A Formalization of the Attribute Mapping Problem

Elizabeth Shriver
Computer Systems and Technologies Laboratory
HP Laboratories Palo Alto
HPL-1999-127
October, 1999*

attribute-based storage management, formalization

An *attribute-managed solver* makes assignments of workload units to devices based on attributes specifications that describe the workload units and the devices. This paper presents the sets of attributes that model workloads and devices. It also discusses the necessary formalization of an attribute-managed solver in terms of objective functions and constraint expressions. These formalizations enable progress toward an implementation of an attributed-managed storage system.

Internal Accession Date Only
* Originally written in July 15, 1996

Contents

1 Introduction

An introduction to self-configuring self-managing storage systems and attribute-managed storage can be found in [Golding95]. An understanding of this paper depends on an understanding of that paper.

We begin with an overview on how attributes are specified in Section 2. We follow this by descriptions of the models for workloads and devices that are needed for our approach in Sections 3 and 4. We overview the mapping problem in Section 5 and formalize it in terms of objective functions and constraint expressions in Section 6. We present a brief discussion of previous work, future work, and our conclusions in Sections 7, 8, and 9.

This version of this paper (Rev E) has been updated from the previous version (Rev D) in a number of important ways:

- The workload use patterns of burst_request_rate and burst_count were added to Section 3.1.1.
- Several typos were fixed and additional explanations were added.

There are some concepts in this paper that still have to have additional details added. A partial list of what will have additional details or has additional details discussed in another paper follows:

- availability and reliability
- streams and objects
- capacity as a distribution
- system goals

There are also concepts that are presented that are not used yet; a partial list of these follows:

- workload use patterns such as run length and run stride (These will be used in determining the service time for complex devices.)
- zones on the devices

2 Attributes

Our basic method of describing workloads and devices is through *attributes*. This section discusses what an attribute is, how we use them, and how are they specified. In attribute-managed storage, attributes are the only things we use to map workload units to devices.

Attributes allow us to specify abstractly how the workload and devices behave and what are the performance needs of the workload. The workload attributes are represented

in terms of *requirements* and *use patterns* and the device attributes are represented in terms of *capabilities*.

Some attributes are measured as single numbers, while others require more complex specifications for a more accurate representation. The complex specifications are not discussed here; at this point, we assume attributes are specified as single numbers.[1]

A *workload* (respectively, *device*) *specification* is one or more workload units (devices), with values assigned to the attributes. Attribute values could either be specified (e.g., by a system administrator for workload attributes and device manufacturer for device attributes) or they could be derived from values that have been specified. As an example, reliability of a device can be derived from the availability of the device.

Unspecified attributes that cannot be derived can be left blank in a specification. This implies that the value of the attribute is not important.

3 Workload specification

We use two items to model a workload—*storage objects* and *data streams*. *Storage objects*, which we refer to as *objects*, are the basic persistent unit that applications access, and that must be assigned to storage devices. These objects could be files, tables or part of tables in a database, recorded continuous media streams, or blocks of a scientific data set.

Separately, applications access storage objects through *data streams*, which we call *streams*. A stream represents a group of data requests for the object.

The interaction between streams and objects has not been formalized, and is therefore not discussed here. We present this formalization using the term *workload units*, which can be thought of as a file that needs to be stored on disk with a single open access path. A workload unit comprises many requests. A device's workload is zero or more workload units accessing data on the device.

The relative importance of workload units is given by the importance attribute, which supports a ranking of workload units. This attribute represents the importance of the workload units in the "grand scheme of things." Importance is used to determine which workload units' requirements are the most important if all of the workload units' needs cannot be met.

Similarly, the relative importance of workload unit attributes is given by a weight parameter, w_{at}, for attribute at; the weight parameter allows a ranking of the importance of the individual attributes. For example, a high weight value for long-term transfer rate means long-term transfer rate is important for that workload unit.

Grouping together similar workload units into *workload unit classes* can reduce the size of the search space when assigned workload units to devices. It also makes the process of specifying workload units easier. Workload units having the same specification can be in the same workload unit class.

