synch
6) Pull, Diff, Synch
7) Push, Total, Synch
8) Push, Diff, Synch

This paper includes our analysis on the first four strategies, with asynchronization (asynch) being the common parameter. No analysis on the four synch alternatives has been performed as of the date of this paper; however, the intent is to perform future analysis on them. All of the above asynch alternatives are implemented in a HyperCard-based analytical model (see Sec. 3: The Distributed Directories Modeling Tool).

Basic Definitions

The following is a list of basic terms in X.500 with definition for each [ISO89a]:

1) DIT: Directory Information Tree (Database). Each node in the DIT refers to a directory.

2) DSA: Directory Service Agent -- a server responsible for manipulating the directory data.

3) Master DSA: This DSA has administrative authority over a set of data. Updates (modification to data) can only be done at the master.

4) Shadow DSA: This DSA maintains a read-only copy of the data and must get changes from the master. Also called a slave, consumer, or shadow-consumer.

5) Supplier DSA: Cannot update itself, but can update shadows (acts like a master and a shadow)[1].

6) Refresh: The process where the shadow receives a current copy of the data.

7) Pull: Shadow initiated refresh. Update schedule held at shadow. (This is the implementation of Quipu).

8) Push: Master initiated refresh. Update schedule held at master.

9) Total refresh: The entire data set (whole table) is sent to the shadow. (This is the implementation of Quipu).

10) Differential refresh: The set of changes to the data is sent to the shadow.

11) Synch: Multi-level refresh. In the case of a Push - all copies of the data will be consistent. In the case of a Pull if a shadow receives refreshes from a supplier, then the supplier AND shadow will be refreshed.

12) Asynch: Single-level refresh. In the case of a Push only shadows of the master will be refreshed. (This is the implementation of Quipu).

13) Snap Time: The last time the shadow received a refresh. In Quipu this is implemented with a version number.

To keep the terminology consistent, DSAs will be referred as either a master or shadow, although the DSA could be a supplier in any of the alternatives discussed.

---

[1] Supplier DSAs arise when the DSA hierarchy contains more than two levels. The top level is occupied by the master and the bottom level is occupied by the shadow(s). Suppliers lie between the master and shadow levels; therefore, a supplier is considered the master of the shadow(s) beneath it, and the shadow of the master above it.

2. Basic Steps in the Refresh Process

Asynch Pull

The basic steps followed in the Asynch Pull refresh process are:

1) Shadow sends message to Master indicating it desires a refresh.

2) The Master processes the message (identifies which shadow is making the request, determines if there been any changes to the data since the last refresh to the requesting Shadow, etc), and determines if a refresh is warranted (If time of last change to data is later than snap time).

3) If a refresh is warranted, the Master prepares the data for transmission (selection and packing time). If a refresh is not warranted, then the Master returns a message indicating so. The Differential algorithm may take longer to pack up the data because each record's time stamp is compared to the shadow's snap time. Preparation includes encoding into ASN.1

4) The data is transmitted to the Shadow. The data file is first transferred to the CP (Communication Processor), and then the CP transmits the data file over the network.

5) Shadow receives the data, directly placing it in main memory. This includes decoding the data and placing the data into a tree structure. Shadow is locked out from other users during the refresh operation.

6) Shadow backs up memory by copying data to disk. Shadow writes out entire data set in both Total and Differential cases in this example. Quipu is implemented on a Unix platform and maintains the data file in virtual memory, since the data file usually exceeds the main memory capacity of the computer. A copy of the file is written to disk to serve as a backup in the event the Master is not accessible. In general, Unix based systems do not provided any form of indexed file structure; therefore, Quipu provides the simple solution: rewrite the entire data file.

Note: The shadow does not acknowledge the receipt of refresh.

We assume the CPU processing time to construct a request message to be insignificant; therefore, it is not considered in the analysis.

Asynch Push

The basic steps followed in the Asynch Push refresh process are:

1) When a refresh is warranted, the Master prepares the data for transmission (selection and encoding time). The Differential algorithm may take longer because each record's time stamp is compared to each shadow's snap time.

2) The data is transmitted to the Shadow(s) sequentially one after another. The data file is first transferred to the CP (Communication Processor), and then the CP transmits the data file over the network.

Steps 3, 4 and 5 occur in parallel.

3) Shadow(s) receives the data, directly placing it in main memory. Decoding and recreating the tree structure.

4) Shadow(s) backs up memory by copying data to disk. As in Asynch Pull, shadow writes out entire data set in both Total and Differential cases.

5) Shadow(s) sends acknowledge message to Master.

6) Master reads the acknowledgement message(s).

We assume the CPU processing time to construct an acknowledgement message to be insignificant; therefore, it is not considered in the analysis.

The real elapsed time is the real time cost to complete the Push refresh process. The "start" time is the moment the master begins the refresh process. The "finish" time is moment the master finishes reading the last acknowledgement message from a shadow. The real elapsed time is the difference between the start and finish times.

Although the real elapsed time may vary, a lower bound estimate can be calculated (a best case time).

Legend

RET = Real Elapsed Time.
ST = Shadow Time. Time to refresh one shadow (Summation of Steps 1 thru 6 in Push).
NRS = Number of Remaining Shadows. Total number of shadows - 1.
MT = Message Time. Time to read/process acknowledgement message from shadow (Step 6 in Push).
CPT = Communication Processor transfer Time. This is the time it takes the master to transfer the data file from memory of the CPU to the memory buffer of the communication processor (Part of Step 2 in Push).

The Best case time is calculated by:

$$RET = ST + NRS * Max(MT, CPT)$$

After the first acknowledgement message, the remaining acknowledgement messages will arrive at time intervals of time equal to CPT. If MT < CPT then the master is waiting for the next message to arrive. If MT > CPT then the master will still be processing the previous message and the current message sits in the queue waiting to be processed. Hence Max (MT, CPT).

3. The Distributed Directories Modeling Tool (DirMod)

NetMod Basis

The Network Modeling Tool (NetMod), a HyperCard-based tool, uses simple analytical models to provide the designers of large interconnected local area networks with an in-depth analysis of the potential performance of these systems. It assists the development of an understanding of the performance of state-of-the-art local area network (LAN) technologies in either a university, industrial, or government campus networking environment consisting of thousands of computer sites. A principal application of the tool is to help the campus network designer to configure a potential user's data network using proposed hardware and software components.

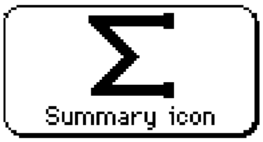
NetMod can analyze an existing or proposed network in terms of its basic performance characteristics, e.g. component utilization, throughput, and packet delay times. It provides a capability for sensitivity analysis of the performance based on changes in the workload parameters and either minor or major changes in the network topology and connectivity. Thus it is possible to quickly evaluate and compare several alternate configurations: a feature which will greatly assist not only the network design process but also distributed directory design.

