Towards an Archival Intermemory

Andrew V. Goldberg and Peter N. Yianilos

NEC Research Institute, Inc.
4 Independence Way
Princeton, NJ 08540

Abstract

We propose a self-organizing archival Intermemory. That is, a noncommercial subscriber-provided distributed information storage service built on the existing Internet. Given an assumption of continued growth in the memory's total size, a subscriber's participation for only a finite time can nevertheless ensure archival preservation of the subscriber's data. Information disperses through the network over time and memories become more difficult to erase as they age. The probability of losing an old memory given random node failures is vanishingly small – and an adversary would have to corrupt hundreds of thousands of nodes to destroy a very old memory.

This paper presents a framework for the design of an Intermemory, and considers certain aspects of the design in greater detail. In particular, the aspects of addressing, space efficiency, and redundant coding are discussed.

Keywords: Archival Storage, Distributed Redundant Databases, Electronic Publishing, Distributed Algorithms, Error Correcting Codes, Erasure-Resilient Codes, Information Dispersal Algorithm, Digital Library, Internet.

1 Introduction

Through publication we preserve and transmit our knowledge and culture. Electronic media promises to improve transmission but the important issue of preservation has yet to be addressed in the context of the emerging worldwide network of computers.

Print publications are preserved by our collective agreement to fund and support libraries at educational institutions and several levels of government. This solution amounts to a distributed redundant storage scheme and is further strengthened by its self-organizing nature. That is, there is no essential central world library. It is possible that the role of libraries will simply expand to include the collection of purely electronic items but in this paper we explore another possibility: that a largely self organizing distributed *world memory* might emerge.

Our first thought was that commercial services might arise to offer archival data storage. In this scenario the publisher of an academic journal might, for example, pay an up-front fee to ensure that a journal issue will be available archivally in electronic form. The data would be widely distributed, and financial reserves established to care for it in the future. Sloan suggests a simpler alternative in [18]; that such a service might be provided by a single major university or other institution. These approaches are, however, inconsistent with the essential spirit of the Internet, i.e. growth and effectiveness in the absence of strong central control or commercial organization.

Given the current state of world connectivity, and the clear prospects for continued growth, widely distributing archival data is not a significant technical problem. Ensuring that it survives is the real issue since unlike books which persist for centuries or more, machines have lifetimes measured in small numbers of years.

In the solution we envision, a user *donates* storage space S to the Memory for a period of time, and in return receives the right to store a much smaller amount of data αS in it. A three-year donation of one gigabyte might, for example, correspond to the right to archivally store 200 megabytes. Here the system's *efficiency* α is 0.2.

Our approach rests on an assumption of continued system growth. Falling costs per bit of data storage and growth in the number of Internet/Web users and their appetite for data storage are supporting factors. It is because of this assumption that a bounded-time donation of storage space can correspond to an unbounded-time storage space grant. The system's efficiency factor

must be low enough to support this trade and to allow sufficiently wide distribution of each memory item.

This paper begins the process of formalizing the notion of *Intermemory*, presents specific proposals for certain aspects of its design, and comments on the manifold other challenges that must be addressed before the first stable and secure Intermemory goes online.

We remark that large organizations might implement one or more private Intermemories resulting an *Intramemory* that is maintained without explicit backup over the organization's wide area network. The public Intermemory that motivated this paper is in this sense the *world's memory*.

The contributions of this paper are:

1. General development of the notion of Intermemory and its architecture.
2. The idea of trading a finite-term donation for unbounded storage rights.
3. Specific designs for distributed redundant data storage.
4. A specific approach to the addressing problem.

The growing interest in digital libraries was recently surveyed [17] and a general discussion of the problem of preserving digital documents is contained in [16]. We observe that libraries serve two distinct roles: maintenance of a historical record, and selection of appropriate materials. Web-indexing approaches dispense with the second role and allow a user to view all the world has to offer. They fail, however, to deal with the first. Creating a record of everything on the Internet [9] represents, by contrast, an emphasis on the first role and eliminates entirely the second. Our concept of Intermemory combines the archival function with what amounts to a self-selecting publication process. It is worthwhile noting that Intermemory solves the preservation problem associated with the ephemeral nature of computer storage media. As new subscribers replace old ones, memories are automatically copied onto the latest medium.

Anderson's work [3] parallels our own, but his emphasis is on the freedom-of-expression and individual rights rationale, and less on the specific technical and architectural issues involved.

The problem of efficiently distributing a file within a distributed system having connectivity described by a given undirected graph is considered in [11]. That is, a node can reconstruct data from its neighbors. This idea is related to that of *diversity coding* (see [15] for recent work) in which there are multiple encoders and each of several decoders has access to some fixed subset of the encoders. By contrast, we assume that the world network provides a complete graph. So while this work is clearly related to our problem it does not appear to be directly relevant.

