HEWLETT PACKARD

# Attribute-managed storage

*Richard Golding, Elizabeth Shriver, Tim Sullivan, and John Wilkes*

**Storage Systems Program, Computer Systems Laboratory**

**Hewlett-Packard Laboratories, Palo Alto, CA**

*Storage systems are continuing to grow, and they are become shared resources with the advent of I/O networks like FibreChannel. Managing these resources to meet performance and resiliency goals is becoming a significant challenge. We believe that completely automatic, attribute-managed storage is the way to address this issue. Our approach is based on declarative specifications of both application workloads and device characteristics. These are combined by a mapping engine to generate a load-assignment that provides optimal performance and meets availability guarantees, at minimum cost.*

## 1 The big picture

Current storage systems—file systems, databases, and backup and archival tools—all require significant administration. At the same time, the number of configuration and management choices is increasing because both the number of components and their configurability is increasing. For example, experience with RAID systems shows that few users have the expertise and the information to configure a disk array well, and that initial installation can be a time-consuming process. This is all made much worse if the system must adapt to changing workloads.

The increased load on, and sophistication required of, administrators invariably means that the systems being constructed are not optimally configured—either they cost too much or they do not provide adequate service.

To address these problems, we believe that storage systems must be *self-managing:* they must adapt themselves to the workload and resources they are given, without requiring human intervention. Our experience with the HP AutoRAID system [Wilkes95], which can dynamically select RAID layout policies based on usage, indicates that both performance and ease-of-use can be improved by doing so.

If a system is to be self-managing, it must be given a set of goals to manage towards. Our approach is to do this in two parts: our storage system is given a specification of the workload it has to support, the data it needs to store, and of the storage devices at its disposal. It then decides how many of each kind of storage device to use, and how to balance the load and data across them.

Both the workload behavior and the device capabilities are specified by describing *attributes* of the load or the device respectively. Thus, we refer to our approach as *attribute-managed storage*. We view it as an extension of system managed storage [Gelb89].

Workload attributes include performance requirements such as mean throughput, maximum latency, and jitter; resiliency needs such as availability, reliability, and fault models; cost bounds; data sizes, and so on. Device attributes are expressed similarly.

An attribute-managed storage system maps data to devices so that the requirements associated with each data item are met. When new resources are added or old ones fail, the system can detect this and reapportion storage accordingly.

We are investigating the construction of attribute-managed storage systems. As part of this we are working to answer several questions:

- How does one specify workload requirements?
- How does one specify device capabilities?
- What algorithms work well for automatically mapping storage objects to devices based on their attributes?
- How does the assignment of an object to a device translate into low-level resource reservation? What do these reservations mean at run time?

Position paper presented at the Workshop on Modeling and Specification of I/O (MSIO), San Antonio, Texas, 26 October 1995. The authors may be contacted at {golding, shriver, sullivan, wilkes}@hpl.hp.com.

We are initially limiting our focus to a subset of the problem by modelling only transaction processing, multimedia, and scientific computing applications; and by modelling only the capabilities of disk drives and tapes.

Attribute-managed storage is a part of our larger investigation into distributed storage systems. These systems consist of many smart, network-attached storage devices (NASDs) shared among many systems.

## 2 Model and principles

We are concentrating on the problem of assigning storage objects to appropriate devices using the model shown in Figure 1.

An attributed-managed storage system is concerned with reserving low-level resources to meet abstract application requirements. These reservations are maintained in two ways: persistent reservations are associated with stored data, while transient reservations are associated with the streams that access the data.

*Storage objects* are the basic persistent unit that applications access, and that must be assigned to storage devices. These objects could be files, tables or parts of tables in a database, recorded continuous media streams, or blocks of a scientific data set.

The objects carry sets of *attributes*. Some of these specify the Quality of Service (QoS) requirements that must be guaranteed to or are anticipated for applications that use the object. Other attributes indicate expected application behaviors that the system can use to make better layout decisions. The attributes are initially predeclared; later, the system may infer them from observed usage patterns.

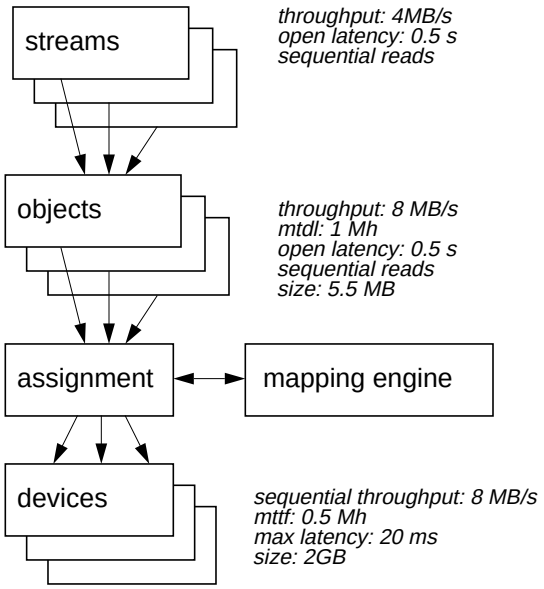


**Figure 1**: components in the model.

Separately, applications access storage objects through *streams*. A stream represents the application workload on the object and the resources that workload uses. The stream carries its own set of requirements and behaviors.