NetMod provides a graphic interface that corresponds to the world view of the network designer, with icons that represent rings, buses, routers, workstations, etc. This is combined with the quick feedback which is possible using the analytic approach, to provide a capability for interactive analysis of potential network designs, quickly ruling out trouble-prone configurations and identifying potential bottlenecks. The designer does not need to be an expert in any modeling discipline, nor does he/she need to keep a bookshelf lined with the specifications of various network media and protocols. One merely needs to know what devices are present in the networks and how they are connected.

Extensions for Distributed Directories

The Distributed Directories Modeling Tool (DirMod) is a HyperCard extension of the Network Modeling tool (NetMod)[BSST90]. The purpose of DirMod is to model and provide the user with the performance statistics of refresh strategies over various types of networks. NetMod provides all interconnect devices (bridges, routers, gateways, etc...) and their corresponding network statistics. DirMod uses the available network calculations (mainly propagation delay over a network), and incorporates Directory System Agents (DSAs) and their various refresh strategies defined earlier into this tool. The following sections describe the additions to NetMod that have been made that define DirMod (see Appendix A for a list of equations utilized by DirMod to calculate Pull and Push times).

The DSA icon resembles the familiar Bell picture of fingers paging through a directory. The DSA icons have identical connection properties as workstations do in a network, but they contain different attributes based on their purpose in the model. Each DSA has an associated disk random block and sequential block access time, processing speed, average packet size and rate, block size, and background load on its host machine. These properties are used in the calculations for a given refresh algorithm and are easily entered for each DSA by double-clicking on the icon.



The Summary icon is represented by a summation symbol. Its purpose is to actually specify the refresh algorithm performed on a set of existing DSAs. Thus each summary icon provides the refresh attributes of master, slave(s), refresh policy (in terms of Pull/Push, Total/Differential, and Asynchronous/Synchronous), the number of records involved in a refresh from the master, and the record size of each record in the master DSA. Each summary then can capture a refresh policy for DSAs on any part of a network provided they are in some way connected.

Once DSAs have been defined and properly connected over a network, and summaries have been created to specify the refresh algorithm performed on a set of DSAs (see Fig. 1), calculations can be performed as in NetMod.

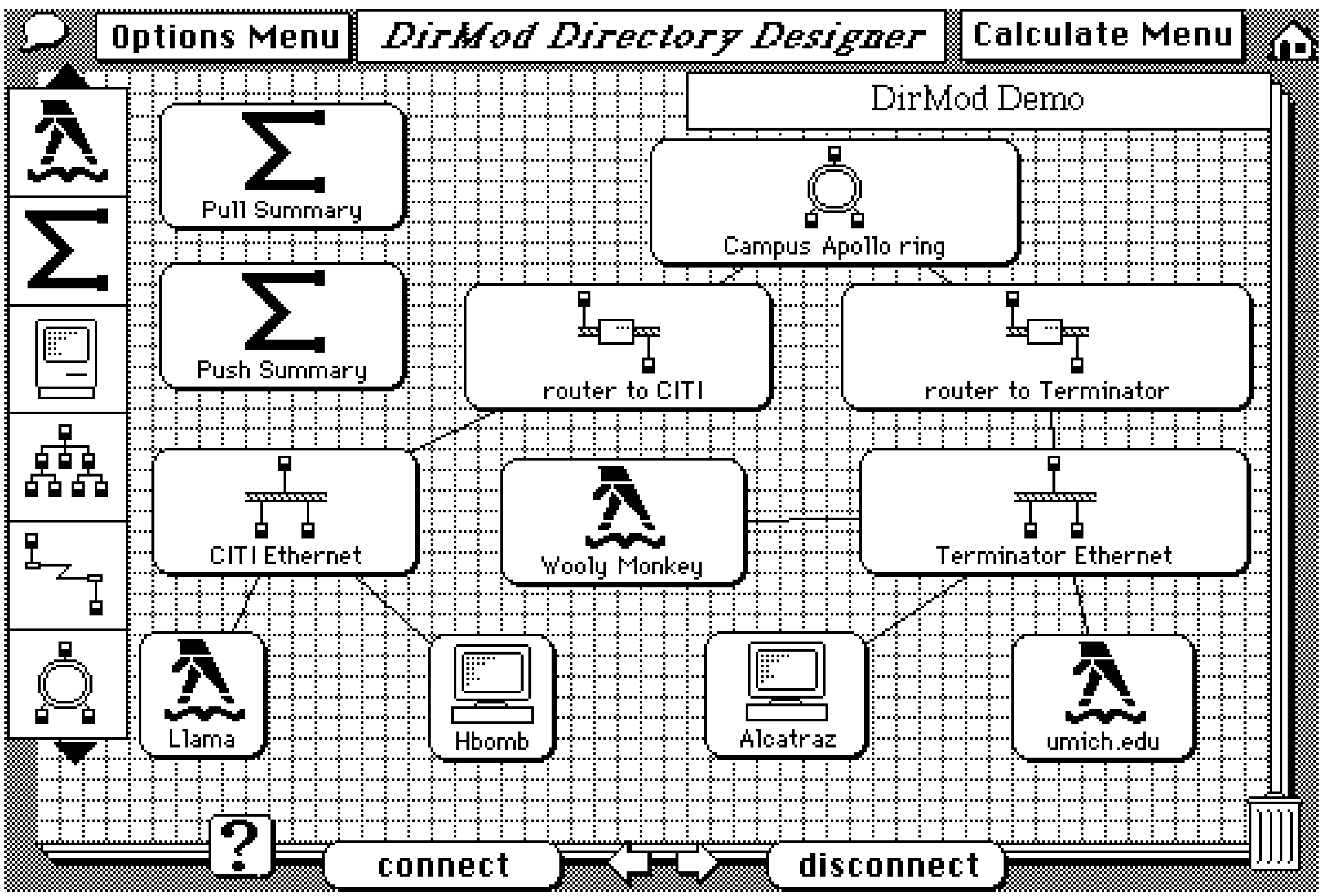


Figure 1 : Basic DSA and Summary setup in DirMod

By turning calculations on, each summary computes master, network, and shadow(s) times based on its refresh policy as shown in Fig. 2.