[1] **Simplification:** Attributes are specified using only single values (i.e., no distributions).

Table 1: The requirement attributes for workload units. A value of probability in the fourth column in the table means the value is between 0 and 1. B refers to bytes and s refers to seconds.

attribute	symbol	description	unit
long-term transfer rate	long_term_transfer_rate	the amount of data that the workload unit needs generated or consumed by the device per unit time over the lifetime of the workload unit	B/s
last-byte latency	last_byte_latency	the per-request latency for the last byte to be received that can be tolerated by the workload unit	s
data capacity	capacity	the amount of data storage that the workload unit needs on the device	B
availability	availability	the fraction of time that the device is servicing requests that is needed by the workload unit	probability
reliability	reliability	the probability needed by the workload unit that the device will be servicing requests continuously from time 0 to time t	probability (as a function of t)
correctness violation	correct_viol	the fraction of data that the workload unit can tolerate dropped or incorrectly transfered	probability

3.1 Our workload unit model

3.1.1 The attributes

We develop a workload unit model that captures the storage device needs of the application and the behavior of the application. We divide the characteristics or attributes of a workload unit that we are interested in into two groups: *requirements* and *use patterns*.

Requirement attributes Requirements define what the supporting device needs to deliver. Long-term transfer rate and capacity are sample requirement attributes. The requirement attributes of a workload unit are described in Table 1. (One of the attributes in the table is last-byte latency; this is called *response time* by some.[2]) Since the availability

[2]We define the *last-byte latency jitter* to be the variance in the last-byte latency. The last-byte latency jitter can be modeled when the last-byte latency is modeled as a distribution.

Table 2: The requirement attributes derivations for workload units.

attribute	symbol	derived from/derived as
long-term transfer rate	long_term_transfer_rate	request_rate * request_size
last-byte latency	last_byte_latency	buffer size (for some applications)
reliability	reliability	availability

and reliability can be specified independently as requirements, the effect of the device going down is not factored into the long-term transfer rate and the last byte latency.

There are four possible transfer rates that can be specified:

- *burst transfer rate*: the transfer rate that the workload unit is receiving data at once the device starts to transfer data,
- *request transfer rate*: the transfer rate computed over the interval from request issuance to request completion, and
- *interval transfer rate*: the transfer rate computed over an interval of specified length, and
- *long-term transfer rate*: the transfer rate computed over the lifetime of the workload unit.

If we only model the storage system, we cannot accurately determine the burst transfer rate that the device will be supplying because of the large impact of the communication network on this measure of transfer rate; therefore, we do not have burst transfer rate as one of our requirement attributes. The request transfer rate is equal to the request size divided by the last-byte latency, thus either the request transfer rate or the last-byte latency can be used as an attribute. We have chosen the term last-byte latency since it is the more-commonly used measure. We currently do not have a need for the interval transfer rate requirement.

Some of the requirements can be derived from other requirements or use patterns; the requirement attribute derivations of a workload unit are shown in Table 2. (Some of the attributes in the "derived from/derived as" column in Table 2 are defined in Table 3.) Having the system derive requirements allows the workload unit to be specified in terms of attributes that are important for that workload unit. For example, it might be more natural for a transaction processing application to be specified in terms of the requests per minute and the mean request size than in terms of the long-term transfer rate.

Use pattern attributes Use patterns describe how the workload unit will behave once it is running on a storage system. They are also referred to as the *workload behaviors*. The use patterns of a workload unit are described in Table 3 and Table 4

Table 3: The use pattern attributes for workload units.

attribute	symbol	description	unit
request rate	request_rate	the rate that individual requests will be made from the workload unit to the device	requests/s
bursty request rate	burst_request_rate	the request rate during a burst that individual requests will be made from the workload unit to the device	requests/s
bursty interval length	burst_count	the amount of time that a bursty interval lasts	s
request size	request_size	the amount of data that is requested at once	B
access type	access_type	read or write	
read fraction and write fraction	read_fraction, write_fraction	the fractions of the requests that are reads and writes over the lifetime of the workload unit	probability

