HEWLETT PACKARD

DataMesh™ — scope and objectives: **a commentary**

John Wilkes

19 July 1989

HPL–DSD–89–44

This paper is a commentary on *DataMesh — scope and objectives* [Wilkes89], explaining the reasoning and conclusions of that document in greater detail.

A DataMesh is a *large, fast, highly functional storage server* built in a particular way: every storage device will be paired with a high performance VLSI microprocessor, each of which will be linked to its peers by a high performance, extensible interconnect.

A DataMesh can be viewed equally well as a storage server with a lot of local processing power, or as a highly-parallel computer with embedded storage. A physical manifestation might be as a wall-sized array of storage / processor nodes (discs, optical drives, page-addressable RAM, etc) linked together by a web-like interconnect. Such a thing would live in the basement, and serve the storage needs of an enterprise of tens to hundreds of people. A DataMesh will:

- scale to large amounts of secondary storage;
- use processor and storage parallelism to deliver high performance;
- use processor parallelism to provide higher functionality than traditional storage servers.

1 Objectives

The *DataMesh project* is researching the best way to construct large, fast, highly functional storage servers. Along the way, it will investigate the optimum algorithms to use inside them and the right services for them to provide.

The DataMesh architecture is designed for use in a Co-operative Computing Environment (CCE) where users collaborate to access, manipulate, and update the information resources of an enterprise. Computing resources are physically distributed through the physical plant of the enterprise, although the computing infrastructure appears almost seamless to its users: they will rarely need to be concerned with the physical location of information or computing resources.

Access to the CCE will be via a local computer with a display and moderate-to-large amounts of processing power. Such machines might span the range from high-end *workstations*, with fast, highly-functional graphics coprocessors and considerable local processing power, to the computer equivalent of a telephone (what I call a *PeopleStation)*: there'll be one on every desk, but there will also be "public" ones that anybody can use to access the system if they happen not to be in their office. In either case there will be enough processing power to do the commonly-performed tasks, but individual machines will increasingly be dependent on resources elsewhere in the network, such as databases, computation servers, network gateways, and so on.

The DataMesh will provide the shared, common storage repository for a CCE. It will hold files (both UNIX-like byte streams and record-oriented formats), structured and versioned data (e.g. as used in software engineering environments), objects of varying complexity, traditional databases, and multimedia elements such as images and voice—in short, all the storage needs of a collaborative working environment of the mid to late 1990s.

One DataMesh can be expected to serve between a few tens and a few hundreds of users. Once an enterprise grows beyond this size multiple DataMesh servers will be connected together in a loose federation, much like file servers are today. Some organizations (e.g. the Library of Congress) will use a DataMesh to provide services to clients across the global internetwork.

1.1 Large

The DataMesh size goals (0.1–10 Terabytes of online data) correspond to roughly 1 to 100 Gigabytes per client system. Most of this would be stored on the cheap, high-capacity devices that will become prevalent between now and 1995 (such as optical jukeboxes). These will enable cheap online storage of tens of Gigabytes of data—albeit with multi-second access times.

To obtain acceptable performance, perhaps 10% of the online storage will be in the form of higher-speed devices like discs—probably a smaller proportion at the larg-

UNIX™ is a registered trademark of AT&T Bell Laboratories in the US and other countries.

er DataMesh sizes. If we assume that discs will be about 1 Gbyte in size and that 100 Gbyte bulk-storage devices will be available, a DataMesh would consist of between 10 and 200 storage devices.

For calibration, IBM believes that its “median commercial account” will have 300 MIPS of processing power and 1.2 Terabytes of online storage by 1990—five years before the DataMesh target timeframe [Gelb89]. Technical environments are currently less storage intensive than commercial ones, but it seems reasonable to suppose that the explosion in storage requirements that we are currently witnessing will continue—especially as more of the “large system” functions like database technology are adopted in the technical environment.

The size of the smallest “introductory” DataMesh will be determined by the relative costs of regular discs and DataMesh storage nodes: the only advantage of a single-node DataMesh is its data management functions, and it will doubtless cost more than a simple disc since it has more hardware. The functions available from a large DataMesh (with an array of storage devices) are the ones that will really make it worthwhile. This might happen for as few as four nodes, perhaps eight, although the exact number is not all that important—it will certainly be greater than one.

At the high end, the maximum size for a DataMesh will be determined by when it stops being cost-effective to add more nodes. As a DataMesh gets very large (a few hundred nodes), it will exhibit *scalability failure*: the benefit from adding each additional node will get smaller as the costs of managing the nodes and maintaining consistency between the data items outweigh the benefits of adding more resources. Some of this decline will be due to limitations in the internal interconnect (e.g. bandwidth constraints); some will be due to the resource management algorithms being optimized for the more common case of a few tens of nodes.

1.2 Fast

Processing power (cpu speed) is increasing faster than physical storage device transfer rates and access speeds. This disparity in speeds is widening, rather than narrowing—and the ratio of primary to secondary memory speeds is already in the region of ten thousand to one. A key requirement for storage systems of the future is to minimize the effects of this gap.

The performance metric cited in the *DataMesh — scope and objectives* document [Wilkes89] was that the DataMesh provide a storage service that was at least as fast as that available locally, so that clients would *want* to migrate their data into the DataMesh, even if it provided no additional functions beyond simple data storage. This requires that the DataMesh use its parallelism and local processing power to provide higher performance than individual clients can.

Two trends will make this particularly challenging: improvements in the semantics of distributed storage systems so that they better approximate the behaviour of a uniprocessor—the *single system image* model—and the continually increasing performance of client nodes.

1.2.1 Shared-storage semantics

Prior work in studying file servers has found that server CPU load is the dominant parameter in their performance [Lazowska86, Howard88]. Techniques such as “stateless” servers (NFS [Sager85, Sandberg85]), or whole-file copying (Andrew [Howard88]) require less overhead in the server, but they achieve this at the expense of degrading the semantics of the file system interface from the single-machine case. As a result, they can only support a limited class of applications: those that don’t update shared data, and, in Andrew’s case, those that only access files small enough to fit in a client machine’s local disc cache.

But this will not be viable in storage servers for the 1995 CCE market: much software relies on (or would much prefer to rely on) single-system image semantics when sharing data with other processes. Besides, recent research work has shown that it is possible to achieve performance as good as the simpler schemes by careful use of cooperative caching between the storage server and its clients. Examples include HP’s Distributed UNIX product (HP–DUX) [Bartlett88a, Gutierrez88] and UC Berkeley’s Sprite [Nelson88]. Even simple modifications to the stateless NFS protocol can produce significant improvements in performance at little cost [Srinivasan89].

1.2.2 Client capacity trends

Performance needs for storage servers are also being driven by changes in the client computers. Today’s 10 MIPS workstations will be replaced by 100 MIPS machines in the 1990s, and today’s 2 MIPS personal computers and X display-servers will have evolved into 20 MIPS PeopleStations. At the same time, main memory sizes will have grown: perhaps 1 Gbyte of RAM will be common for high-end machines, 64 Mbytes for low-end ones. These changes will have two effects: first, latency requirements will become even more stringent, because I/O latencies represent increasingly large amounts of wasted resources as processors become faster. Second, larger main memories will cause caching to be used more aggressively, reducing *some* of the traffic to and from a central server. Counterbalancing this will be an increase in the amount of shared data, higher degrees of concurrent access, and greater use of high-volume data such as images and voice.