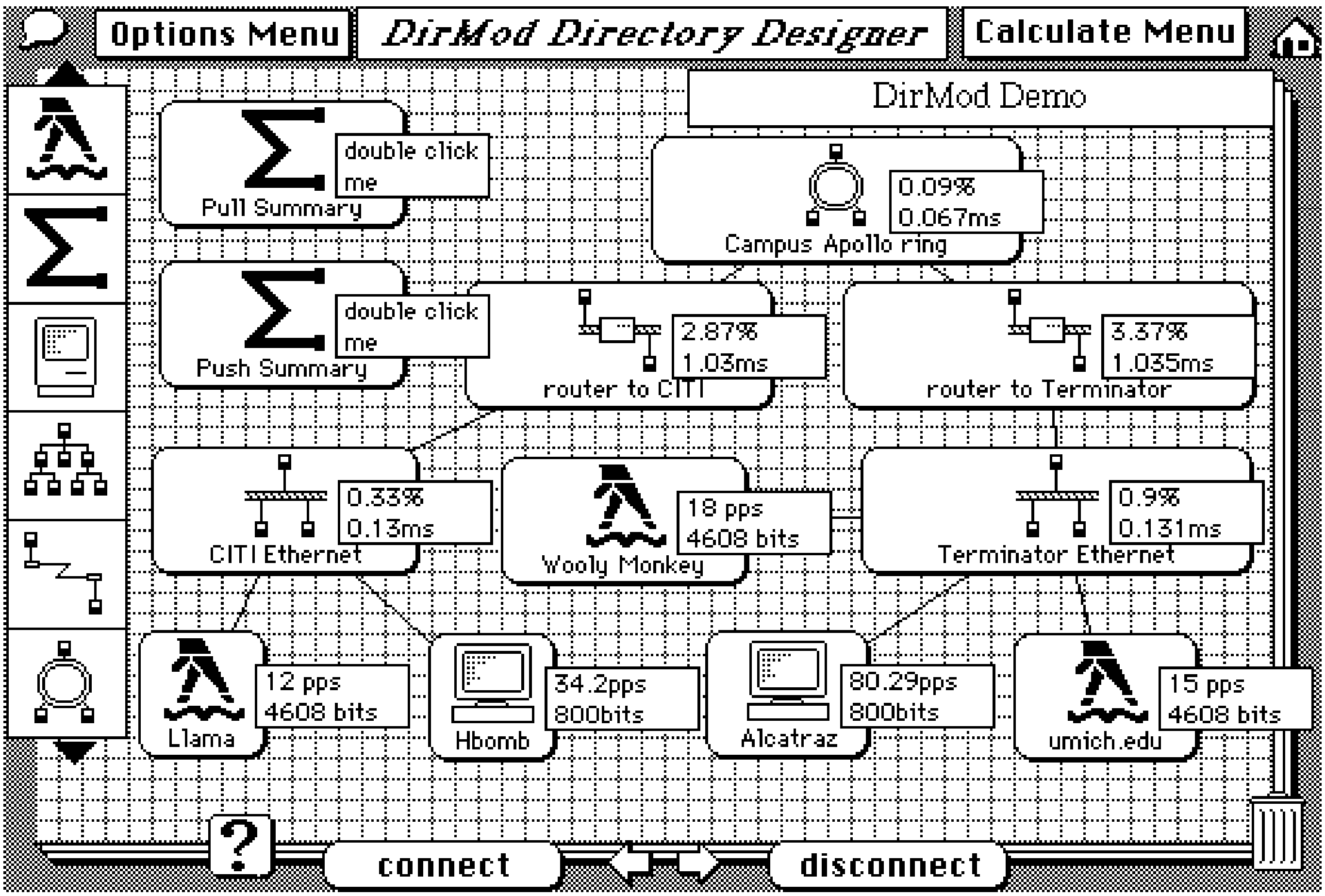


Figure 2 : DirMod screen with Calculations On

By double clicking on a summary's adjacent viewfield on a network diagram in DirMod, the three different times for a refresh calculation can be viewed on a summary statistics page. These results can be compared with other summaries containing the same DSA relationships with different refresh strategies. For example, the summary containing a Push from Wooly Monkey to Llama and umich.edu can be compared to a refresh strategy with a Pull from Llama and umich.edu to Wooly Monkey (see Figs. 3 and 4). Additionally, changes can be made to DSA properties to try and improve the master, shadow(s), and network times for a given relationship summary.