We have three use pattern attributes that capture *time access patterns—request rate*, *bursty request rate*, and *bursty interval length*. The request rate is the rate that the requests are made from the workload unit, averaged over the lifetime of the workload unit or some other suitably long interval. The bursty request rate is the rate during a burst, averaged over the bursty interval length. A burst is defined as a group of contiguous requests, where each pair of sequential requests occurs in less than the *burst inter-request gap*. The value of the burst inter-request gap is computed from the histogram of the inter-request gaps of the workload unit, where the gaps are sorted by size. A workload unit that has a bursty arrival process where the burst size and frequency can be modeled by exponential functions will have an inter-request gap; in the first figure in Figure 1, the inter-request gap should be a value between x_1 and x_2.

The *access type* attribute is set to either "read" or "write" if the workload unit performs only reads or only writes. If a workload unit performs both reads and writes, the read and write fractions (read_fraction and write_fraction) tell us the fraction of requests that are reads and writes. (By definition, read_fraction $= 1 -$ write_fraction.)[3]

Run length and *run stride* measures of the *spatial data access patterns*. These attributes are similar to what [Patterson93] suggests for applications to use as hints to the operating

[3]**Simplification:** We assume in Section 6 that a workload unit can only have one access type, i.e., a workload unit can either read or write.

Table 4: The use pattern attributes for workload units, continued.

attribute	symbol	description	unit
run length	run_length	the number of sequential bytes accessed by consecutive requests	B
run stride	run_stride	the number of bytes between the beginning bytes of two consecutive runs	B
locality fraction	locality_frac	the fraction of requests that are in a run	probability
number of requests read forwards	num_requests_read_forwards	the number of requests that are spatially locality (defined as the data wanted by two requests to be within x bytes) read forwards within y requests	requests
number of requests read backwards and forwards	num_requests_read_forwards_and_backwards	the number of requests that are spatially locality (defined as the data wanted by two requests to be within x bytes) read backwards and forwards within y requests	requests
forwards fraction	frac_requests_read_forwards	the fraction of requests that are in a spatial locality interval	probability
forwards and backwards fraction	frac_requests_read_forwards_and_backwards	the fraction of requests that are in a spatial locality interval	probability

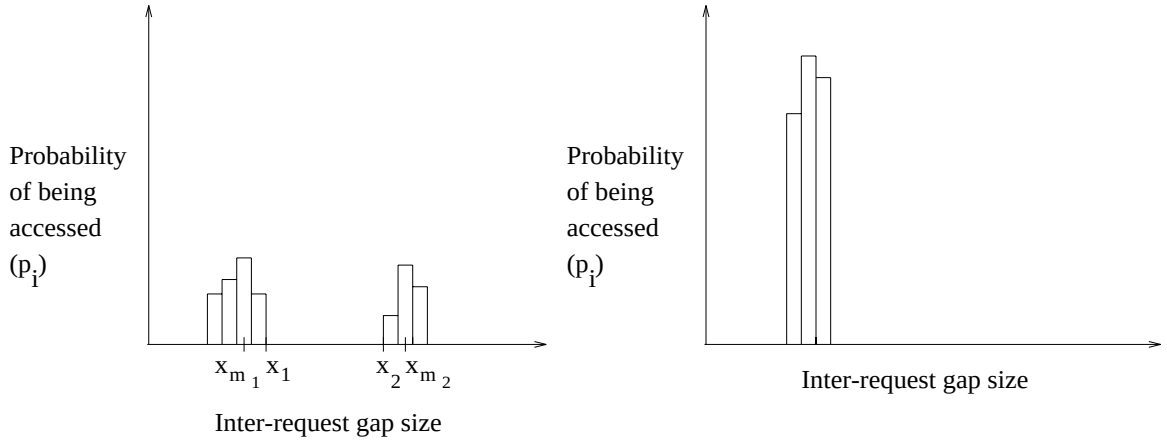

Figure 1: Sample histograms of the inter-request gaps of two workload units. The one on the left represents a workload unit having requests gaps of either x_{m_1} or x_{m_2}. The one on the right represents a workload unit that has a close-to-constant inter-request gap, and therefore is not bursty.