1.3 Highly functional

A DataMesh will need to be more than a simple repository for blocks of data. It will be operating in an environment where availability of shared data will be vital to the effective operation of an enterprise, where ease of use will be essential because the majority of users will not be trained computer specialists, where incremental growth will be required to allow storage capacity to expand gracefully in size and throughput, and where new device types and storage subsystem functions will continue to be introduced.

1.3.1 High availability

Any centralized service has to be *available, accessible, robust* and *secure* in direct proportion to people's dependence upon it. Because a DataMesh will serve as the repository of all the shared data needed to support an enterprise, it's crucial that it be excellent at all four. (In the commercial environment, it is commonplace for a factory to have to close for the rest of a shift if the computer system breaks down for more than a couple of hours. The associated losses are usually larger than the cost of the computer system.)

Available data is accessible to its clients even in the face of partial faults (such as a disc head crash, or a power outage). Availability is typically measured in terms of Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR). To be highly available, the DataMesh will have to be *fault tolerant:* it must mask the effects of faults, presenting the illusion of a functioning system to its users.

Successfully meeting MTBF and MTTR goals requires careful use of techniques such as:

- *redundancy*—maintaining multiple copies of a resource such as a data item, communications link, or processing element;
- *fault detection*—noticing that a fault has occurred before its effects can propagate (e.g. detecting a corrupted storage management algorithm before it has overwritten valuable data);
- *fault isolation*—containing the effects of a fault so that other parts of the DataMesh can continue operation;
- *rollback*—undoing partial actions that cannot be completed because of the fault (e.g. if one of the files being updated by a multi-file update is made unavailable because a device loses power, the other files should be put back to their state before the update began rather than allow a partial update);
- *recovery*—redoing actions that had been (unintentionally) undone by the rollback;
- *rebalancing*—returning to the steady state, and reapportioning the workload between the remaining resources.

High availability needs to be selectively applied: not everything needs to be protected with equal care. Some items (such as temporary work files) can be trivially recreated; for others, a half-day recovery period might be acceptable if it obviates the need to keep two copies online. Since very high availability is potentially expensive, the precise degree should be under user control. But it must be there if it is needed.

Accessible data must be reachable from anywhere in the CCE: whether a user is at their "home" workstation, or accessing the network from a public PeopleStation; whether they are working on an HP-UX workstation or a non-UNIX timesharing machine; whether they are on the same local area network as the DataMesh, or on the other side of the continent; whether the DataMesh is in its normal state or is being repaired or backed up.

Robust means that external failures (e.g. in client machines, or client-supplied software) are not allowed to compromise the integrity or availability of data stored on the DataMesh.

Secure data is protected against inadvertent or malicious access by unauthorized people or programs. The DataMesh, as a repository shared by many users, will need to ensure that the security and integrity of the data it holds is not compromised.

1.3.2 Structured data

The trend to complement today's simple flat-file systems with support for more structured data is well under way. Various forms of structured data (from application-level record-structuring techniques, though system-supported access methods, up to full traditional databases) have been used in the software engineering and commercial data-processing communities for several years. They are becoming increasingly popular in the technical environment (e.g. in CAD/CAM) as understanding of their benefits grows and their performance properties are adapted for the new applications.

The reasons are many-fold: the structure of data is itself information that would otherwise be hard to represent; structured data allows more sophisticated support tools and techniques to be used (such as inter-element consistency constraints); and typesafe data protects against inadvertent misapplication of a tool or update to the wrong object.

Since not every structured data type can be foreseen, the DataMesh will need to have mechanisms to allow new data types to be added in the future. A sufficiently general scheme would include provision for user-specified, polymorphic, typesafe, dynamically loadable data type support, such as described in [Wilkes88].

Supporting structured data in the DataMesh will allow it to take advantage of knowledge about the structure of the data to improve performance, parallelism, and availability (e.g. splitting a large relational table at tuple boundaries when doing disc striping).

1.3.3 Scalability via incremental growth

A DataMesh must be able to grow with its user's capacity and performance needs. There are two sub-goals: a wide range of sizes to support many different user communities; and a low-cost upgrade path that approximates a cost proportional to capacity, without large hiccups in the cost curve.

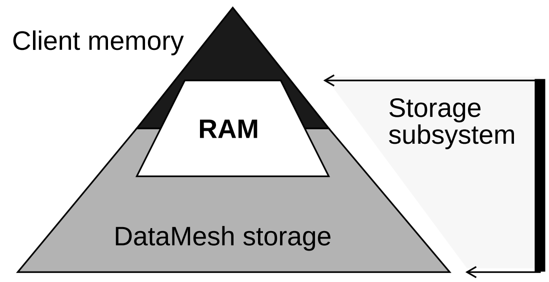
It should be possible to purchase a new storage element, hook it up, and have the DataMesh start to take advantage of it immediately: if the workload needed rebalancing, that would occur automatically; if data should migrate to take advantage of the new resource, that would occur as needed; if the storage hierarchy could be rebalanced to good effect, that would take place too.

1.3.4 Storage hierarchy

The goal of a memory, or storage, hierarchy is to present the illusion of an array of fast, high-volume storage devices using only large slow devices and small fast ones. Such techniques are already widespread in the processor world, where the hierarchy consists of a few very fast main registers (order of 10 ns access time), a few hundred kilobytes of fast cache (50–100 ns), several megabytes of main memory (500 ns), and many hundreds of megabytes of discs (10–20 ms). A DataMesh will extend this support into more levels of the storage hierarchy. For example:

- page-addressable bulk RAM (about 1/3 the cost of normal RAM, but only accessible in page-sized chunks);
- high-cost, high-performance discs (e.g. multiple arms, restricted seek distance, magnetic drums, fixed per-track heads);
- magneto-optical discs (ten times the track density of magnetic discs, but half the linear density, larger seek-settling and write times);
- optical WORM devices (very high capacity, but only write-once, and large access times);
- CD-ROMs (read-only, slow);
- optical jukeboxes (very large aggregate capacity, but multi-second access times if the disc needs changing);
- new tape media, such as Digital Audio Tape.

A DataMesh does not stand alone: there will be considerable collaboration between the storage hierarchy management algorithms in the DataMesh and those used by the storage subsystem components of its clients. The two storage hierarchies overlap at the level of their respective main memories.



1.3.5 Multiple device types

A large DataMesh will represent a substantial financial investment. To preserve this investment, the DataMesh must be capable of evolving incrementally to accommodate the latest storage technologies as they become available, and never have to be replaced *en masse*. Extensible support for multiple device types will make this possible—ideally, even easy.

The capacity to evolve gradually through incremental replacement will also allow a DataMesh to take advantage of new processor and RAM technologies. By using standard parts, rather than a special-purpose processor design, it will be able to keep pace with the state of the art in VLSI commodity processors, and not accidentally find itself relegated to a technology backwater. (This is a conscious difference from some work that has advocated specialized processing engines and coprocessors for storage servers such as database engines [Glavitsch89].)