The topic of [10] is a discussion, in general terms, of several design issues and tradeoffs relating to the implementation of redundant distributed databases. Similar issues are discussed in [1] and both papers cite Rabin's work [14], which introduced the idea of coding-based redundancy to the database community. He in turn recognizes the earlier related work [5]. Odlyzko is also interested in perpetual storage [12], where his focus is on preservation of and access to research literature.

The *redundant array of inexpensive disks* (RAID) approach [13] distributes a file system over an array of disks directly attached to a host computer. Redundancy is provided by variations of a simple parity approach. Further distributing not only the storage, but also the control of a file system, is the subject of numerous papers including [4], which describes the xFS system now under development. This system distributes a file system over cooperating workstations while retaining high performance. By contrast: i) our proposed Intermemory includes redundancy and dispersal on a much larger scale, and is capable of tolerating the loss of a large portion of the participating nodes, ii) the emphasis is on the archival aspect rather than on performance, iii) participating processors are assumed to be widely dispersed on the world internet, their participation of an ephemeral nature and subject to very little central control, iv) adversarial attacks are a primary Intermemory design concern, v) our focus is on implementing a block-level substrate upon which more than one file system format might be implemented over time, and vi) because of our emphasis on archiving, a write-once model is our starting point with research continuing towards a read-write system.

2 Components of Intermemory

For the purposes of this paper the Intermemory is an immense, distributed, self-organizing, persistent RAM containing 2^m memory blocks. It is addressed by an m-bit binary address, and each block consists of N words of w bits. Subscribers are allocated fixed addresses into which only they may write. We assume a write-once model: once written, the data cannot be modified or deleted. Anyone may read from any location.

To achieve wide-spread use the Intermemory will, no doubt, have to deal with many high-level issues such as intelligent addressing, search and perhaps other functions built on top of the simple substrate we consider — as well as interpretation of specific data types (such as

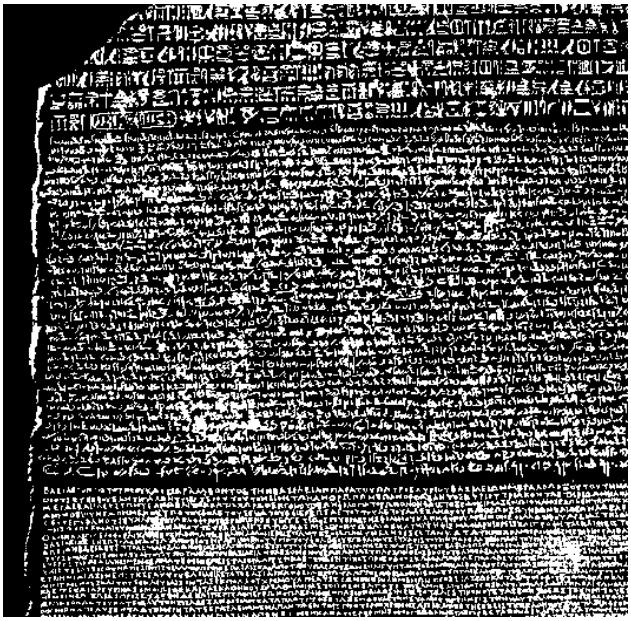

Figure 1. The Rosetta stone was discovered by a French engineering officer in August of 1799 while strengthening a fort located on the west bank of the Rosetta arm of the Nile [7]. It records a bilingual decree promulgated by the whole of the priesthood of Egypt in the ninth year of the reign of Ptolemy V (B.C. 196). It concludes with a resolution ordering that a copy of the inscription in hieroglyphs, Greek, and demotic shall be placed in every large temple in Egypt. While papyrus was in use at the time, this message was recorded is a basalt slab. We observe that the priesthood's message was dispersed to several locations, recorded in a durable medium, and that its bilingual nature ensured semantic longevity – an approach not so different in principle than the Intermemory we now contemplate. Given that Egypt's earliest record was already regarded as ancient by the ancient Greeks, we might imagine that archival preservation was very much on the mind of Ptolemy V and the priesthood. The modern world is fortunate that their attempt at preservation succeeded as it led to the first deciphering of the hieroglyphic record left thousands of years earlier.

text, image, and even programs) in an archival manner. These topics are beyond the scope of this paper. We also set aside the important cryptographic issues that must ultimately be addressed for the system to honor its archival promise — and at a low level, enforce write permissions and provide data authentication. Yet other dimensions of the problem are identified in sections 3 and 4.