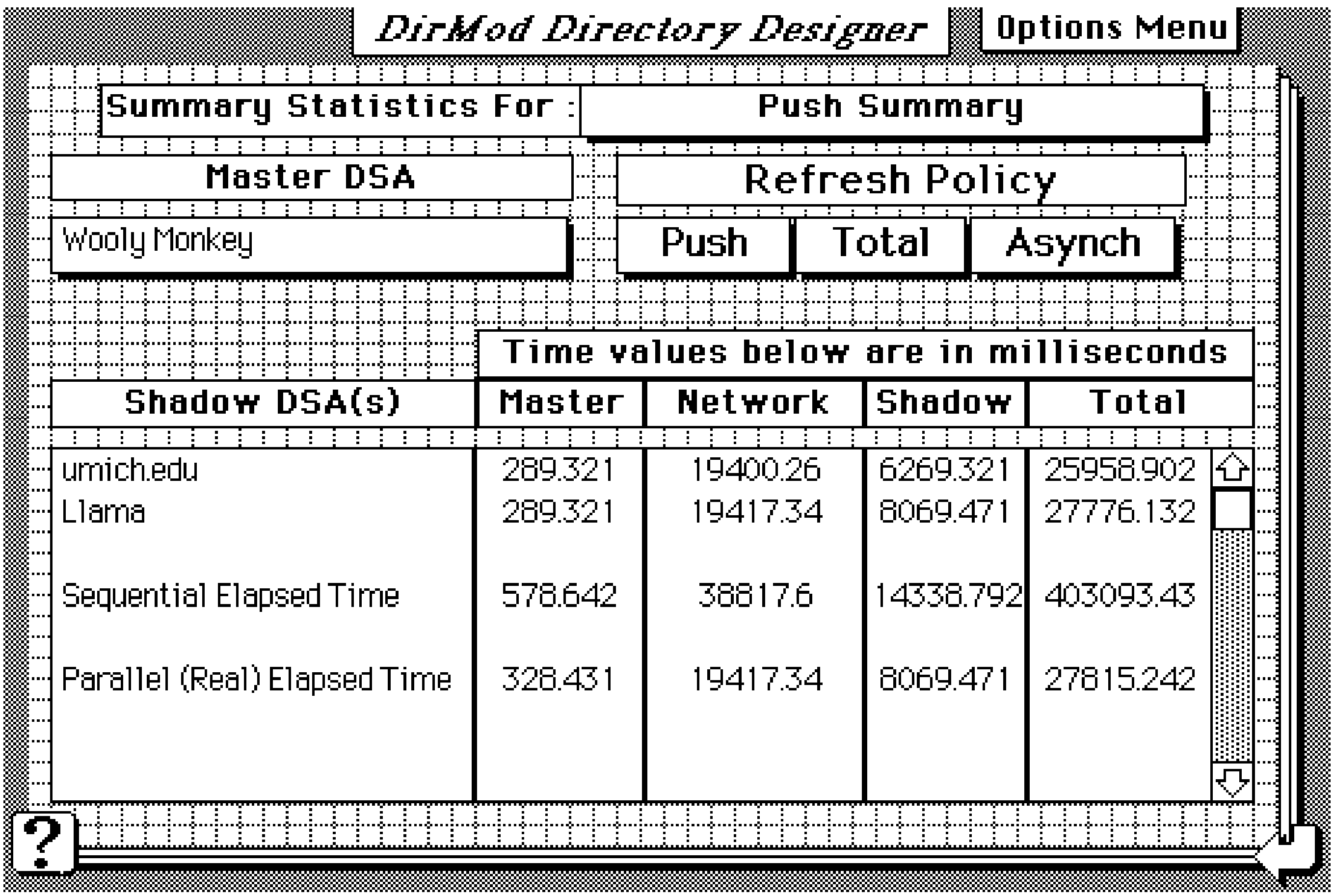


Figure 3 : Summary statistics for a Push Total Asynch

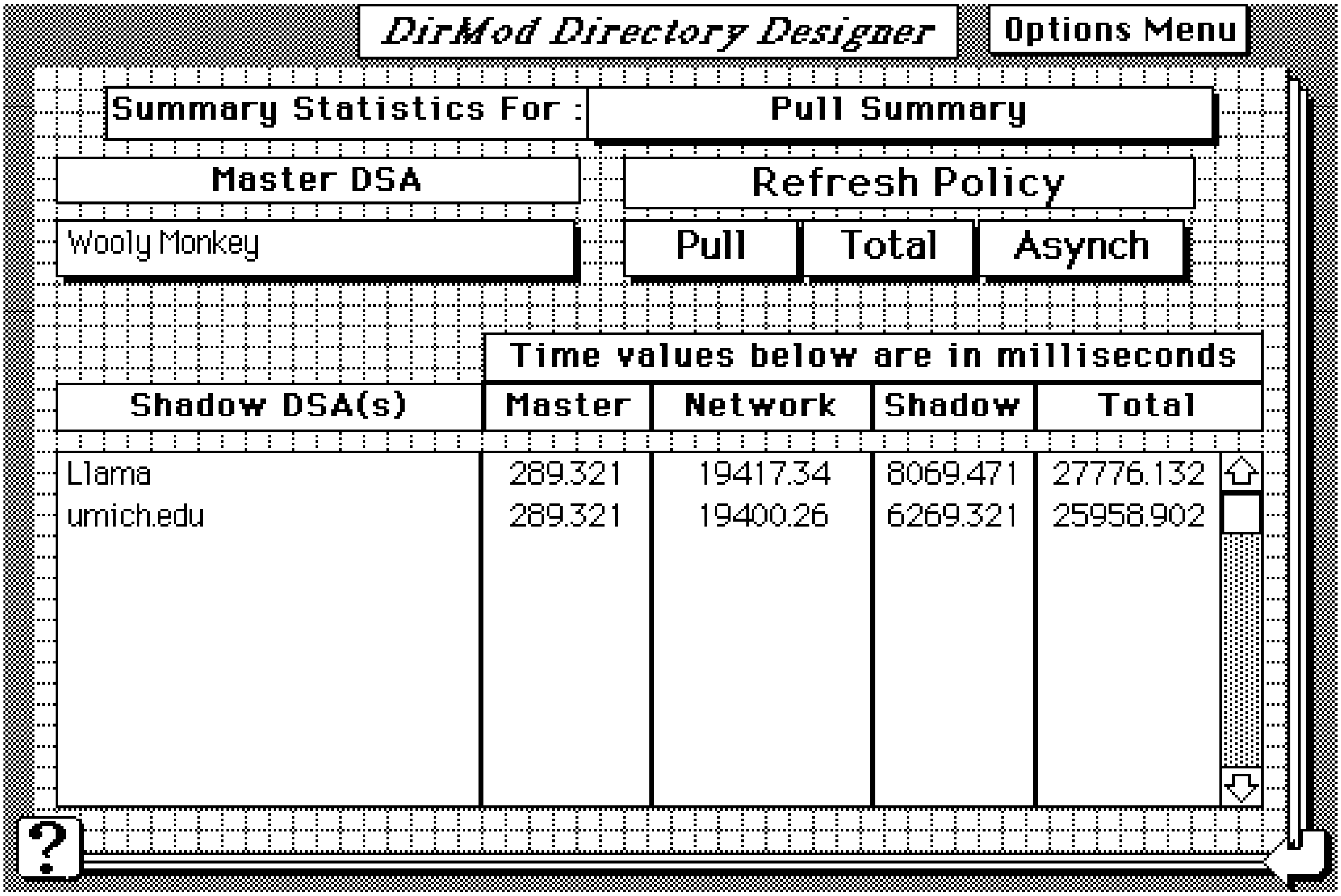


Figure 4 : Summary statistics for a Pull Total Asynch

DirMod, with its NetMod framework and its directory extensions, provides the tools and experimentation grounds to accurately study refresh performance results of real or planned distributed directories.

4. Experiments

Parameters

The following is a list of the parameters involved in the refresh strategies :

1. The file size is 500 records.

2. The block size is 2 KBytes, which results in a blocking factor of 1.

3. The average distance between the supplier and a shadow is 2000 kilometers.

4. The CPU processing time has a fixed instruction count of 1000. This appears in the following steps in our Push and Pull algorithms :
   a) Supplier prepares data (Push and Pull)
   b) Shadow places data into memory (Push and Pull)
   c) Supplier decodes request message (Pull)
   c) Supplier reads acknowledgment message (Push)

5. The instruction count time per record (for encoding/decoding) is 50000 for a total refresh and 55000 for a differential refresh. This occurs for the master encoding and the shadow decoding the data. The instruction count for the shadow to reconstruct the tree is 70000 for a total refresh and 75000 for a differential refresh. We assign the extra 5000 instructions to compensate for the extra DSA CPU time in a differential refresh.