1.3.6 Offloading data-intensive work from clients

There are five main ways to take advantage of the processing power available in the DataMesh:

1. *speeding up "traditional" data access operations* through techniques such as disc striping, storage hierarchy management, and data replication;
2. *providing additional storage management functionality*, such as increased availability (through stable storage, mirroring, transactional techniques, and parity discs);
3. *preprocessing raw data* into a more information-rich form by filtering it before it is dispatched to the client;
4. *taking advantage of processor-intensive algorithms to improve I/O performance* (e.g. using data compression in the higher levels of the storage hierarchy, thereby increasing their effective size);
5. *using "spare" computation cycles* in the DataMesh to offload computations such as database queries from a client machine.

To do any of these, the DataMesh will need to know about the structure and content of the information it is manipulating. For example, with knowledge of logical record boundaries a DataMesh could log logical updates rather than physical pages on transactional data, thereby greatly reducing traffic to the log (an important bottleneck in high transaction-rate systems [Stonebraker84]).

The third technique, that of filtering data near its source, is an obvious one to apply at each storage element. Tandem has used this technique to great effect in their disc drivers, which perform a first-pass filtering of relational tables as they are read in, thereby substantially increasing the performance of their database [Borr88]. ICL disc controllers support a *content-addressable file store*, allowing a disc to be searched associatively. For these kinds of data filtering, the filtering algorithms will reside in the DataMesh, while their search parameters will be supplied by clients in their requests.

Once a simple filtering mechanism is in place, it isn't hard to generalize. For example, a DataMesh could support arbitrarily-interconnected graphs of filtering computations—much as a relational database query processor does. Note that this offloading of function into the DataMesh is a small extension to the functionality that will be present anyway to handle structured data.

A more aggressive variant is to allow clients to download their own filtering algorithms into the DataMesh. This raises several difficult questions, such as how to maintain the integrity and availability of the DataMesh while retaining the low overheads that resulted from it containing only trusted software.

If the protection issues could be addressed successfully, it might be thought worthwhile to execute arbitrary computations in the DataMesh to offload its clients. But this would be a mistake. It would weaken the DataMesh emphasis on being a *data* server, dedicated to providing optimal data access performance to its clients. It would complicate the resource management and optimization algorithms. It might even require backwards compatibility with client execution environments such as UNIX.

The processing power in the DataMesh is there so that data access can be speeded up, but it is so cheap that if no other sensible use can be made of it, that's no great loss. To use an analogy first expounded by Robbert van Renesse, simply because a modern kitchen is well populated with Z80s doesn't mean that we feel the need to apply load-balancing algorithms to them! If a CCE installation needs more general-purpose computing power, it should invest in more computation engines or a dedicated processor bank.

1.3.7 Client-specified data properties

A key feature of IBM's announced data management support for the 1990s is the concept of *System Managed Storage* [Gelb89]: rather than requiring users to specify where the system should put their data, how many copies of it there should be, and so on, the system itself takes care of these details by deducing them from a set of user-supplied *data behaviours*.

In a DataMesh, the following benefits would result:

1. It will be easier to achieve particular levels of performance and availability. Users don't have to understand detailed device specifications and global resource management tradeoffs to get the effects they desire.
2. The DataMesh will make better fine-tune placement and storage hierarchy decisions than a human administrator, thereby increasing the effective storage capacity for a given performance level. An external administrator has to take a conservative approach when faced with a complicated set of potentially-conflicting needs, whereas the DataMesh can use its better internal knowledge.
3. If there are unused resources in the higher levels of the storage hierarchy, the DataMesh will take better advantage of them to provide more-than-minimum performance. An external allocator has to assume the worst case.

Data behaviour properties occur in two forms: *absolute requirements* that the DataMesh must meet; and *hints* that the DataMesh can use to optimize its behaviour.

For example, a user might require that a set of KSAM files be available with 99.999% probability, with no more than a 10 second recovery time, an 100 ms (or better) access time under no-fault conditions, and a 20:1 ratio of reads to writes. The DataMesh might choose to implement this by mirroring the data on a pair of disc drives on DataMesh elements that did not share a common power source, but caching the index in page-addressable RAM. (Log-based recovery techniques would probably be ruled out by the recovery time considerations; the page-addressable RAM would be needed to provide a fast-enough access time; and mirroring would be necessary to achieve the required level of availability.)

Another user (or a program like a compiler) might indicate that a set of files was transient—and could as easily be recreated as replicated. Such files might never be allocated disc space, and would participate only in a limited fashion in transactional updates (cf. non-recoverable storage in Quicksilver [Haskin88]).

A third user might suggest that some data would almost never need to be accessed again a month after its creation date, but that it should be kept for auditing purposes “just in case”. At the end of the month, the DataMesh might choose to migrate this data to a lower-cost tertiary storage device such as an optical jukebox.

Finally, the DataMesh itself might notice certain access patterns (e.g. a particular set of files always accessed in sequence), and remember performance hints for future use. These would enable it to optimize subsequent accesses—e.g. by prestaging the remaining data in the set whenever one of the files is accessed.

1.3.8 Automatic internal workload balancing

A key premise of the DataMesh design is that it should be continuously optimizing its internal resource allocations to accommodate external requests. The resource balancing has to be dynamic to handle:

- slowly-changing workloads (e.g. the interactive use that predominates during daylight hours has a different kind of access pattern than the daemons that run in the small hours of the morning);
- rapidly-changing loads (e.g. when a client system is restarting, it will change the overall traffic patterns and the data being accessed until it gets past the boot phase and has filled its caches);
- newly-introduced resources (such as a storage element);
- failures of processing elements, storage device and interconnect links (internal or external).

1.4 Storage server

Future CCE systems will have much greater support for cooperative work, where information from multiple sources is simultaneously interwoven into multiple calculations, analyses and data presentations, while itself being updated. Shared storage is the enabling technology for this, and the CCE thrust towards server-based architectures means that storage servers will be the preferred way to support it.

2 Basic premises

This section outlines the technology trends behind the DataMesh proposal.

2.1 Chip performance

2.1.1 Dynamic RAM

The semiconductor industry continues to emit projections for a factor of four increase in storage capacity every three years [Motorola89]: by 1995, production will be ramping up on the 16 Mbit dynamic RAM parts that will have been introduced in 1991–2; production of their predecessors (4 Mbit DRAMs) will have reached a peak in 1994.

2.1.2 Client machines

The performance of high-end workstation processors is currently increasing at a compound annual growth rate of about 70% [Rosenbladt89]. The top of the line workstations will push performance upwards by the use of multi-chip processors, very fast backplanes, large caches, and so on.

Commodity workstations—those that every engineer can expect to have on their desk—will advance somewhat more slowly because of the difficulty of scaling memory bandwidth in the face of a constant number of memory chips (memory just gets larger, it doesn’t get much faster). Even assuming a fairly conservative rate of performance improvement (a compound annual growth rate of 40–50%), such machines will deliver at least 40–60 MIPS by 1995, starting from today’s 5 MIPS machines. Of course, not every engineer will have the latest product sitting on their desk: depreciation schedules of 2–3 years will ensure that maybe half of the machines will be of the latest generation, while the rest are from the previous one, with 50–75% of the performance. The “typical” machine might thus have about 40 MIPS.

2.1.3 Commodity processors

Low-end microprocessors will continue to perform a multitude of controller-like functions. Like today’s “workhorse” chips (e.g. the 680x0 family), they will be built using high-yield semiconductor processes for low cost, and their processing power will be determined by how much can economically be put onto a single VLSI chip. With current trends, by 1995 a 15–20 MIPS processor in a single-chip package will be as readily available (and as cost-effective) as a 68000 is today. (For calibration, a recently-announced 12.5 MIPS 32-bit Transputer currently costs $269 [Inmos89].)