The objects are stored on *devices*. These devices have a set of low-level resources, which are modelled by *capability* attributes. Device resources are reserved for streams and objects. The system will only allow the creation of an object or stream if sufficient resources are available to meet its requirements.

The system maintains *assignment* metadata for the mapping of objects to devices. Algorithms for automatically finding good mappings is part of our research. It's important to note that unlike existing distributed file systems, an object is not constrained to a particular device by being part of a volume. Instead, each object can be assigned any appropriate device in the system.

### 2.1 Attributes and reservations

We describe objects, streams, and devices by attributes. Attributes allow us to specify abstractly the requirements and behaviors of streams and objects, and the capabilities of devices. It is important that the attributes themselves be intelligible to a human user.

Attributes are the only things we use to map objects to devices and to manage streams. Within a device, the assignment of an object to a device uses up concrete resources, but this set of resources is determined by finding the minimum set that will meet the abstract, attribute-level performance requirements. This limitation is important for keeping the system modular and extensible.

A few kinds of attributes are involved in mapping objects or streams with devices (Figure 2). An object provides a set of *requirements*, indicating its needs, and *behaviors*, indicating how it will be used. A device similarly provides *capabilities*, which model the low-level resources that can be allocated to an object, such as throughput and capacity. When an object is assigned to a device, the system negotiates between the object's requirements and the device's capabilities to form a *contract*. This contract is part of the assignment, and the system guarantees only hold when applications use objects according to the behaviors recorded in the contract.

Some attributes represent *behaviors* that applications will follow. The mapping system can use these behaviors to make a better selection of devices. For example, if an object is guaranteed to mostly be read sequentially, then it should probably be organized sequentially on disk.

Device capabilities measure the resources available to an object or stream. For example, sequential throughput (in

bytes / sec) and capacity (in bytes) both are consumed as objects with throughput and size requirements are stored on the device. Other capabilities, such as mean time-to-failure, represent device characteristics other than consumable resources.

Some attributes are measured as single numbers, while others require more complex specification. An application might indicate that it needs a throughput of 4 MB / s, and that the jitter on this value must be no more than 2%; or a disk might indicate that 90% of all requests will be satisfied within 20 ms. Yet more complex attributes might account for the effect of failures on these values [Wilkes91].

Other systems have used QoS attributes to represent requirements. For example the LLNL High Performance Storage System [Louis95] attaches a data structure containing both usage hints and requirements to stored files and to storage classes.

### 2.2 Storage objects and streams

Storage objects represent the persistent, static allocation of data, while streams represent the transient, dynamic use of the data. Both are used to reserve device resources.

Storage objects carry persistent resource reservations. These reservations are derived from the attributes that applications place on the object. The mapping engine uses these attributes to pick the devices on which the object should be stored, and to determine what fraction of the device's resources should be allocated to the object. For example, some video stream file might have a particular playout rate, and at any time at least three streams should be able to access that stream. Enough resource would need to be allocated to the object so 3x the playout rate was available, among other things.

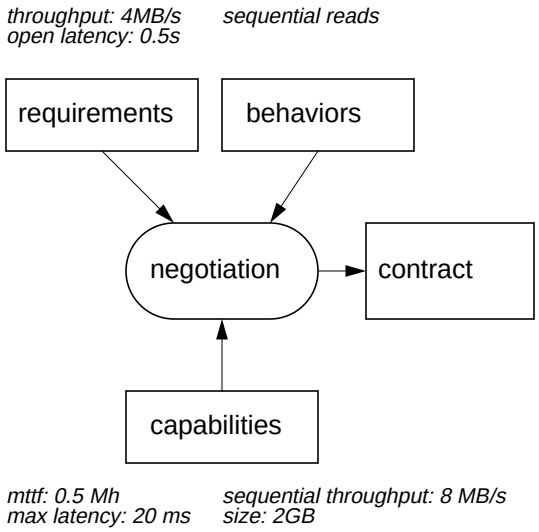


**Figure 2**: how attributes are used.

Similarly, a vitally important data file would carry a long mean-time-to-data-loss attribute.

A stream, on the other hand, has short-term, session-based reservations. When a stream is created that uses an object, it consumes some of the resources reserved to the object. In the video file example, a third of the throughput resources reserved for the file would be allocated to the a new stream.