6. The computer speed is 4 MIPS.

7. The channel speed[2] is 5 Megabytes per second. This is considered a conservative lower bound.

---

2 Channel speed is the memory to memory transfer time between the CPU and CP (Communication Processor).

Hypotheses

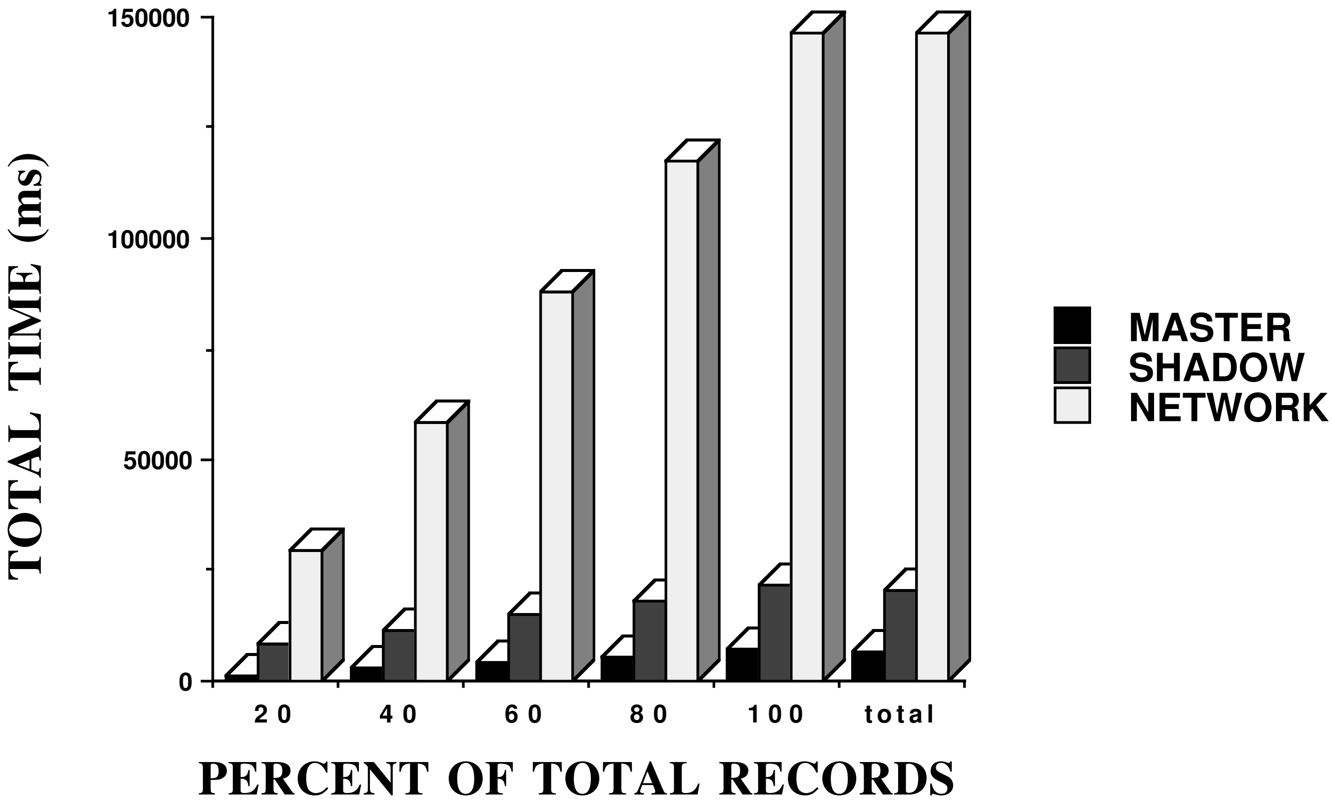
1. Network transmission time is the bottleneck in both Push and Pull algorithms.
2. The Differential refresh algorithm provides significant time savings compared to the Total refresh algorithm in the overall refresh time.
3. The Push refresh algorithm provides a higher degree of parallelism not found in the Pull algorithm.
4. Although the total refresh time for all shadows in a Push, Asynch operation is dependent on the number of shadows, the actual elapsed time is only slightly larger than the total refresh time for one shadow.

Analysis

Based on our model and parameters above, we have calculated times for the various steps and have made some initial observations about the refresh algorithms to support our Hypotheses.

Hypothesis one: Network transmission time is the bottleneck.

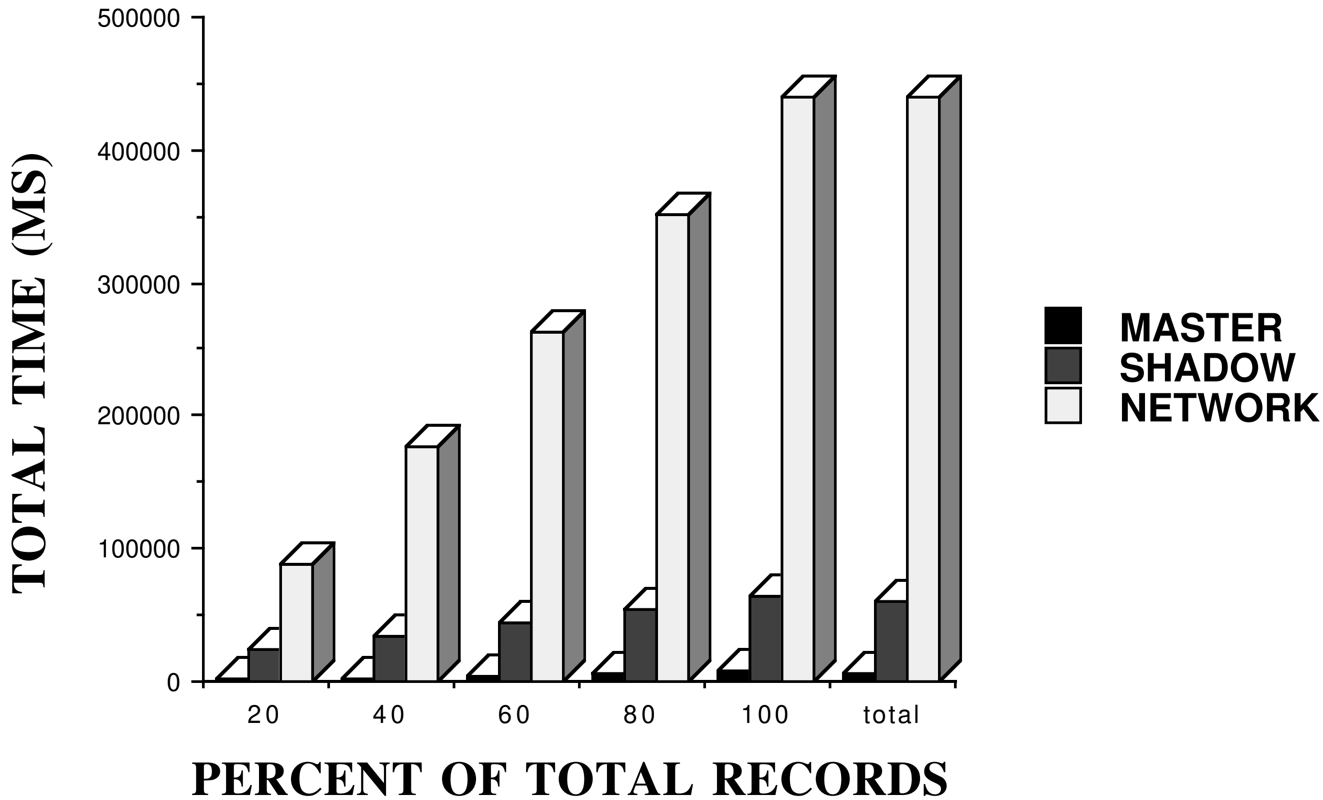
In most of the test cases, network transmission time represented about 83% of the total refresh time. We initially set the network speed to 56 kilobits per second (Kbps), and observed the network time to be the dominating factor. (see Figs. 5 and 6).