2.2 Storage device performance

The accompanying graphs show that magnetic disc dynamic performance will soon reach physically-imposed limits [Shula89]. For example, disc seek times have bottomed out at about 16–17 ms for 5.25" discs, plus 8 ms of rotational latency. (3.5" discs will probably eventually reach 7 ms seek time and 5 ms rotational latency.) Transfer rates will peak at about 4.5Mbytes/s with currently-foreseeable head and data channel technologies.

The improvements being made to discs are concentrating on making them physically smaller, increasing their linear and track recording densities, and increasing their reliability. Production costs are being driven down by manufacturing techniques aimed at meeting the volume needs of the personal computer marketplace, and specialized disc technology will be restricted to use with supercomputers and commercial data processing mainframes.

By 1995, a typical device will be a 3.5" Winchester drive with 1 Gbyte on a single spindle and an MTBF of 120,000 hours or more. Seek and rotational latencies will together be about 16 ms, and the data transfer rate will be 4 Mbytes / s. A single track will hold 32 Kbytes of data, and sectors will have grown to about 2 Kbytes in size from their current 0.5–1 Kbytes.

2.3 Storage subsystem I/O demands

Suppose we take the 40 MIPS commodity workstation as the typical DataMesh client. Applying the “1 MIPS needs 1 Mbyte memory and 1 Mbit / s I / O bandwidth” rule of thumb, such machines will be demanding an average I / O bandwidth of 5 Mbytes / s while active, and a peak bandwidth of 15 Mbytes / s (I / O traffic is typically quite bursty). Fifty such machines would require about 250 Mbytes / s aggregate I / O bandwidth, and perhaps 50 Mbytes / s more to handle peak I / O rates if these affect up to 10% of the clients at a time. (Of course, not all the 50 machines would be simultaneously active, so, depending on the work patterns of their users, 50 active clients might correspond to between 70 and 200 users.)

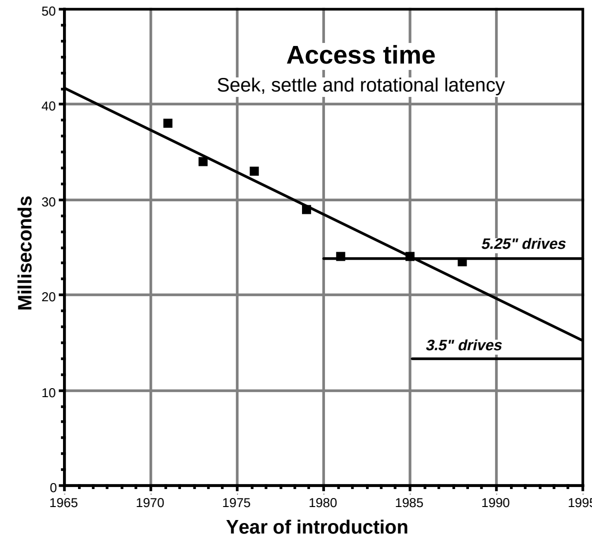
If I / Os are done in 32 Kbyte chunks (i.e. the track size) the average throughput will drop from the maximum 4 Mbytes / s to around 1.3 Mbyte / s (16 ms seek+rotational latency, 8 ms data transfer time). Supporting an I / O throughput of 300 Mbytes / s in a DataMesh would require 230 such spindles, or about 4.6 Gbytes per client machine.

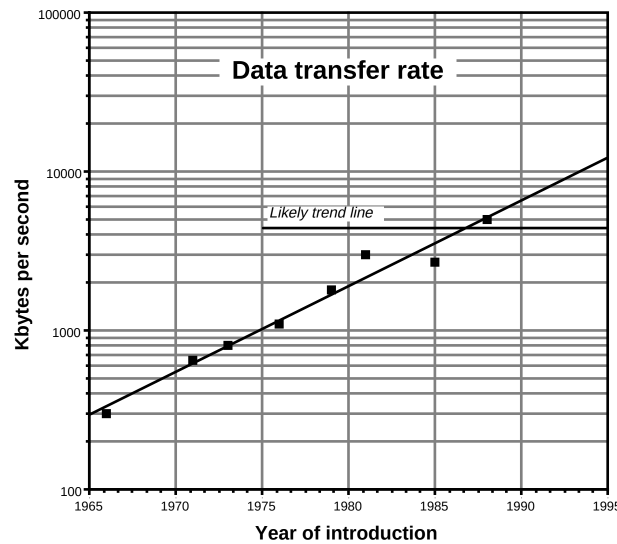
There are conflicting pressures on the transfer size: the heavy preponderance of small files [Satyanarayanan81] tends to encourage smaller disc transfers; the desire to increase throughput encourages larger ones (but these incur between-track seek times of perhaps 5 ms). The table below summarizes these effects.

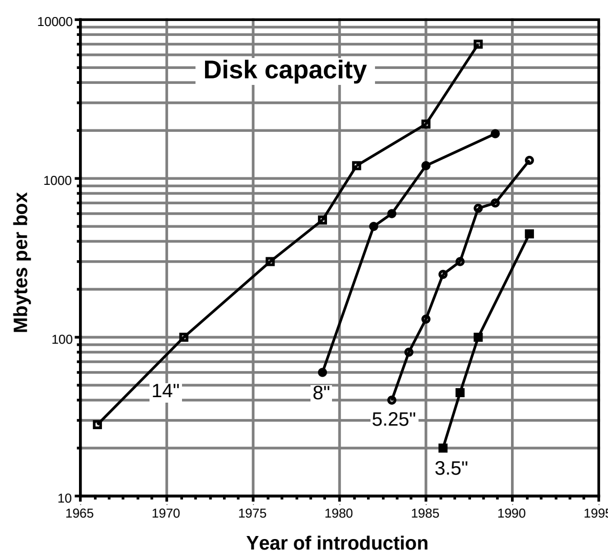
Transfer (Kbytes)	Throughput (Mbytes/s)	Spindles	Capacity (Gbytes/client)
8	0.4	675	13.5
16	0.8	375	7.5
32	1.3	225	4.5
64	1.7	174	3.5

If discs that can read or write simultaneously to all the heads in a cylinder become available, the seek and rotational latencies will remain unchanged but the data transfer time for each 16 Kbytes will drop to 0.5 ms (eight heads transferring 2Kbyte sectors in parallel). This table shows the result.

Transfer (Kbytes)	Throughput (Mbytes/s)	Spindles	Capacity (Gbytes/client)
8	0.5	610	12.2
16	1.0	310	6.2



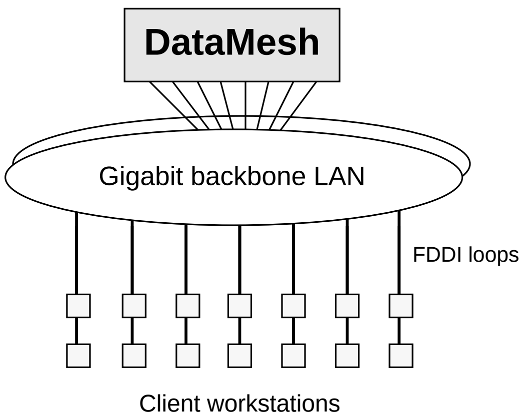




32	1.9	160	3.2
64	3.6	85	1.7

2.4 LAN hardware performance

The next generation of networks (like FDDI) only offer a factor of ten improvement in throughput over today's LANs (e.g. IEEE 802.3). Nevertheless, by 1995, such networks will be widespread, and the first of the next generation LANs (with between 1 and 10 Gbit / s links) will be starting to be installed. This is just as well: the hypothetical 300 Mbytes / s I / O rate for a 50-client DataMesh needs about 40 FDDI rings to service it assuming 60% LAN utilization, but could be easily accommodated by a single 10 Gbit / s LAN, or five 1 Gbit / s LANs.