We contend, however, that while deployment of a complete system in the senses identified above is a daunting task, less complete solutions are practical today. To support this contention we have chosen to present our Intermemory design in rather concrete terms in which specific values are proposed for the system parameters such as m, N, etc. In order to illuminate the architectural issues underlying each of these design decisions we will comment on the tradeoffs that apply.

A world-wide fully connected network is assumed such that the Intermemory service on each participating processor can be contacted at some network address (NA) – today an IP address and port number suffice.

2.1 Redundancy and Dispersal

In this section we describe a particular scheme based on well-known *erasure codes* and the idea of using them for information dispersal. For simplicity we will assume that the N words within each block are elements of some finite field F of approximately 2^w elements. The natural choice of Z_p with prime $p \in [2^{w-1}, 2^w - 1]$ allows for simple computation at the expense of a very small increase in storage space when translating the original block to field-element representation. These words may then be regarded as the coefficients of a polynomial of degree $N-1$ over F. Then the value assumed by this polynomial at any N distinct points suffice to uniquely identify it. This classical observation is a key idea in coding theory [6] and corresponds to the Vandermonde matrix case of Rabin's information dispersal framework [14] in the extreme setting where each block contains a single word. The idea is that one may evaluate the polynomial at $r \cdot N$ points, expanding the block by a factor of r, such that any set of N values suffices to reconstruct the original block – with these separate values dispersed among subscribing processors. Note that we must have $r \cdot N < |F|$. The approach is space-optimal since every subset of processors sufficient for reconstruction, i.e. of size N, contains exactly the total space of the original data. Encoding and decoding correspond to the *polynomial evaluation and interpolation* problems respectively. Using $O(N \log N)$ FFT-based polynomial multiplication

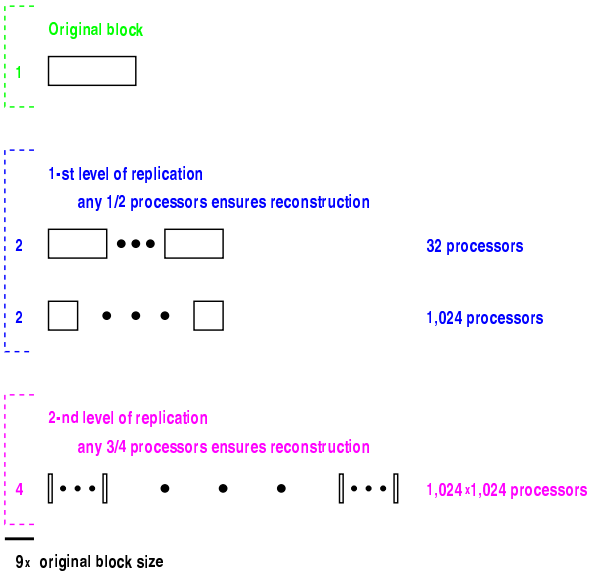

Figure 2. Data replication and dispersal. At the first level of replication, each dispersal level takes twice the original block size. The second level of replication takes four times the original block size. The total memory requirement is nine times the original block size. When the dispersal is completed, over 2^{18} processors need to be disabled for the data to be lost.

one may perform both evaluation [8] and interpolation in $O(N \log^2 N)$ time. Since these algorithms are also efficient in practice, encoding and decoding complexity does not limit our choice of N. We remark that more complex coding schemes [2] might be considered that are asymptotically faster yet but require slightly more than N words to reconstruct the original. Also, the topic of reconstruction given some number of erroneous values has been considered in the literature.

The $r \cdot N$ word output block may be arbitrarily divided into subblocks and dispersed among subscribing processors. The maximum dispersal possible using the scheme above sends a single word to each of $r \cdot N$ processors. An adversary would then have to destroy N processors ($32,768$ in our example below) to erase a maximally dispersed memory block. The minimum dispersal consists of storing the entire output block at one processor. The primary consideration in establishing N is the tradeoff between the degree of information dispersal possible, and the inconvenience of large blocks (by today's standards).

Given r and N it follows that $w \approx \log_2(r \cdot N)$ minimizes the input data block size which is then $N \log_2(r \cdot N)$ bits. For example, $N = 2^{15}$ and $r \approx 2$ gives $w = 16$ and a block size of 64 kilobytes. If Z_p is used then the requirement $r \cdot N \leq p$ forces r to be slightly less than 2.