	Differential Refresh					Total Refresh
File Size Unit	20%	40%	60%	80%	100%	
Master	1414.722	2828.833	4242.834	5656.945	7070.947	6445.947
Shadow	8319.471	11608.583	14897.584	18186.695	21475.697	20225.697
Network	29396	58689.714	87901.143	117194.85	146406.286	146406.286

(time in milliseconds)

Figure 5: Asynch Pull for 500 records at 56 Kbps

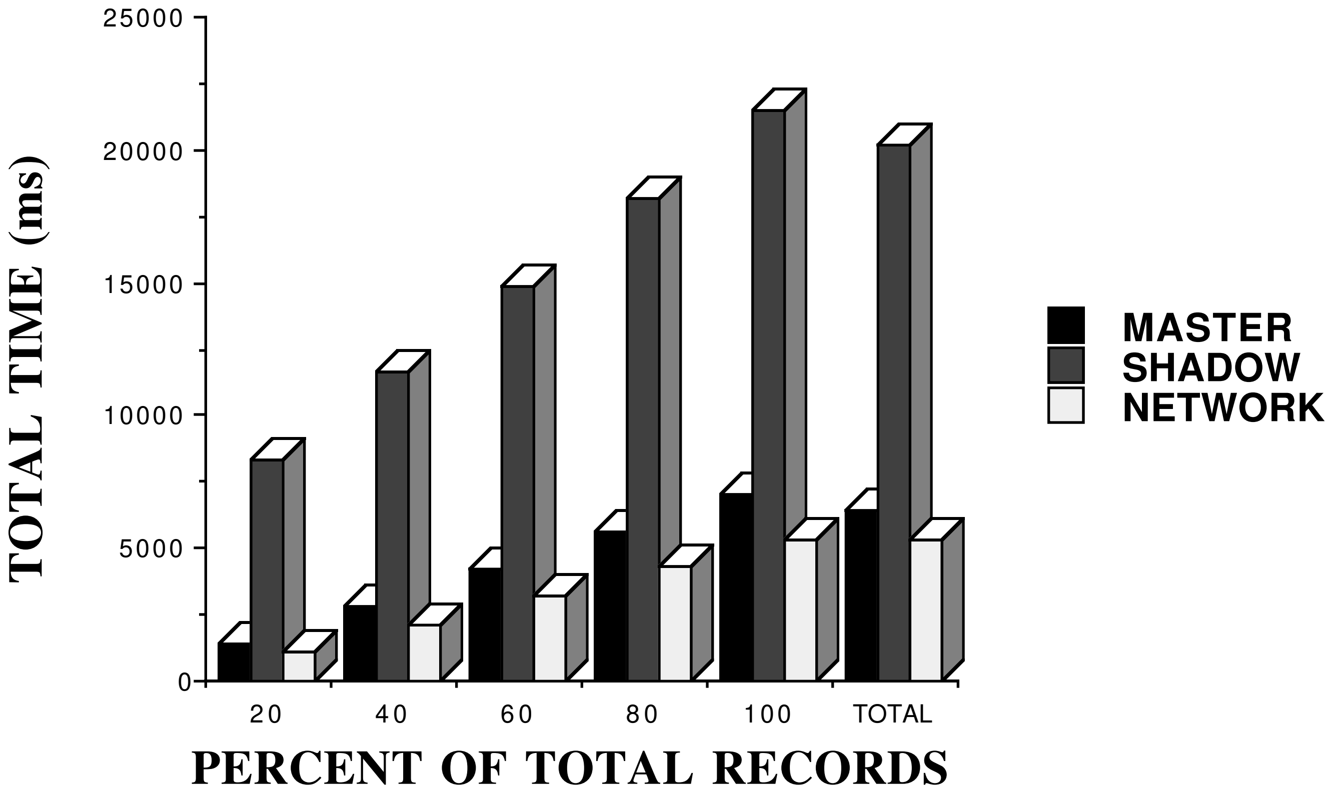


	Differential Refresh					
File Size	20%	40%	60%	80%	100%	Total Refresh
Unit						
Master	1414.721	2828.832	4242.833	5656.945	7070.947	6445.947
Shadow	8319.471	11608.583	14897.584	18186.695	21475.697	20225.697
Network	29396.00	58689.714	87901.143	117194.857	146406.286	146406.286

(time in milliseconds)

Figure 6: Asynch Push for 500 records & 3 shadows at 56 Kbps

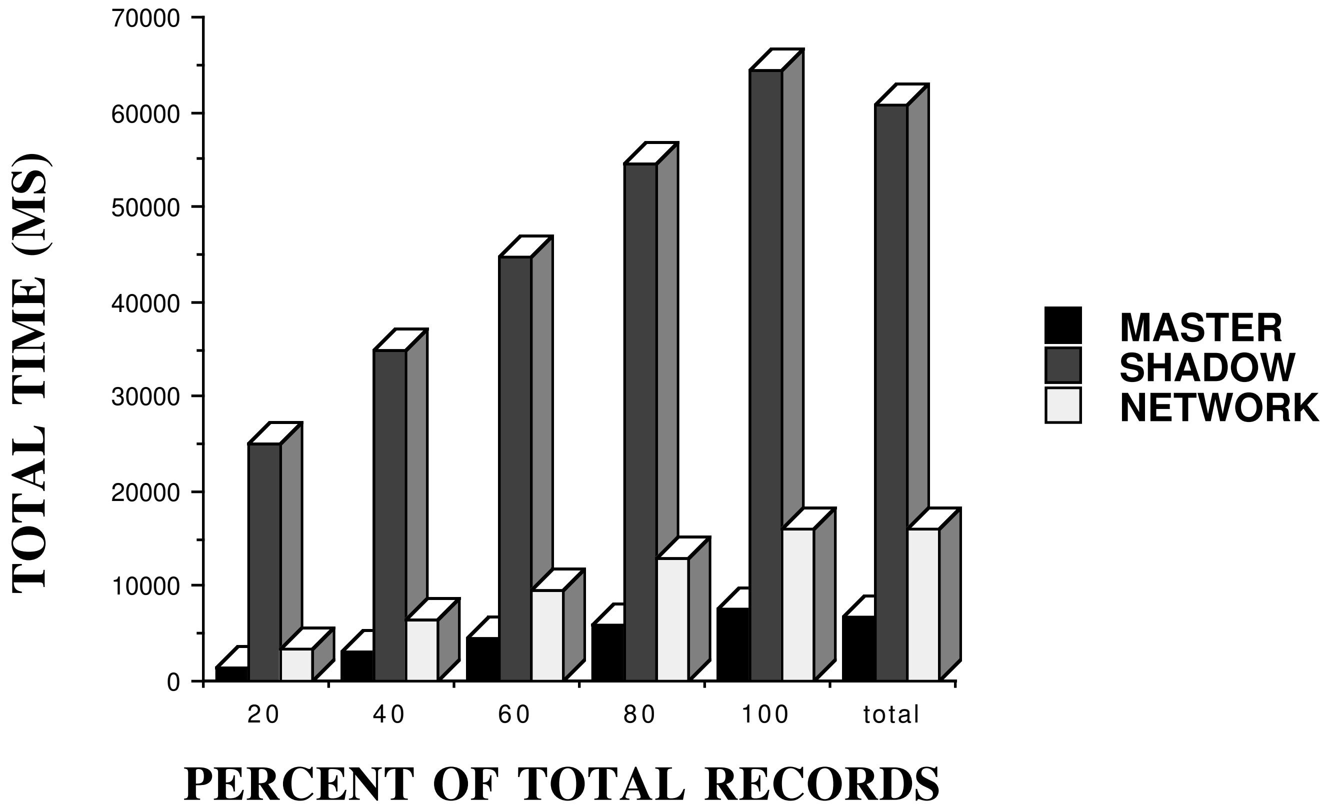
When the network speed was increased to 154 Kbps, the shadow time dominated. The best case was at the total refresh when the shadow time was 63% of the total. while the worst case, 76%, was at the 20% differential. By using a faster network we have shown that the shadow time becomes the dominating force. The master also plays a significant role as it has higher times than the network.
So our original hypothesis of the network as the bottle-neck is incorrect. The shadow is the bottle-neck with a lot of time spent on making the new tree structure in memory. (see Figs. 7 and 8).