Even so, since the I / O capacity of each DataMesh node will be limited, the DataMesh will probably need on the order of 10–30 taps onto the 10 Gbit LAN, and FDDI may still be used to hook onto a backbone LAN using the multi-Gigabit hardware, partly for cost reasons, partly because FDDI networks will already be installed. In any case, availability considerations will also encourage the use of multiple LANs even if a single one could handle the traffic.

2.5 Parallelism, caching and data synthesis

Since the performance gap between individual processors and storage devices is widening, new techniques must be pressed into service to improve the effective performance of a storage hierarchy. All of these rely on some form of parallelism, caching, or data synthesis:

1. *Access multiple devices simultaneously*, rather than a single device serially. The overall latency will be that of the slowest single operation, rather than the sum of the individual costs, and the data transfer rates will increase to the sum of the individual rates. Techniques such as disc striping (disc interleaving) do this [Patterson88].
2. *Replicate data*, and try to retrieve each copy simultaneously: the copy with the shortest access time will determine the overall latency. Data can be replicated across a single disc, and / or across multiple devices [Kure88].
3. *Insert intermediate levels in the storage hierarchy*, such as page-addressable RAM, to reduce the size of the performance gap between processors and storage devices.
4. *Avoid accessing secondary storage:* reduce the need to retrieve or store data by *caching*: putting copies of frequently-accessed data in higher-performance levels in the storage hierarchy, such as RAM. Caching is a specialized kind of data replication that is effective because of a common property of data accesses: *spatial and temporal locality* ("nearby" data items are frequently referenced together over a short period of time). Caches typically become more effective as they get larger, although there is usually an upper bound beyond which they provide little benefit because of this locality [Ousterhout85].
5. *Data reduction*: use data compression techniques to reduce the number of times a secondary storage device needs to be accessed to retrieve a given quantity of data. (Examples: Lempel-Ziv data compression; index compression by elision of leading common substrings.)
6. *Data synthesis*: more radically, synthesize or compute data when needed from a small set of parameters, rather than store it in its entirety. (Example: rule-based information systems.)

The DataMesh will perform all these functions. Its architecture makes the data reduction and data synthesis schemes possible because their computations can be carried out in parallel.

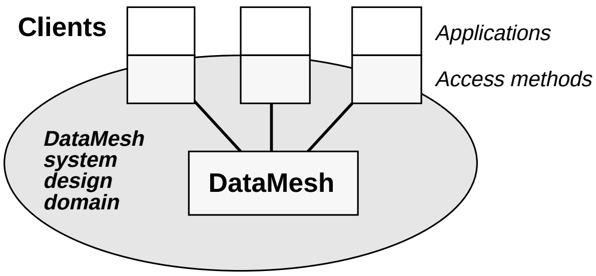
2.6 The DataMesh premise

Processing power and RAM will be cheap, because it will be available in mass-produced, single-chip packages. Storage will remain relatively slow. Adding a processing element to each storage device will be cheap. Linking them together into a DataMesh will provide a highly-parallel storage server platform. With suitable software, the result will be much greater performance and functionality at little incremental cost.

2.7 Storage server clients

Most applications will not communicate directly with the DataMesh, but rather interface to it via an *access method* interface running on their client node. As used here, the term access method includes system-level functions such as UNIX-like file systems, record-oriented file types (e.g. as used by IBM's MVS, HP's MPE, or DEC's VMS), and databases. A *few* specialized applications might make direct use of the DataMesh interface (e.g. image-manipulation services and parallel grep).

Caching by the access methods will be an important technique. For optimal performance, the DataMesh and access methods will collaborate, and may even be designed jointly. The DataMesh *system design* has to include parts of the DataMesh's clients as well as the DataMesh itself.



The first releases from both OSF and Unix International will have well-defined internal interfaces to make it easy to add new file system types [Hurwitz89]. The widespread propagation of these implementations will encourage the development of new file system variants (such as "stateful" versions of NFS, or the Andrew file system [Howard88], or Sprite [Nelson88]). By 1995, this will be well-established practice.

The DataMesh will need to support standard client implementations such as the new UNIX file system variants, but it may be able to offer better service to client file systems that understand they are talking to a DataMesh rather than a normal file server. It may even be advantageous for the DataMesh project to try steering the standardization efforts for file servers in desirable directions as our experience evolves.

2.8 Standards-dominated interfaces

The server-dominated nature of CCE architectures, and the need to allow multi-vendor CCE implementations, will increasingly dictate the standardization of interfaces between services.

Such standards will come to dominate all inter-server communication at both hardware and software protocol levels. Consistent with current trends, these standards will be designed to meet the needs of local-area networks (LANs), but they will be based on protocols that allow communication across wide-area internets (e.g. the OSI reference model and its protocols).

The very nature of the standards and the huge investment that their pervasive installation will represent will constrain changes to occur very slowly. The form and properties of the connections between a DataMesh and its clients will thus be determined largely by the then-current standards, not the available state-of-the art.

2.9 The internal DataMesh interconnect

There will be two kinds of internal DataMesh interconnect: that linking the DataMesh nodes themselves, and that between the processors and the storage devices. In both cases, there will be opportunities to exploit non-standard, higher-performance links.

The pressures for picking a standard microprocessor are induced by the enormous investment required to make such devices with comparable performance and cost as the commodity microprocessor vendors. The pressures for picking standard server–client connections result from multivendor operability needs over a period of several years to allow the large investments made in network installations to bear fruit. In contrast, the internal DataMesh interconnects will be much simpler to develop than a microprocessor, and they don't have to conform to (or achieve the status of) external standards because their use inside the DataMesh prevents external dependencies on their details from being built up. The result is that some DataMesh-specific optimizations are possible, and the interconnect can more easily track the state of the art, rather than be beholden to the state of the standards.

2.9.1 Specialized interconnect techniques

The small physical size of a DataMesh (perhaps 3 m in diameter) means that its internal interconnect can use techniques that work well inside a machine room, but that don't necessarily scale well to the larger distances typical of a LAN (1 km or so across). Such techniques frequently offer performance advantages from lower latencies, less heavyweight link protocols, and lower error rates. The net result should be an interconnect that is very much faster than a general-purpose LAN.

As currently envisaged, a DataMesh is a homogeneous collection of processors tied to a heterogeneous collection of storage elements. This homogeneity will make it easier to use techniques like remote memory access, which provides low latency communication (10–20 microseconds compared to the 1 ms or so of message-based techniques like remote procedure call). The DataMesh will benefit from remote memory access in two ways:

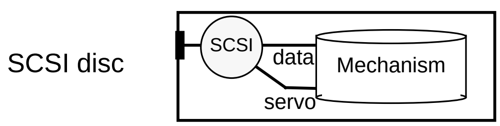
1. algorithms that require access to all the distributed information about an object (e.g. finding the most up to date copy of an item in the DataMesh) will run faster, because they can gather the data they need much more quickly;
2. algorithms that make a decision whose quality depends on the number of samples they can take will be able to do a better job in a fixed amount of time (e.g. an algorithm trying to choose the best place to position a data block could poll an entire 100-node DataMesh using remote memory access in the time

it would take to query two nodes with message-passing).