system for file system caching. A *run* is a group of bytes that are accessed consecutively across requests. For example, for a video-on-demand application, a run is the entire video clip. Run length and run stride, as described in Table 4, define whether the workload unit patterns are sequential, consecutive, or random. If the pattern is sequential, run length and run stride specify the strided or regular pattern [Nieuwejaar95] if there is one. We use Nieuwejaar's terminology for sequential and consecutive; a *sequential request* is one where the byte offset being accessed is at a higher file offset than the previous request, and a *consecutive request* is a sequential request that begins where the previous request ended. The locality fraction parameter, locality_frac, is used to capture the case when only a fraction of the requests are in a run.

We also define a *database random pattern* that models database accesses where the database record is larger then the request size. In this pattern, a random record is read/written in request_size-chunks. Examples of run length and run stride for the various types of patterns are:

- random: run_length = request_size, run_stride = 0

  A zero value for run_stride means the value is undefined.

- consecutive: run_length = the amount of data to be accessed in total, run_stride = 0

- sequential: for some positive integers, x and y, such that $y >$ request_size $\cdot\, x$

  run_length = request_size $\cdot\, x$, run_stride = y

- database random: for some positive integer, x,

  run_length = request_size $\cdot\, x$, run_stride = 0

Some workload units may have spatial and temporal locality that can not be described in terms of run length, i.e., the workload may have non-consecutive, yet still spatially close, accesses. We capture this using the num_requests_read_forwards and num_requests_read_forwards_and_backwards measures (along with the frac_requests_read_forwards and frac_requests_read_forwards_and_backwards). Informally, these measures define *locality intervals* where the num_requests_read_forwards is the number of local requests in the interval if only the forward direction of the requests are examined. The num_requests_read_forwards_and_backwards is the number of local requests in the interval if the forward and backwards direction of the requests are examined. Another way to look at num_requests_read_forwards_and_backwards is it represents the number of requests that will be read when the $x/2$ byte is read in a group of x bytes by a workload unit. The frac_requests_read_forwards and frac_requests_read_forwards_and_backwards fractions determine the fraction of the total requests that have spatial locality.

Not stated in the tables are the *averaging intervals*, which is the length of time that the values are determined over. As currently specified, all of the use patterns need an averaging interval.

If an attribute is not specified, then the value of the attribute is not important or it can be derived.

3.1.2 Utility

When the requirements of a workload unit are being met, there are some attributes that can be supported by any value in a range of values. The workload unit requirements are not always hard requirements, i.e., if the workload unit is given less than its optimal value, it still might be able to function. We have chosen to model this as *utility*.

Jensen [Jensen91] discusses real-time completion constraints of tasks being serviced by an operating system in terms of *hard* and *soft*. A hard constraint signifies that a result has zero or negative utility if produced after a certain time and a soft constraint means the result has a corresponding utility which is a function of the time at which it is completed. Campbell *et al.* [Campbell96] discuss three levels of service (*deterministic*, *predictive*, and *best effort*) which are ways to guarantee performance as hard, soft, and firm.

We use these ideas to determine how the values specified for the workload unit requirements have to be meet. That is, each workload unit requirement attribute has a corresponding *utility function*, which gives the utility of the various possible values of the attribute. The utility function identifies the utility of values that the attribute can take.[4] Figure 2 shows what the utility function for long_term_transfer_rate can look like if a workload unit needs at least 4 MB/s of bandwidth. Figures 3 and 4 show possible utility functions for last-byte latency.

With the current set of attributes defined for workload units, we believe that the utility curves will always be as simple as the examples presented in Figures 2, 3, and 4.

[4] **Simplification:** This discussion assumes that the requirement attributes are given as single values. This becomes much more complicated when taking into account distributions.