	Differential Refresh					
File Size Unit	**20%**	**40%**	**60%**	**80%**	**100%**	**Total Refresh**
Master	1414.722	2828.833	4242.834	5656.945	7070.947	6445.947
Shadow	8319.472	11608.583	14897.584	18186.695	21475.697	20225.697
Network	1085.451	2147.917	3207.399	4269.865	5329.347	5329.347

(time in milliseconds)

Figure 7: Asynch Pull for 500 records at 1.544 Mbps



	Differential Refresh					
File Size Unit	20%	40%	60%	80%	100%	Total Refresh
Master	1493.664	2985.998	4478.002	5970.336	7462.340	6837.340
Shadow	24958.414	34825.748	44692.752	54560.085	64427.090	60677.090
Network	3256.352	6443.751	9622.197	12809.596	15988.042	15988.042

(time in milliseconds)

Figure 8: Asynch Push for 500 records & 3 shadows at 1.544 Mbps

Hypothesis two: Differential vs Total

From our test cases (Figs. 5 thru 8), it is apparent that the Differential provides savings in data transmission time, since usually less than 100% of the file is transmitted over the network. The processing time spent at the master and shadow also decrease due to the differential refresh. In addition, when the Differential is a significant percentage of the total file (approaching 100%), the CPU processing time will exceed that of the Total refresh algorithm. It will be important to find the exact threshold limit of the differential algorithm so that a total refresh can be performed in cases where the size of the differential is greater than the limit.

Hypothesis three: Push vs Pull.

The Push refresh algorithm provides a higher degree of parallelism despite the fact that our tests indicated that a Push and Pull consist of almost identical resource usage for the same number of shadows.

In the case of a Pull, we are speaking of n shadows requesting a refresh from the same master. Since locking occurs with each transaction to the master, the other n - 1 shadows will initially wait for the locks to be released before their refreshes can be processed. Thus, although each algorithm has the same resource usage, the elapsed time to complete the total refresh time will be quite different.

As will be shown by the analysis for hypothesis four, the data in a Push can be transferred to each shadow almost in parallel, while in the Pull algorithm, each shadow will wait until the data is transferred over the network and the locks are released before the next refresh can be processed. This fact shows the advantage in real elapsed time of using the Push algorithm over Pulls from all n shadows.

Hypothesis four: Push elapsed time

Three metrics for measuring a Push exist: Per Shadow, Sequential elapsed time over all Shadows, and Real elapsed time. Real elapsed time was discussed earlier as the time to complete the Push process. The following chart illustrates the three metrics for a Total refresh for 3 shadows at 1.544 Mbps (see Fig. 8. Time figures are in milliseconds).

Sequential

Steps executed	Elapsed Time Per Shadow	Elapsed Time Over 3 Shadows	Real Elapsed Time
Master prepares data	6250.250	6250.250	6250.250
Master send data to CP	195.337	586.011	195.337
Data transmitted over network	5316.363	15949.088	5316.363
Shadow receives data	15195.587	45586.761	15195.587
Shadow stores data on disk	5030.000	15090.000	5030.000
Shadow sends Ack. msg. to CP	0.110	0.330	0.110
Message transmitted over network	12.984	38.953	12.984
Master reads Ack. message	0.360	1.080	391.034
Total Time	32000.991	83502.472	32391.665

The first metric (per shadow), is the breakdown of time for each individual shadow in a Push. The second metric (sequential elapsed time) contains the total times for all shadows in a Push refresh. In our model, we used three shadows. Notice that the numbers in this metric are essentially three times the values given on a per shadow basis. This is consistent with what we would expect given that algorithm for an Asynch Push refreshes all shadows immediately underneath of the master.

The third metric gives the real elapsed time. This is the calculated best case time. Using our formula: RET = ST + NRS * Max(MT, CPT)

RET = 32000.991 + 2 * max(0.360, 195.337)
RET = 32000.991 + 2 * 195.337
RET = 32391.665 (which agrees with the total real elapsed time in the chart).

As the data indicates, the time spent at the master, shadow, and on the network are only slightly larger than the values for elapsed time for an individual shadow. For the real elapsed time case, we take into account the large degree of parallelism that naturally takes place which reduces the actual real time. As our description indicated when we earlier introduced the steps in the Push algorithm, after the master transmits the initial dataset to the first shadow, everything is done in parallel for the remaining shadows until the point where the master has to acknowledge each individual message from all shadows. The values in the real elapsed time metric reflect this knowledge of parallelism in the model.

Thus we can conclude that the real elapsed time for a Push from a master to n shadows is only slightly larger than the time taken to transmit to one shadow. We know that this value in no way approaches the magnitude of the total, unparalleled time that is seen in the data for sequential elapsed time.

5. Conclusions

The performance statistics of refresh strategies over various types of networks indicate that the network time plays a crucial role in determining the total refresh time (it is important to have good through-put on the network). The performance of the shadow is also a critical factor in the refresh process. It must do more work than the master, and it takes a very long time to set up the data in memory. Further research is needed so that that shadow time can be reduced.

Even though the Push algorithm has a large amount of parallel processing, the total system time used by the Push and Pull algorithms is essentially the same over a number of shadows.

Although this paper focused the attention on the time aspect of updating shadows in the directory system, the use of a particular strategy, or combination thereof, depends on many other factors such as : user needs, economic cost, etc. User needs may dictate that one particular shadow be allowed to Pull, while other shadows are Pushed to, since current information is of prime importance to the users at this shadow. Economically, since dollar cost is based on system resource usage, the Pull strategy may be better in certain circumstances.

Finally, the X.500 refresh strategies are not limited to applications in the directory services area. For example, one function of an Executive Information System (EIS) is to provide information, in detail and/or summarized form, to the management of an organization. If an organization has separate systems for each of the various functional areas (i.e. accounting, budget, personnel, etc.), a common approach would be to extract information from all data sources, place this data in a central database, and then make the information available to the requesting users. Distribution, or refresh strategies will naturally have to be resolved for such an organization.