It is even possible that two different internal networks may prove optimal: one for control traffic, the other for bulk data transfer. In any case, the DataMesh interconnect will not be handicapped by the need to conform to the external (and only slowly advancing) protocol standards that will constrain general-purpose LANs. One suggestion (due to Bob Rau) is to pick the emerging IEEE Scalable Coherent Interface (SCI [Kristiansen89]) as the DataMesh interconnect, although it is not yet clear what the cost implications of this will be.

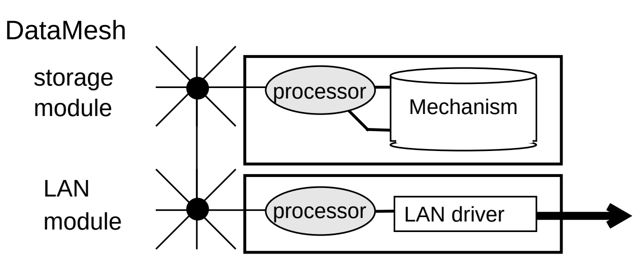
2.9.2 Secondary storage device connection

Typical secondary storage devices contain a *mechanism* component that has two data channels: a digital data stream, and a servo control interface.



The servo controller itself is integrated into the mechanism. Current HP disc drives contain a three-chip SCSI controller and interfacing set, a power supply (not shown), and a SCSI cable connector. Soon, several kilobytes of RAM for track buffers and read caches will become commonplace.

Since a DataMesh node needs to be cost-comparable with traditional storage devices, it is probably best thought of as a “storage device with smarts”, rather than as a processor with a storage device attached. In such a device, the disc’s SCSI controller and connector would be replaced by a VLSI processor; the track buffer RAM would become the DataMesh node memory (a few megabytes instead of a few tens of kilobytes); and the SCSI protocol chips would turn into the DataMesh interconnect.



2.10 Alternate node structures

The DataMesh architecture calls for a processor attached to every storage device. Two alternate structures come to mind:

- build a multicomputer solely out of processors and an interconnect, and then attach one or more storage devices (e.g. via SCSI daisy-chains) to each node;
- specialize some DataMesh nodes by building multiple devices into them (e.g. a “mirrored volume” node, or a “disc striping” node).

Manufacturing economies of scale will favour the regularity of the DataMesh “processor with every node” architecture. The interconnect represents a significant fraction of the cost of a SCSI device, so there would be little or no cost advantage to building a DataMesh by attaching a chain of I/O devices to each processor—especially if they were in separate boxes.

The second alternate is also unattractive because it would: force a very early binding of resources to particular uses (e.g. replicated data could only go on mirrored volumes); increase the probability of non-independent device failure (e.g. both discs would become unavailable if a mirrored-volume processor node failed); and limit the freedom to do resource load balancing throughout the DataMesh. The main advantage would be simpler software—but even that would be true only if the global resource management features that make the DataMesh so effective were not supported.

It may still be the case that the DataMesh could be extended to support non-storage nodes: for example, one solution to the protection difficulties of allowing arbitrary client code to execute in the DataMesh might be to introduce special “client processing” nodes. These would be tied tightly into the DataMesh interconnect, and so gain many of its performance advantages, but they would run special software, with many more internal firewalls.

2.11 Distributed versus centralized storage

One obvious alternative to the whole DataMesh idea is to distribute the storage capacity amongst the client nodes. Such a scheme might seem to offer many of the benefits of a DataMesh, such as high availability, incremental growth, and increased parallelism, but a DataMesh is still a better solution:

- *Single-purpose service.* A DataMesh is dedicated solely to serving its clients. Individual client machines have their own users, and their own computations to run. This autonomy manifests itself in limits on how much of a node’s resources can be used to serve other users—a local optimization that favours the “owner” of a client node at the expense of lower overall performance. (For example, when Apollo provided their users with a distributed load-balancing facility, their users demanded limits on the resources a machine would make available to other nodes.)
- *Increased parallelism.* A centralized DataMesh architecture would concentrate more storage

mechanisms at one site than would be the case if they were spread evenly over many client nodes. A DataMesh will thus be better able to exploit higher degrees of parallelism in techniques such as disc striping.

- *Less wasted storage.* Handling the peak I / O needs (10–15 Mbytes / s) of a 25–30 MIPS client locally would require 6–10 discs, or 6–10 Gbytes per client (assuming 32 Kbyte I / O transfers). This is about three times as much storage as a DataMesh would need to deliver the same performance.
- *Faster internal interconnect.* The DataMesh internal interconnect will perform better than the general purpose LAN linking its clients together.
- *Tighter hardware/software coupling.* The DataMesh hardware and software architectures will be developed simultaneously, each optimized around the needs and capacities of the other. A more general-purpose software storage subsystem, optimized for portability to multiple client architectures, would not be able to take full advantage of the underlying hardware. (For example, DataMesh could use the very large HP-PA virtual address space to good effect, whereas almost all portable implementations limited themselves to 32-bit addresses.)
- *A better balanced storage hierarchy.* Since remote data access is slower in the distributed case, more local caching is needed to achieve a given performance level. The system will behave more as a set of independently-managed caches than as the single global cache the DataMesh can provide. As a result, more cache memory will be needed to achieve the same performance level, so a given storage capacity will cost more.
- *Ample processing power near each of the storage elements.* Whereas a workstation node will be under pressure to use "reasonably efficient" storage management and placement algorithms, the cost / benefit tradeoffs in the DataMesh can be slanted more heavily towards the performance benefits by using algorithms that require more processing, such as compression, to achieve higher overall performance. Similarly, techniques that trade main memory capacity for performance (such as faster space-allocation techniques) can be used to full advantage in the DataMesh, whereas they might be considered too profligate for client machines.
- *Fewer implementation overheads.* Because it runs specialized, trusted software, a DataMesh can minimize its "overhead" costs like accounting for processor cycles, heavyweight inter-process protection, and security provisions that can withstand arbitrary attacks—thereby achieving lower latency and higher throughput.
- *Lightweight runtime system.* Because a DataMesh is designed for a specific task it can be built on a specialized software platform instead of a general-purpose operating system kernel. This platform will be a small, lean, fast runtime system, using techniques like really lightweight threads, synchronization and communication mechanisms (e.g. [Anderson88, Bershad89]), only resorting to heavier-weight mechanisms where necessary. Because general-purpose operating system kernels are designed to execute in less constrained environments, they must assume the worst case by default and suffer associated additional costs.
- *Simpler algorithm implementations.* The algorithms and protocols used for high-performance locking, availability, and data placement are quite complicated. Moving them inside the DataMesh means that they can execute in a simpler environment than would be the case in a client machine, and so more of the development effort required to implement them can be invested in the algorithms and protocols themselves, and less on the "hassle factors" such as inter-system portability.
- *Extensive support for parallel, distributed algorithms.* The designers of DataMesh algorithms will be able to draw on support that will be in place in the DataMesh for controlling and isolating failures, providing access to performance measurements, and other parallel and distributed algorithms optimized for the DataMesh environment. Such libraries are harder to construct for arbitrary distributed computing environments because fewer assumptions can be made about the properties of such environments.
- *No "lowest common denominator".* Standards-dominated networks are prone to operating at the "lowest common denominator" functionality level. For example, if any machine is unwilling to participate in a fancy cache consistency protocol to improve performance, the entire computing environment may have to use a simpler, less efficient protocol. A DataMesh can act as an agent of the less capable clients in transactions with other nodes, so the nodes capable of using the higher-performance protocol do so (and go faster) while the others can still participate as full members in the shared storage subsystem.
- *Simpler power distribution and noise containment.* A Terabyte of storage (even using the new storage devices being developed) will consume moderate amounts of power and cooling, and generate sufficient noise that it will not be an ideal installation for an office-like environment. Centralizing it, and putting it into a computer room (or a basement) will minimize the impact of these problems.
- *Simplified system administration.* It is much easier to administer, repair, maintain, and backup a central-