The system practices admission control on both objects and streams. It will reject the creation of a new object if there isn't enough resource to allocate—like space or throughput. New streams can opportunistically allocate unreserved resources when possible: if there were three streams already using the video stream in our previous example, a fourth stream might be admitted if the system had enough unreserved resources to support one more stream.

Sometimes one wants to make reservations for a group of objects, so that applications are guaranteed access only to some subset of the objects. For example, a video file service might guarantee that up to fifty video streams could be active concurrently to some subset of the hundreds of video files it stored. Such a reservation would require far fewer device resources than requiring that every file be accessible by up to fifty streams.

Some applications are flexible, and can cope with a range of different performance levels. These attributes include the desired performance level, the minimum acceptable level, and a *goodness* function indicating the value of supplying resources above the minimum. For example, if an application can function properly with a stream having 512 kB / s throughput, but any additional throughput up to 1 MB / s would be better, the stream's throughput attribute would carry a goodness function that increased linearly from 0 to 1 as the throughput ranged from 512 kB / s to 1 MB / s.

When a stream is created the system may also initiate the execution of a prefetch policy to migrate a copy of the object onto faster media. Expressing complex prefetching and inter-level migration is as yet not well understood.

### 2.3 Devices

Devices have a set of capabilities that abstract the resources they can provide to objects and streams. These capabilities are used when mapping objects to devices. The underlying resources are reserved to objects or streams, and at run time a stream can only use up to its reserved resource.

Devices also have costs. The cost includes both the one-time cost of purchasing the device, and the ongoing costs of providing power, cooling, and maintenance. The mapping algorithms we use to assign objects to devices attempt to minimize the total cost of the devices.

Many devices are simple and obvious, like disks, tapes, and cache memories. Other important devices include

the interconnection network, device controllers, and power supplies.

We also model *virtual devices*. A virtual device modifies the capabilities of a physical device, usually to optimize for a particular stream behavior. For example, a device driver can improve a disk's performance by scheduling concurrent requests efficiently. The capability attributes of a virtual device derive from the capabilities of the device or devices it uses and from the way it modifies those devices.

Some virtual devices are parameterizable, and the mapping engine must select parameters that match the objects being assigned to the virtual device. For example, a generic disk striping device can vary the stripe depth and width to match application throughput requirements.

## 3 The mapping problem

We are considering four storage management problems:

1. planning the device capacity required to support a set of objects or streams;
2. planning the set of streams or objects that can best be supported by a set of devices;
3. batch reorganization of the objects in a system; and
4. incrementally adding new objects to a system, or changing an object's requirements;

These are cases of what we call the *mapping problem:* to allocate device resources so that

- object requirements are met;
- the device cost is a low as possible; and
- as many of the objects' optional requests are met as possible. (That is, the goodness values of the optional attributes is as high as possible.)

Solutions often trade cost for goodness: it may take extra devices to satisfy the optional requests, such as bandwidth above some minimum. A system administrator can specify the trade-off in the abstract—perhaps how much additional device cost is acceptable—or can perform what-if analyses to find the right trade-off.

### 3.1 Mapping engines

We are beginning with the batch planning problems, and deferring the problem of incremental assignment until we have more intuition about the mapping problem.

A number of mapping engines based on traditional approximate resource allocation algorithms appear promising. In particular, we are looking at the multi-constraint knapsack problem, simulated annealing, and genetic algorithms.

We are conducting a series of experiments to compare different mapping engines. We are assembling a number of test cases, each with a small population of objects and devices. Each engine solves each of the test case, and we compare the resulting assignment according to:

- speed of execution;
- the device cost of the solution;
- the goodness of the assignment;
- the distance from optimal for both cost and goodness; and
- the fraction of test cases for which an assignment was found.

## 4 Conclusions

Distributed storage systems are coming to include large numbers of configurable, shared resources as I/O networks and directly-attached storage devices become available. This is causing the number and complexity of management options to grow to the point that manual administration is no longer feasible.

We are working on self-managing storage systems based on attribute-managed storage. Our approach uses declarative, abstract specifications of both application requirements and device capabilities to guide automatic management mechanisms. The system includes a mapping engine that automatically maps objects to devices, ensuring that performance requirements are met and minimizing cost. This enables transparent object migration for fault tolerance, performance enhancement, or hierarchical storage management.

## References


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

[Louis95] S. Louis and D. Teaff. Class of service in high performance storage system. In *the Proceedings of 3rd International IFIP TC6 Conference on Open Distributing Processing (ICODP '95),* pp. 307–18, February 1995.

[Wilkes91] John Wilkes and Raymie Stata. Specifying data availability in multi-device file systems. Position paper for *the 4th ACM-SIGOPS European Workshop,* Bologna, September 1990. Also published as Operating Systems Review **25**(1):56-9, January 1991.

[Wilkes95] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system technology. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles,* Copper Mountain Resort, CO, December 1995. Also published as Operating Systems Review **29**(4).