References

[Bauer 89] M. Bauer. Naming and Name Management Systems: A survey of the state of the art. Technical Report 241, The University of Western Ontario, May 1989. Department of Computer Science Distributed Directories Lab.

[BBS 89] M. Bauer, J.M. Bennett, J. Slonim. A conceptual Framework for Distributed Directories. Technical Report 240, The University of Western Ontario, June 1989. Department of Computer Science Distributed Directories Lab.

[Bauer 90] M.E. Bauer, J.M. Bennett, S.T. Feeney, J. Blustein, R. McBrook. Replication Strategies For X.500: Experiments with a Prototype X.500 Directory. Technical Report 279. Oct 1990, 36pp. Department of Computer Science, Distributed Directory Lab. University of Western Ontario.

[Bennetta 89] J.M. Bennett. Distributed processing: A survey of the state of the art. Technical Report 242, The University of Western Ontario, May 1989. Department of Computer Science Distributed Directories Lab.

[Bennettb 89] J.M. Bennett. A survey of Research in Distributed Operating Systems. Technical Report 242, The University of Western Ontario, June 1989. Department of Computer Science Distributed Directories Lab.

[BSST 90] Bachmann, D.W., Segal, M.E., Srinivasan, M.M., and Teorey, T.J. "NetMod: A Design Tool for Large-Scale Heterogeneous Campus Networks," to appear in *IEEE J. Selected Areas in Communications (JSAC), 1990.*

[Kille 89] S.E. Kille. "The Design of QUIPU," UCL Research Note RN/89/14, February 1989.

[KRRT 90] S.E. Kille, C.J. Robbins, M. Roe, A. Turland. The ISO Development Environment: User's Manual, Volume 5: QUIPU. Department of Computer Science, University College of London, January 1990.

[LM 89] T. Lauriston, J.K. Mullin. Distributed Database Management Systems. Technical Report 243, The University of Western Ontario, June 1989. Department of Computer Science Distributed Directories Lab.

[LHMHPW 86] B. Lindsay, L. Hass, C. Mohan, H. Pirahesh, P. Wilms. A Snapshot Differential Refresh Algorithm, *Proceedings of ACM, Sigmod 1986: International Conference on Management of Data,* Washington, D.C. (May 1986) pg. 53-60.

[Rosea 89] M.T. Rose. NYSERNet White Pages Pilot Project: User's Handbook.1989. ISODE 6.0 documentations set.

[Roseb 89] M.T. Rose. NYSERNet White Pages Pilot Project: Administrator's Guide. 1989. ISODE 6.0 documentations set.

[Teor 90] Teorey, T.J. *Database Modeling and Design: The Entity-Relationship Approach,* Morgan Kaufmann, 1990, Chapters 8-9.

[ISO 89a] Recommendation X.500 - The Directory - Overview of Concepts, Models, and Services.

[ISO 89b] Recommendation X.521 - The Directory - Selected Object Classes.

[ISO 90] ISO/IEC Working Documentation on Replication and Knowledge Distribution. June 1990.

Appendix A - Formulae

This appendix includes a list of the formulas used to calculate the time to complete each step of the refresh process and the overall refresh time. All variables with respect to size are in bytes and all variables with respect to speed are in bytes per millisecond, unless otherwise noted.

Basic Steps

Step 1: Message to Master from shadow[3]

$$\text{Number of Packets(Message)} = \text{Ceiling}\left[\frac{\text{Message size}}{\text{Packet size of Shadow}}\right]$$

$$\text{CP Transfer Time (Both DSAs)} = 2\left[\frac{\text{Number of packets} * \text{Packet size}}{\text{Channel Speed}}\right]$$

Packet Transmission Time = Max(Propagation Delay, Inverse of DSA Packet Rate)

Transmission Time = Number of Packets $*$ Packet Transmission Time

Total Time = Propagation Delay + CP Time + Transmission Time

Step 2: Master Processes Message[4]

$$\text{Total Time} = \frac{\text{Number of Fixed Instructions}}{\text{DSA instruction speed}^{5}}$$

---

3 In the case of a Pull it is the request message. In the case of a Push it is the acknowledgement message and is the second to last step in the Push algorithm. In both cases, the time is the same.

4 In the case of a Pull it is the request message. In the case of a Push it is the acknowledgement message.

5 DSA instruction speed is the number of instructions executed per millisecond

Step 3: Master Prepares Data

Total Number of Instructions = Fixed Instructions + (Number of records transferred * Instructions per record)

$$\text{Total Time} = \frac{\text{Total Number of Instructions}}{\text{DSA instruction speed}}$$

Step 4: Transmission Time to Shadow

$$\text{Number of Packets(Transfer)} = \text{Ceiling}\left[\frac{\text{Transfer File size}}{\text{Packet size}}\right]$$

$$\text{CP Transfer Time (Both DSAs)} = 2\left[\frac{\text{Number of packets} * \text{Packet size}}{\text{Channel Speed}}\right]$$

Packet Transmission Speed = Max(Propagation Delay, Inverse of DSA Packet Rate)

Transmission Time = Number of Packets * Packet Transmission Speed

Total Time = Propagation Delay + CP Time + Transmission Time

Step 5: Shadow Places Data Directly into Memory

Total Number of Instructions = Fixed Instructions + (Number of records transferred * Instructions per record)

$$\text{Total Time} = \frac{\text{Total Number of Instructions}}{\text{DSA instruction speed}}$$

Step 6: Shadow Writes Data to Disk

Number of RBAs = 1

Number of SBAs[6] = $\text{Ceiling}\left[\dfrac{\text{Number of Records}}{\text{Floor}[\text{Block size / Record size}]}\right] - 1$

Total RBA Time = Number of RBAs $*$ Access Time per RBA

Total SBA Time = Number of SBAs $*$ Access Time per SBA

Total Access Time = Total RBA Time + Total SBA Time

Overall Times Pull/Push case[7]

Shadow Time $= \text{Step 5} + \text{Step 6} + \dfrac{(\text{Step 1 CP Time})}{2} + \dfrac{(\text{Step 4 CP Time})}{2}$

Master Time $= \text{Step 2} + \text{Step 3} + \dfrac{(\text{Step 1 CP Time})}{2} + \dfrac{(\text{Step 4 CP Time})}{2}$

Network Time = (Step 1 Total Time - Step 1 CP Time) +
(Step 4 Total Time - Step 4 CP Time)

6 Assumes Block size >= Record size
7 Push case times are for one shadow.

Overall Time for n Shadows in Push case[8]

Shadows Time = Shadow Time[9] $*$ n

$$\text{Master Time} = \text{Step 3} + n * \frac{(\text{Step 4 CP Time})}{2} + (n * \text{Step 2})$$

Network Time = Network Time[10] $*$ n

8 Number of shadows > 1.
9 Time is for one shadow.
10 Transmission time to one shadow.