ized storage repository than a physically distributed one.

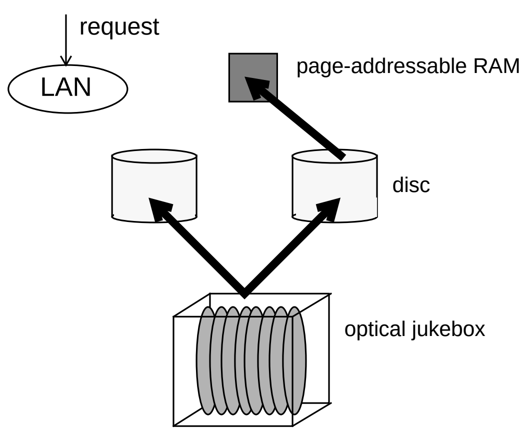
- *More resources; better utilization.* A 100-node DataMesh will provide roughly 1000–1500 MIPS and 1.6 Gbytes of RAM, essentially “for free” with the storage capacity. Although these resources will not be available for arbitrary computations, they will probably be better utilized by being a shared resource than if they were dispersed amongst the client nodes. They represent a significant additional capacity to offload storage-related processing that would otherwise have had to be run on the clients.
- *Customer capture effect.* From a marketing standpoint, once a few DataMesh nodes had been purchased, the customer would have a high incentive to acquire more of the same over stand-alone discs, which could not partake of the DataMesh performance and functionality benefits.
- *Single-system software.* Offering comparable functions in a multi-vendor environment requires that the software to provide them has to be ported onto many different vendor platforms, and in multiple releases of many different operating systems. Putting it into a DataMesh means that it can be written once, for maximum performance rather than portability.

3 Key investigation topics

This section describes the focus that the DataMesh project will apply to narrow its field of interest. The goal is to select a small enough problem domain to be able make a substantial contribution with the resources available.

3.1 Possible research areas

There are many areas where we do not currently have the knowledge required to put together a DataMesh. To help illustrate some of these, consider a simple read request for some data held in the DataMesh.



Suppose that some of the data needed to satisfy the request is in a high-level cache, some is duplicated across a pair of discs, and some still resides in an optical jukebox. The first problem is in finding all this data, since the read request might come in at any of the LAN connections in the DataMesh—not necessarily one of the nodes holding a block of the file.

Stage one will be to convert the file handle passed in into a *fileID*, or unique file identifier: a bit string known to be unique across the whole DataMesh. How this is done will depend on the format of the file handle—some clients might use the fileID directly; others might pass in a file name that has to be looked up via a directory mechanism of some sort. Authentication and security checks will be needed to prevent unauthorized access, and various checks will be needed to make sure that an intruder is not replaying an earlier conversation between a client and the DataMesh. Once that has been done, the node where the request came in can proceed to locate the data.

Simply broadcasting a request to each node to see whether it has some of the data is impractical: once the DataMesh scales beyond a few nodes, the costs of handling the interruptions would prevent each node making forward progress. Similarly, telling the whole DataMesh any time a block of data moves to a new location is impractical. Research is needed to determine the best scheme: perhaps the jukebox node is charged with knowing where its data has been copied to, and the request-processing node can try there the first time it accesses this file (or perhaps the highest-level node in the storage hierarchy should fill this role). Once this has been done, perhaps the request-handling node should register interest in the particular file (or a subrange of it), so that it can be informed of future movements of the file’s data. It’s not at all clear yet what the *best* scheme is.

Once the data has been located, the optimum way of accessing it can be computed, taking into account the relative speeds of the different levels in the storage hierarchy, and the load and head positions on each of the two discs where the data is replicated.

Perhaps the DataMesh has observed a particular locality-of-references pattern for previous accesses to this file, so it might be a good idea to start a read-ahead from the optical jukebox of the next segment that is likely to be requested. In fact, it might even be decided to copy the data needed to satisfy this request up into one of the two discs (or maybe a third!). Once these determinations have been made, the access can be executed, correctly sequenced to make sure that the data can be sent back to the client in the order it was requested. The request may need to be logged so that its properties can contribute to future access predictions. Finally, the resources acquired to execute this request can be released, ready for the next request.

Of course, this read might be part of a larger transaction, and account may have to be taken of locks held by other transactions. Perhaps the data held in the page-addressable RAM is stored in compressed form, to maximize the caching capacity. And so on. A research project into building a DataMesh will have plenty of interesting areas worthy of investigation!

3.2 Primary research areas

The hardest balancing act will be between maximizing the performance of a DataMesh and ensuring that sufficient attention is paid to the advanced functionality that will be required of a future storage server.

The research topics below are listed in decreasing order of importance: our primary focus will be on the first one, and we will spend decreasing amounts of effort on the ones further down the list.

1. *High performance through parallelism* is the primary research emphasis for the DataMesh project. This includes algorithms for parallel data access and processing, and parallel / distributed caching and storage hierarchy management.
2. *Increasing the availability of data* has been selected as the most important functionality. Researching it in parallel with the performance-oriented work is important because many performance-improvement techniques (e.g. replication) are intimately connected with availability-enhancing ones (e.g. data redundancy).
3. *Client-specified data properties* are going to be crucial for any future storage server if the complexity of system administration is not to become a barrier to the effective installation and use of cooperative computing. In addition, the importance of performance and availability properties fits in tidily with our two primary research topics.
4. *Support for an extensible set of data models* (beyond flat files and databases) is essential if a DataMesh is going to be able to evolve over its lifetime to adapt to new needs. Such support will also have significant impact on the overall DataMesh software architecture.

3.3 Secondary research areas

There are many secondary areas of interest that will guide the thinking and approach taken by a project as complex as DataMesh. Some of the items listed in this section are techniques, features, or needs that we will ponder as we go about our research, perhaps with the idea that they might be supportable in future versions, and certainly with the expectation that their consideration might cause us to alter the way we go about achieving our primary goals. The list also enumerates a number of things that we are *not* going to pursue with the same energy as the primary research areas.