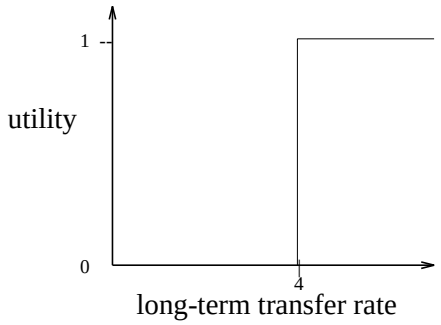

Figure 2: The long-term transfer rate utility graph for a workload unit that needs at least 4MB/s of bandwidth. (The capacity utility graph has the same shape as this curve.)

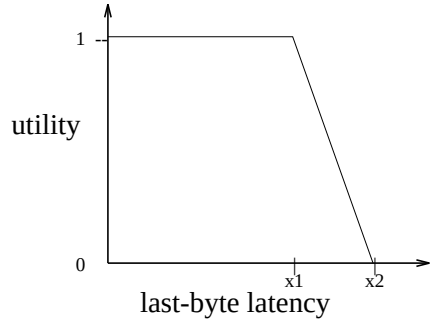

Figure 3: The last-byte latency utility graph for a workload unit that has very high utility for values of last byte latencies of less than x_1, but can tolerate values of last byte latency (with decreasing utility) to x_2. (The availability, reliability, and correctness violations utility graphs can all have the same shape as this curve.)

The utility function could be used to also represent the weight of the attributes discussed in Section 3. The decision of whether utility and the weights be kept as separate values for user input should be made based on the ease of user specification.

If a requirement attribute is not specified, the system can assume that the utility curve is always 1.

3.2 Open issues for the workload model

The following are the current open issues for the workload model:

- It has been thought that the correctness violation attribute might be used to capture the data lost if the data is stored on the device using lossy compression. How should this be formalized?
- *Spatial locality* is the tendency of the workload to access data that is physically close

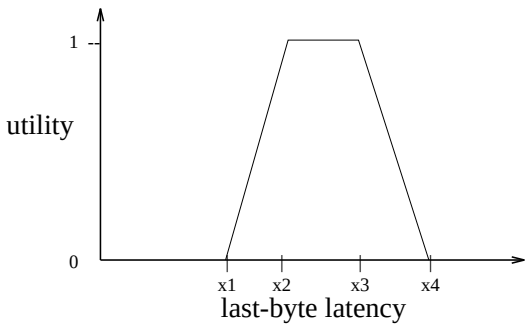

Figure 4: A possible last-byte latency utility graph for a workload unit that determines the last-byte latency based on the size of the buffers.

to other accessed data. The greater the spatial locality, the greater the gain in cache hits as the cache line size is increased. We currently measure the spatial locality by run length, which only has a value if the access pattern is sequential or database random. Should another use pattern attribute be added?

- One application can have many different *phases*, where the phases run sequentially and have possibly different specifications. How can this be modeled? (Phases are seen as long-term behaviour.)

  One solution is to model different phases as different workload units and to create a worst-case from these different phases where, if the requirements of the worst-case workload unit were met, the requirements for each one of the phases would also be met. This solution could possibility assign much too much resources to the application.

  Another solution is to model different phases as different workload units and to assign the "composite workload unit" to a device only when the requirements of each of the phases can be supported. This solution seems to increase the number of constraint equations that have to be checked when making an assignment.

- Workload unit requirements can vary over short periods of time (i.e., bursts). How should this be modeled? ([Low93] might have some ideas.)

- We have looked at the following question: how is a mapping done so that the needs of the Y workload units are met with a probability of 90% when X workload unit specifications are given (where $Y < X$)?

  One solution to this is based on ordering the combinations of workload units. To be able to order the combinations, we must be able to able to order the workload units. Thus, we must define a relation between workload units and devices that is a total order. The logical choice for a relation is the binary relation, G, on the set of workload units such that aGb if any device meets the requirements of b, it will meet

the requirements of a. But, in reality, this is only a partial order. Does there exist another relation which is total order and has the flavor of the above relation?