Reading a maximally dispersed block is a very expensive operation in any model in which network connections have significant cost. For this reason we propose that each data block be stored at several levels of dispersal. In the example above, the encoded block might be stored:

- In its entirety at a single processor
- Dispersed among 32 processors
- Dispersed among 1,024 processors
- Dispersed (maximally) among 65,536 processors

In most cases we expect the block to be readable by contacting a single processor. If this fails we must contact at least 16 of the 32 at the next level, and so on. The interesting property of the encoding transformation is that this dispersal takes place with no additional computation. That is, blocks are merely subdivided and dispersed. To read a block, it is merely necessary that a sufficient number of words from the encoded block can be accessed. These might come from different levels. These four levels consume $7\times$ the storage of the original block (not 8 because the first level need not include any redundancy at all). The idea, then, is that space is traded for reduced expected read time.

As fragments of a block are received at a processor, they are stored in its memory along with the corresponding address and index within the block. Separate storage areas exist for the different fragment lengths corresponding to the four levels of dispersal above.

Because block addresses must be stored along with each fragment[1], maximal dispersal introduces large space penalties, on the assumption that $m \gg w$. For this reason we suggest that the bottommost level be omitted. The new bottommost level disperses to 1024 processors.

A second phase of dispersal is then possible in which the contents of the bottom level incoming data buffer is treated as a data block and further dispersed. This dispersal does not take place until the this buffer has accumulated a full block. We suggest a single widespread $r = 2$ dispersal to, say, another $1,024$ processors. In this way the processor's bottom-level buffer is, in effect, backed up in the network. The result is that an adversary must attack $\approx 2^{18}$ processors (1/4 of the 2^{20} involved) to erase a memory block that has been

[1] The indices also required represent minimal overhead.

dispersed to this degree (assuming there enough processors in the network to allow such dispersal). Notice that replacing the bottom level with a second phase of dispersal increases the overall space cost to $9\times$ because of the additional redundancy introduced by the second phase. Multilevel dispersal schemes such as this represent an effective way to limit the overhead of address storage while achieving very wide dispersal and moderate block size.

In the system we propose a failed processor is detected by its neighbors, its data reconstructed, and its storage responsibilities assumed by another processor in the network. The manner in which these failures are detected and the reconstruction accomplished is discussed later.

Dispersal on the scale proposed above is not necessary to achieve archival performance under a model in which network or processor failures are assumed to be independent. But an assumption of independence is far from valid for many reasons. These include systematic failures introduced by software bugs, viruses, and overt adversarial action. We suggest that a healthy Intermemory might result from many independent implementations of the distributed algorithm starting from a common specification that is simple enough to be formally verified.

The probability of losing even one memory block is then dominated by the probability that one will, despite the repair activities mentioned above, permanently lose access to $\approx 2^{18}$ processors, spread randomly (by virtue of the addressing approach described later) throughout the network. We will not present calculations here because the issue of an appropriate failure model is beyond the scope of this paper. But it clear that the degree of maximum dispersal is the key variable.

The discussion above illustrates how the variables N, r, w, m, and the choice of levels and phases of distribution are related in the design of an efficient Intermemory, and we next turn to the system's behavior over time.

2.2 Trading a Moment for Eternity

In our model of Intermemory, each subscriber "invests" a certain amount of memory for a certain time period and earns "interest" at each time unit. Then the original investment is withdrawn but the interest remains invested and represents the user's perpetual ownership of space within the system. Subscribers enter the system and later leave it but we assume that the net effect is that total investment increases over time. The source of the "interest" is the system's assumed

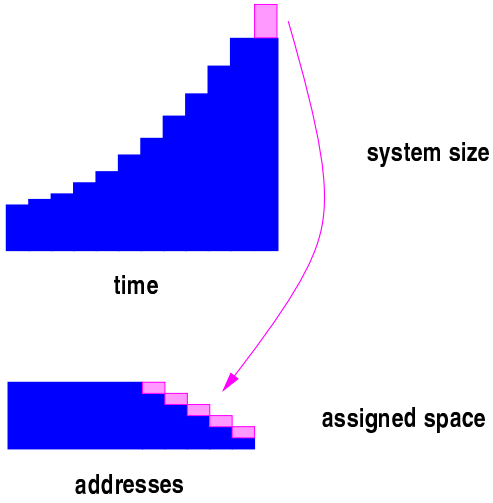

Figure 3. New memory entering the system is assigned to Intermemory locations that have not been stored or completely dispersed. A freshly written block becomes more widely dispersed with time, until it reaches the maximum dispersal level.

continued expansion. That is, each unit of growth in the system's total capacity is paid-out as "interest" to current and in some cases to past subscribers.