1. *Designing an optimal storage hierarchy*. This would involve knowledge about the detailed characteristics of different storage devices (example: the relative performances of serial and random-access reads); research into techniques that develop—and predict—optimal allocation policies; and high-performance mechanisms for migrating data up and down the hierarchy in the face of data accesses, archiving requirements, and availability needs. All are interesting and important, but they will not be a primary focus for our work in the first stage of DataMesh, although there will necessarily be some overlap with the work on client-specified data properties.
2. *Use of non-volatile memory*. Certain types of non-volatile memories have dynamic performance properties akin to semiconductor RAM, but permanence similar to discs. Since a write into such a non-volatile memory is as good as a write to a disc from the point of view of recoverability, some operations (such as recording commit records, or intensions lists) can be greatly sped up. We would very much like to take advantage of such technology, but doing so would probably require us to invest considerable resources to develop an implementation of the needed hardware. We will, however, try to make note of performance opportunities for non-volatile memory as they arise, and may simulate some of them with ordinary RAM to investigate their benefits.
3. *Extensive performance and evaluation support.* Our work is predicated on improving performance. To understand what our algorithms, implementations, and system prototypes are doing, we will need to build in performance-gathering mechanisms, and develop suitable analysis and evaluation techniques. Again, these will be necessary, but they won't be topics of research in their own right.
4. *File systems for write-once optical devices.* Some of the highest-capacity storage devices today use optical write-once read-many (WORM) technology. Their write-once properties require specialized storage allocation policies. Although they can perform admirably in an environment where retention of prior versions, or high-capacity logging, are required, we do not plan to invest much effort into developing the specialized allocation and update techniques needed to utilize them fully.
5. *Commercial data processing and supercomputers.* The emphasis in the DataMesh activity will be on technical CCE application domains, rather than on commercial data processing or the support of single large mainframes or supercomputers.

6. *Interconnected hierarchies or nets of DataMesh servers. We* will concentrate on the functions required of a single DataMesh, rather than the interconnections that might be put into place between multiple DataMesh servers.
7. *Extensive scalability*. It is desirable for a DataMesh to cover a wide range of sizes, but maximizing this range is not an explicit goal.
8. *Supporting arbitrary client computations.* The difficulties inherent in allowing a client to download any computation into the DataMeshare considerable, and they will serve to distract and defocus effort without there being a clear benefit, so we will not pursue this avenue for some time (if at all).

4 Project plan

It is too early in the DataMesh project to specify exactly how we will proceed: the current phase is one of exploration and investigation.

We do have some ideas about our general approach, however. We will conduct a sequence of focussed experiments to investigate and understand the behaviours of storage servers and their clients, and to develop and evaluate new techniques. At the same time, we will conduct exploratory forays to extend our knowledge of how best to structure DataMesh implementations.

After a year or so, we will start to put together the results from these investigations into a series of prototype DataMesh designs. The first prototype will use conventional LAN technology, and will be used as a testbed for the correctness of some of the algorithms. The second and subsequent prototypes will hopefully be based on the specialized DataMesh interconnect, and begin to evaluate system integration (how well our ideas work together), scalability, and performance.

References

[Anderson88] T. E. Anderson, E. D. Lazowska, and H. M. Levy. The performance implications of thread management alternatives for shared-memory multiprocessors. Technical report 88–09–04 (Sept. 1988). Department of Computer Science, University of Washington.

[Bartlett88] D. S. Bartlett and J. D. Tesler. A discless HP-UX file system. *Hewlett-Packard Journal*, **39**(5):10–14 (October 1988).

[Bershad89] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy. Lightweight remote procedure call. Technical report 89–04–02 (April 1989). Department of Computer Science, University of Washington.

[Borr88] A. J. Borr and F. Putzolu. High performance SQL through low-level system integration. *Proceedings of the 1988 SIGMOD International Conference on Management of Data* (Chicago, Illinois), pages 342–9, H. Boral and P.-A. Larson, editors (June 1988).

[Gelb89] J. P. Gelb. System managed storage. *IBM Systems Journal*, **28**(1):77–103 (1989).

[Glavitsch89] U. Glavitsch. *What other people do on DataMesh.* Hewlett-Packard Laboratories technical report HPL–DSD–89–40 (June 1988).

[Gutierrez88] D. O. Gutierrez and C.-S. Lin. The design of network functions for discless clusters. *Hewlett-Packard Journal*, **39**(5):20–6 (October 1988).

[Haskin88] R. Haskin, Y. Malachi, W. Sawdon, and G. Chan. Recovery management in QuickSilver. *ACM Transactions on Computer Systems*, **6**(1):82–108 (February 1988).

[Howard88] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, **6**(1):51–81 (February 1988).

[Hurwitz89] J. S. Hurwitz. A tale of two operating systems: System V.4 and OSF 1. *UNIX in the Office*, **4**(2):1–11 (February 1989). Patricia Seybold's Office Computing Group, 148 State St., Suite 612, Boston, Ma 02109.

[Inmos89] Inmos T425 product announcement. *IEEE Computer*, **22**(7):106 (July 1989).

[Kristiansen89] E. H. Kristiansen, K. Alnæs, B. O. Bakka, and M. Jenssen. Scalable Coherent Interface. *Proceedings Eurobus* (London) (September 1989).

[Kure88] Øivind Kure. *Optimization of File Migration in Distributed Systems*. Ph.D. thesis, published as Report No UCB / CSD 88 / 413 (April 1988). Computer

Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley.

[Lazowska86] E. D. Lazowska, J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel. File access performance of distributed workstations. *ACM Transactions on Computer Systems*, **4**(3):238–68 (August 1986).

[Motorola89] Motorola Inc. Motorola takes the wraps off commodity-memory thrust. *Electronics*, pages 31+ (May 1989).

[Nelson88] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, **6**(1):134–54 (February 1988).

[Ousterhout85] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. *Proceedings of the 10th Symposium on Operating System Principles* (Orcas Island, Washington). Published as *Operating Systems Review*, **19**(5):15–24 (December 1985).

[Patterson88] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD* (Chicago, Illinois, June 1-3, 1988) (1988). ACM.

[Rosenbladt89] Peter Rosenbladt. Presentation to HP-UX kernel lab in Cupertino, Ca. (31 March 1989).

[Sager85] G. Sager and B. Lyon. Distributed file system strategies: the options and their implications. *UNIX Review*, **3**(5):28–31 33+ (May 1985).

[Satyanarayanan81] M. Satyanarayanan. A study of file sizes and functional lifetimes. *Proceedings of the 8th Symposium on Operating System Principles* (Asilomar, Ca). Published as *Operating Systems Review* **15**(5):96–108 (December 1981).

[Sandberg85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. *USENIX Association Summer Conference Proceedings 1985* (11-14 June 1985, Portland, OR), pages 119–30 (1985). USENIX Association, El Cerrito, CA.

[Shula89] Barbara Shula. Personal communications. (15 June and 14 July 1989).

[Srinivasan89] V. Srinivasan and J. C. Mogul. Spritely NFS: implementation and performance of cache-consistency protocols. Research Report 89 / 5 (March 1989). Digital Equipment Corporation Western Research Laboratory, Palo Alto, Ca.

[Stonebraker84] M. Stonebraker. Virtual memory transaction management. *Operating Systems Review*, **18**(2):8–16 (April 1984).

[Wilkes88] *Encapsulating type in storage systems.* Hewlett-Packard Laboratories technical report HPL–DSD–88–1 (May 1988).

[Wilkes89] *DataMesh — scope and objectives*. Hewlett-Packard Laboratories technical report HPL–DSD–89–37 (Revision 1, July 1989).