- Would defining a grouping of "similar workload units" be useful? Currently, we have a *workload unit class* which groups together workload units that have the same specification. Can we group together workload units that have similar specifications? If so, how do we define similar? What would this be used for?
- What should the system administrator be specifying? Garth Gibson has expressed some concern that the attributes that we have specified are not values that a system administrator can specify.
- How do streams and objects differ? How can they be used to model an application? The different needs of streams and objects might be able to be captured by *reservations* and *pre-reservations* (or *reservations in advance* [Wolf95]). [Degermark95, Plagemann96, Sreenan96] might be useful to read.

4 Device specification

4.1 Our device model

Initially, our view of a device is very simple; it consists of device mechanisms that service only one request at a time. A request is specified in terms of an amount of data (i.e., the request size) and the location of the data. There is no caching, queues with interesting scheduling algorithms, groups of devices working as one, etc. (These will be addressed by the modeling of *complex devices*.) The time to service a request is the time to position the head (Position_Time) plus the time to transfer the data. Multiple "transfer regions" can be modeled with *zones*.

Devices are specified in terms of *capabilities*, *behaviors*, and cost (Cost). The capabilities of a device describe the device attributes that are not workload dependent and the device behaviors describe the device attributes that are workload dependent. The capacity is an example capability and last-byte latency is an example behavior.

The device capabilities are shown in Table 5. The device behaviors are shown in Table 6. A number of these attributes are derived from attributes that are specified; the derived capabilities of a device are shown in Table 7. In particular, we have [Gibson93]

$$\begin{aligned} \mathsf{Availability} &= \frac{\mathsf{MTTF}}{\mathsf{MTTF} + \mathsf{MTTR}} \\ \mathsf{MTTF} &= \int_0^\infty \mathsf{Reliability}_t \, dt. \end{aligned}$$

If we assume that the reliability function is a exponential we have [Siewiorek92]

$$\mathsf{Reliability}_t = e^{-\frac{t}{\mathsf{MTTF}}}.$$

Table 5: The capabilities for devices.

attribute	symbol	description	unit
transfer rate	Transfer_Rate	the nominal rate that the device can transfer data (i.e., the maximum transfer rate)	B/s
data capacity	Capacity	the amount of data storage the device can store	B
mean-time-to-failure	MTTF	the mean amount of time to failure of the device	hours
mean-time-to-repair	MTTR	the mean amount of time to repair of the device	hours
correctness violation	Correct_Viol	the fraction of data that the device may drop or incorrectly transfer	
availability	Availability	the fraction of time that the device is servicing requests	probability
reliability	Reliability	the probability that the device will be servicing requests continuously from time 0 to time t	probability (as a function of t)
cost	Cost	the dollar cost of the device	$

Availability has also been defined in terms of mean-time-between-failures and mean-time-to-failure [Gray86].

In addition to the attributes being divided into capabilities and behaviors, the device attributes have three flavors: *consumable*, *generatable*, and *non-affected*. If an attribute is consumable, there will be less of that attribute available once a workload unit is assigned to a device. Transfer rate and capacity are examples of consumable attributes. If there will be more of an attribute once another workload unit is assigned, we say that the attribute is generatable; an example is last-byte latency. If the value of the device attribute is not affected when a workload unit is assigned, the attribute is *non-affected*.[5] See Table 8 for the classifications of the device capabilities.

Zones

The transfer rate of a device depends on whether the device has zones. If the device has zones, the transfer rate depends on the zone, since the zone determines the amount of data stored in the track. Let

$$\mathsf{Bytes\_per\_Track}[i, \mathsf{zone}] = \text{the number of bytes on a track in the } \mathsf{zone}\text{th zone on device } i\text{; can be computed by}$$

[5] **Simplification:** In our first approximation of availability, we assume that it is a non-affected attribute.