More precisely, we measure participation in the system in terms of an arbitrary unit of memory. The number of units participating at time t is denoted P_t. Time is discretized and we model the system's growth over time as:

$$P_t = P_{t-1} - \text{deaths} + \text{subscriptions}$$

and model the last two terms in terms of *rates*, that is:

$$P_t = P_{t-1} - P_{t-1}r_d + P_{t-1}r_s = (1 - r_d + r_s)P_{t-1}$$

where the parenthesized term forms a single growth constant g and we write:

$$P_t = g \cdot P_{t-1}$$

The rate r_d is the reciprocal of each processor's expected lifetime, and r_s combines the effects of new users joining the system, and technology improvements making it possible for new entrants to contribute an increasing amount of space. We assume $g > 1$. Then the growth in space from time $t-1$ to time t is

$$(g-1)P_{t-1}$$

so the new space per unit (at time t) is:

$$\frac{(g-1)P_{t-1}}{P_t} = \frac{g-1}{g}$$

We allow each unit to consume this space storing its memories into the Intermemory. So over each unit's expected lifetime it may spend:

$$\frac{1}{r_d} \cdot \frac{g-1}{g}$$

This expression makes clear that longer lived processors directly compensate for a lower growth rate. Today storage costs are declining rapidly and computers are frequently discarded or redeployed after only a few years use. If one expects costs to fall slower in the future then constant efficiency can be maintained to the extent that processors stay on the job for a longer time. But it is certainly possible that the system's efficiency would decline over time making it less attractive to new subscribers.

Example: Suppose t is measured in days and we expect processors to participate for 1000 days so $r_d = .001$. Next suppose $r_s = .0015$. Notice that $(1 - r_d + r_s)^{365} \approx 1.20$ so that we are assuming an annual system growth rate of roughly 20%. Now $g = 1.0005$ so $(g-1)/g \approx .0005$ and over its 1000 day lifetime a processor will accrue the right to consume roughly 0.5 units of perpetual storage space. So in this example, selected to be plausible, each participating processor can consume half of the memory it donates. An actual system will have to confront the possibility that the combination of growth rate and death rate falls short of expectations, and somehow regulate the consumption of new memory.

In the development above, new space entering the system is allocated to the system's current subscribers, i.e. at time t. So a subscriber's rights vest only during its life. We now explore the effect of a different strategy where the allocation is made to subscribers at time $t-\delta$. Over a unit's expected lifetime it may then spend:

$$\frac{(g-1)P_{t-1}}{r_d P_{t-\delta}} = \frac{1}{r_d}\frac{g-1}{g^{1-\delta}}$$

As $\delta \to \infty$ it seems that a processor may enjoy an unbounded right to consume space – and indeed this is true. But such a system is not practical because it does not represent a fair trade. The subscriber, clearly, expects to vest some rights within a reasonable period of time. It nevertheless serves to illustrate that delayed vesting can boost system efficiency.

In a mixed allocation strategy where a proportion α_i of the new space is dispersed with delay δ_i a unit may spend:

$$\frac{1}{r_d}\sum_i \alpha_i \frac{g-1}{g^{1-\delta_i}}$$

where $\sum_i \alpha_i = 1$ and each $\alpha_i \geq 0$. We remark that uniform allocation over the previous N time units results in a particularly simple expression $(g^N - 1)/(Nr_d)$ for a subscriber's ultimate vested space rights.

Example: Under the assumptions of the previous example, suppose 1/3 of the new space is allocated during a processor lifetime so that the first two levels of dispersal can occur. Then 2/9 vest during an equally long period following its death, and the final 4/9 in a period of the same length that follows. Here $\alpha_1 = 1/3, \alpha_2 = 2/9, \alpha_3 = 4/9$, and $\delta_1 = 0, \delta_2 = 1000, \delta_3 = 2000$. The result is that the processor will eventually accrue the right to consume 1.0 units of space – just as much as it invested.

Thus dispersal takes place over time and old memories are dispersed more widely than new ones.

The discussion above is rather simplistic in that it assumes that the system's size grows consistently, and that this growth is governed by a simple exponential rule. A real system will have to confront problems such as:

1. Nonuniform growth: the system may grow in spurts. In an extreme case a single large user might subscribe and donate an immense amount of storage for a finite time. Such a user cannot expect to vest perpetual rights in proportion to the donation – at least not in a short time frame. Space reserves might be maintained for such reasons.

2. Growth may slowdown: it is not clear for how much longer today's climate of rapid technological advance will persist. If this growth rate declines, then constant efficiency is maintained only by increasing the donation period. But this may occur naturally since a more stable technological environment may lead to longer processor lifetimes.

3. Shrinkage: the system may, from time to time, shrink. If this condition is temporary it might be dealt with by drawing on reserves, or by cannibalizing deeply dispersed memories. That is, by commandeering the space occupied by deeply dispersed memories and using it to store newer ones. If growth resumes this damage will be repaired in the ordinary course of system operation as sketched in the next section. If shrinkage persists, then society, or the sponsoring organization, is left to decide whether preservation is worth the associated costs.

These problems make clear that some regulation of the vesting of rights is needed in a functioning Intermemory. This is itself an interesting question but is beyond the scope of our paper.

Finally, we observe that one might also assume that the memory capacity of an individual subscriber increases over time. In addition to the effect of declining storage cost, the trend towards more online data per user supports it. Under the addressing method described in the next section this assumption simplifies the handling of buffer overflow in a subscribing processor.

2.3 Addressing

In this section we present a particular scheme for Intermemory addressing. The central design objective is that the important functions of dispersal and self-repair may be performed such that each processor communicates with only a limited number of neighbors, and that the total volume of communication is minimized. Other solutions are possible and our current research is now exploring several variations.

The approach we present here combines hashing, pseudo-random generators, and a distributed name server (DNS). We will not comment on the DNS implementation except to discuss its reconstruction in the event of failure, or replacement. All hash functions and generators are assumed to be publicly known but we will not consider the choice of specific functions.

An m-bit Intermemory address A is first mapped to a q-bit *virtual processor number* V — where it is assumed that 2^q exceeds the number of processors in the network (now or in the future). We suggest that $m = 256$ and $q = 64$ represent reasonable values.

The DNS implements a many-to-one relation between virtual processor numbers, and *physical processor numbers*, denoted P. It also implements a one-to-one relation between physical processor numbers and *network addresses*, denoted N, that today would consist of a combined Internet address and port number. The path is then $A \rightarrow V \rightarrow P \rightarrow N$.

In this way we locate the actual processor containing the top-level dispersal of a memory block, i.e. the block in its entirety.

A $V \times P$ relation, once established, persists forever – at least for the purposes of this paper. But the $P \times N$ relation is fluid as processors, and even worldwide network structure, change. The DNS also accepts responsibility for creating relations as necessary, and thus serves as a load balancer by assigning V values used for the first time to lightly loaded physical processors. Other load balancing operations are possible

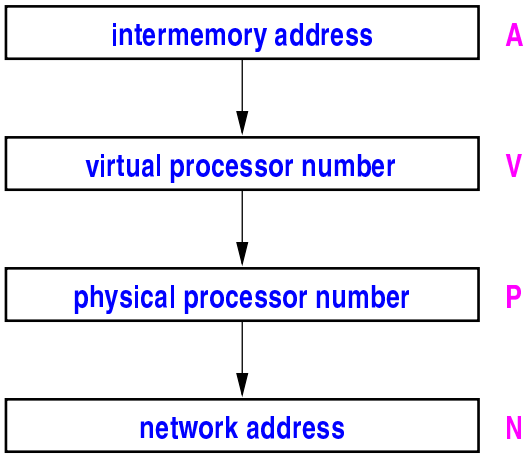

Figure 4. DNS mappings. Note a semantic distinction between Intermemory locations and virtual processors. A virtual processor holds Intermemory blocks (nondispersed), and or dispersed block fragments. In addition, the first DNS mapping performs load balancing. The second mapping assigns virtual processors to physical processors. The last mapping associates a network address of a real processor with a physical processor number.

but we will not discuss this issue further.

A graph relating the processors is defined by using P-values and a pseudo-random generator, or a family of hash functions. The virtual numbers of the neighbors of each processor are obtained by seeding the generator with its physical number P, and generating some number d of values – where d is no smaller than the maximum fan-out in the dispersal scheme. Actually, we require d distinct neighbors, so more than d pseudo-random values may be necessary. Notice that this graph is purely a function of the P and the information in the DNS's first relation.

Each processor retains its P value, along with the current N values corresponding to each neighbor. On DNS failure, we then have a high-degree random graph interconnecting all subscribers that may be used for reconstruction. Even if some portion of the network addresses are nonfunctional, we can with very high probability reach all processors.

We remark that the same network might be used to implement the DNS itself, and its operation might be distributed and performed by a single Intermemory code.

The primary purpose of the graph is information dispersal. Given P, A, and the level of dispersal as seed,

another pseudo-random generator (or collection of hash functions) is used to identify a subset of the processor's neighbors to which the dispersal takes place. Thus all processors in a block's distribution tree can be computed based on A and the DNS.

Another purpose of the graph is repair, and we will sketch this process. Periodic polling of neighbors is used to identify dead processors, and when the processor is replaced this graph is used to reconstruct its proper contents. An item the processor should contain that is not at the top dispersal level must have arrived from another processor. That other processor will eventually poll, notice the item it earlier dispersed is gone, and send it again. A top level item will have been dispersed to at least one level, so the neighbors can be examined for any level two blocks that must, based on the address calculation process above, have arrived from the processor being reconstructed. The corresponding Intermemory addresses are then read and retained. This polling cycle might span, say, a few months. So over that time any new replacement is rebuilt. In an analysis of a complete system the frequency of polling is a key design variable.

We briefly observe that the size of the first DNS relation is $|V|$ which is approximately the number of memory blocks in the system plus d times the number of subscribing processors. The size of the second table is just the number of subscribing processors.

We remark that our mapping of an Intermemory address to a virtual processor number is primarily for semantic reasons. One might, for example, set $m = q$ and use the identity hash so that $A = V$. The semantic distinction is necessary since A refers to the address of an Intermemory block as communicated by a user to the system, while P is another name for some physical processor. Confusion arises when V values are generated as part of graph construction – these are not Intermemory addresses. Also, a case for large m might be made based on some decomposition of the address space, while q can remain small, saving storage space and time in the DNS implementation.

Finally we observe that the DNS can be rebuilt entirely from the information held by each subscribing processor. This is performed as a distributed algorithm on the current graph connecting all subscribers. Thus the DNS need not itself be engineered as an archival system, and might even be replaced from time to time.

The problem of buffer overflow must be considered. That is, when a processor is sent more data than it has room to store. A solution idea is to design the DNS such that if a processor nears capacity the likelihood that the DNS assigns a new V to it tends to be small provided that other processors have more room available. Another problem arises when a V is reused, but given large enough q, this should be rare indeed.

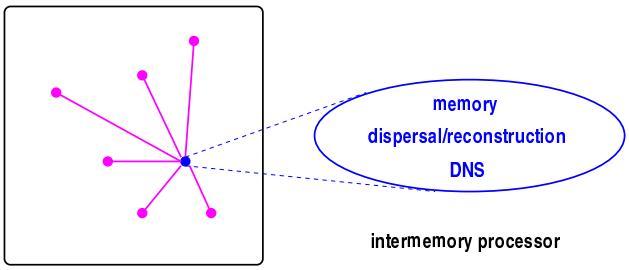

Figure 5. Each Intermemory processor has memory used to store data (including replicated data) and software for memory replication, dispersal, reconstruction, and DNS. Every processor is assigned d "neighbors."

If, somehow, the entire system fills up, a processor should discard small fragments (deep dispersal levels) first. At some future time, this processor will die and be replaced by a new larger one (under our assumption of growth in individual capacities). When this occurs, the repair process will fill in the discarded fragments.

3 Manifold additional issues

For simplicity, and to emphasize the archival component of our proposal, we have adopted a write-once model in this paper. A read/write model is certainly possible but introduces considerable additional complexity as with replicated databases. It becomes necessary to track versions of each memory block and fragment and the design of caches is complicated. We have come to view the read/write version as a distinct research project with somewhat different potential applications. Finally we remark that a one-time *erase* operation may be easier to implement but this matter has not been considered in detail.

In our implementation, data written to Intermemory becomes more difficult to erase with every level of replication. The system may send notices to data owners as the replication level increases. An owner may want to keep a copy of the data until the desired level of replication is reached.

The current paper does not address the distributed computation problems of determining when a processor is dead, implementing fault-tolerant DNS, and the protocol for adding new processors and users to the system. Efficient and robust solutions to these problems are crucial for reliable Intermemory.

Another important set of problems is related to per-

missions and security. In particular, it should be impossible for an adversary to erase or corrupt Intermemory without destroying a very large number of processors. We hope that public-key cryptography can provide acceptable solutions to these problems. Also, we have assumed a simple permission structure in which each subscriber has sole write permission to a set of addresses. We observe that a group permission structure may be needed for some applications. Also, because of the system's archival mission, it may be appropriate to revoke all write permissions after some period of time has passed.

For the Intermemory to function for a very long time and to outlast current software and technology, Intermemory protocols should include a provision for their replacement with new ones, without loss of data.

We have not discussed how new users or new processors join the system. One possible solution is to handle this through a collection of trusted sites in a way similar to that currently used for assigning IP addresses. A related issue is that of policing: When a new user joins the system and is given a block of Intermemory it can write to, the user commits certain resources as a "payment," and the system has to ensure that the user honors this commitment.

4 A Glimpse at the Layers Above

In this section we briefly discuss some of the higher level structures that might be built on top of the low-level infrastructure just described.

A distributed file system might be implemented in which Intermemory blocks correspond to disk blocks. This would allow a subscriber to build a directory hierarchy and specific files could then be accessed by incorporating Intermemory software into an application, or more conveniently by deploying daemons that provide an access service that makes Intermemory directories look like standard disk-based directories.

A gateway service could then connect the traditional Web with the Intermemory, i.e., a Web page could link to a document stored there, i.e. within a file system that is itself implemented in Intermemory. In the write-once model such links would have the virtue of never expiring.

A fascinating problem related to Intermemory is the definition of an archival data type system. For example, JPEG is currently a standard image format. However, there is no guarantee that in 100 years a JPEG decoder may be easily accessible. An important type is that of "program", that is, an archival representation for a computer program that allows it to be run via emulation at any future time. We envision a simple base machine model and I/O specification on top of which compilers and interpreters can be built. Applications such as data conversion might be written using such a system but we suggest that its utility extends to a much broader class of problems.

One issue to consider is whether a data item should be converted to a new data type as the old one becomes obsolete, or instead be translated whenever it is accessed, using an archival program. Each translation carries with it the finite risk of data loss, and over many years one would therefore expect significant losses to occur with either approach. But in the latter case the original data is still available along with the chain of programs to convert it – so in principle the faulty links in this chain might be repaired. Note that on-demand translation approach might be made transparent to the user.

The Internet emerged without any advance consideration of the search problem. We suggest that a complete Intermemory architecture should consider it up front. Internet search systems operate by indexing document tokens and provisions for this approach must be made. But we suggest that a system of objects might be defined to deal with part of the problem by addressing concepts such as "who", "where", "when", and so on. Some number of these might be a required, or strongly suggested component of every Intermemory document. Beyond these issues rather detailed subject area taxonomies might be developed and maintained by experts so that Intermemory contributions can *self classify*. Documents without classifications might later be classified by Intermemory robots, or human experts. Even the design of a system of taxonomies that can be revised over the years but remain connected to past versions seems nontrivial and represents an interesting subproblem.

5 Conclusion

We have identified a general framework and many issues relating to the design of an Intermemory. Preliminary calculations indicate that Intermemory is feasible and practical. That is, one can trade moderate-term commitment of memory resources for archival storage in Intermemory. The detailed design of this system will require the combined efforts of distributed algorithms, cryptography, and systems experts.

In this paper we concentrated on world-wide Intermemory. Archival Intermemories are possible on a smaller scale. For example, a large organization may have an institution-wide Intramemory as a supplement to normal backups; one advantage of this is automated restore operation. Electronic journal publishers may

construct an archival journal Intermemory, to ensure that a journal issue is easily available even if its publisher is out of business. Some aspects of small-scale Intermemories, such as security, may be simpler than those for the world-wide Intermemory. A small-scale Intermemory makes a good test case for the first implementation.

A small working group is now meeting regularly at NEC Research Institute to further define Intermemory and work towards implementing it.

Acknowledgments

The authors thank Baruch Awerbuch, Sam Buss, Jan Edler, Allan Gottlieb, Leonid Gurvits, Joe Kilian, Satish Rao, Scott Stornetta, and Herman Tull, for helpful discussions and comments.

References

[1] G. Agrawal and P. Jalote. Coding-based replication schemes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(3):240–251, March 1995.

[2] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory*, 42(6):1732–1736, November 1996.

[3] R. J. Anderson. The eternity service. In *Pragocrypt 96*, pages 242–252. CTU Publishing, 1996.

[4] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[5] C. A. Asmuth and G. R. Blakley. Pooling splitting and restituting information to overcome total failure of some channels of communications. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 156–169, New York, 1982. IEEE Society.

[6] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, 1968.

[7] E. A. W. Budge. *The Mummy, a handbook of egyptian funerary archaeology*. KPI Ltd., 1987 edition edition, 1893.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, chapter 32. MIT Press, 1994.

[9] B. Kahle. Preserving the internet. *Scientific American*, pages 82–83, March 1997.

[10] R. Mukkamala. Storage efficient and secure replicated distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(2), April 1994.

[11] M. Naor and R. M. Roth. Optimal file sharing in distributed networks. *SIAM Journal on Computing*, 24(1):158–183, February 1995.

[12] A. Odlyzko. Tragic loss or good riddance? the impending demise of traditional scholarly journals. *Notices Amer. Math. Soc.*, January 1995.

[13] D. A. Patterson, G. Gibson, and H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proc. ACM SIGMOD Conference*, pages 109–116, June 1988.

[14] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, April 1989.

[15] J. R. Roche, R. W. Yeung, and K. P. Hau. Symmetrical multilevel diversity coding. *IEEE Transactions on Information Theory*, 43(3):1059–1064, May 1997.

[16] J. Rothenberg. Ensuring the longevity of digital documents. *Scientific American*, pages 42–47, January 1995.

[17] B. Schatz and H. Chen. Building large-scale digital libraries. *Computer*, 29(5):22–26, May 1996.

[18] N. J. A. Sloane. Proposal for an internet service: The eternal home page. Technical report, AT&T Labs - Information Sciences Research Center, 1996 (Revised 1997).