Table 6: The behaviors for devices.

attribute	symbol	description	unit
long-term transfer rate	Long_Term_Transfer_Rate	the long-term rate that the workload is receiving transfered data; this takes into account the positioning time; is a function of the workload assigned to the device	B/s
last-byte latency	Last_Byte_Latency	the per-request latency for the last byte of the requested data to be received; is a function of the workload assigned to the device	s
positioning time	Position_Time	the amount of time that the device takes to position the head (based on the seek time and the rotational latency)	s

Table 7: Derived capability attributes for devices.

attribute	symbol	derived from
transfer rate	Transfer_Rate	rotation speed, number of platters, number of sectors per track in the zone, number of bytes per sector, zone layout
availability	Availability	mean-time-to-failure, mean-time-to-repair
reliability	Reliability	mean-time-to-failure

Table 8: The flavors of each capability attribute.

	consumable	generatable	non-affected
transfer rate	X		
last-byte latency		X	
data capacity	X		
availability			X
reliability			X
correctness violations			X

$$\begin{aligned}
\mathsf{Rotation_Time}[i, \mathsf{zone}, B] &= \frac{\mathsf{Sectors_per_Track}[i, \mathsf{zone}] \cdot \mathsf{Bytes_per_Sector}[i]}{\text{the amount of time that it takes to rotate device } i \text{ to pass } B \text{ bytes when the head on a cylinder in the } \mathsf{zone}\text{th zone. This is based on the revolution speed.}}
\end{aligned}$$

We also can compute average values for a zoned device:

$$\begin{aligned}
\mathsf{Ave_Bytes_per_Track}[i] &= \mathsf{Bytes_per_Track}[i, \mathsf{zone}] \text{ averaged over all zones} \\
\mathsf{Ave_Rot_Time}[i, B] &= \mathsf{Rotation_Time}[i, \mathsf{zone}, B] \text{ averaged over all zones.}
\end{aligned}$$

This allows us to also define the following which we will use as shorthand notation:

$$\begin{aligned}
\mathsf{Bytes_per_Cylinder}[i, \mathsf{zone}] &= \text{the number of bytes on a cylinder in the } \mathsf{zone}\text{th zone on device } i \\
\mathsf{Ave_Bytes_per_Cylinder}[i] &= \mathsf{Bytes_per_Cylinder}[i, \mathsf{zone}] \text{ averaged over all zones}
\end{aligned}$$

If the transfer rate of the device is not given by the device vendor or if the device has zones, we can compute the transfer rate of data as follows:

$$\mathsf{Transfer\_Rate}[i, \mathsf{zone}] = \frac{\mathsf{Bytes\_per\_Track}[i, \mathsf{zone}]}{\mathsf{Rotation\_Time}[i, \mathsf{zone}, \mathsf{Bytes\_per\_Track}[i, \mathsf{zone}]]}.$$

We can approximate the transfer rate a number of different ways:

$$\begin{aligned}
\mathsf{Average_Transfer_Rate}[i] &= \frac{\sum_{\mathsf{zone}=1}^{\mathsf{Num_of_Zones}[i]} \mathsf{Transfer_Rate}[i, \mathsf{zone}]}{\mathsf{Num_of_Zones}[i]} \\
\mathsf{Min_Transfer_Rate}[i] &= \min_{1 \le \mathsf{zone} \le \mathsf{Num_of_Zones}[i]} \mathsf{Transfer_Rate}[i, \mathsf{zone}] \\
\mathsf{Max_Transfer_Rate}[i] &= \max_{1 \le \mathsf{zone} \le \mathsf{Num_of_Zones}[i]} \mathsf{Transfer_Rate}[i, \mathsf{zone}].
\end{aligned}$$

If an equation does not specify which approximation should be used, assume $\mathsf{Average\_Transfer\_Rate}[i]$.

4.2 Open issues for the device model

There are a number of open issues in defining the device model:

- John thinks that correctness violations are not needed for devices, but are needed for network delay. Can the $\mathsf{Correct\_Viol}[i]$ be used to model the network delay or data loss due to the network?

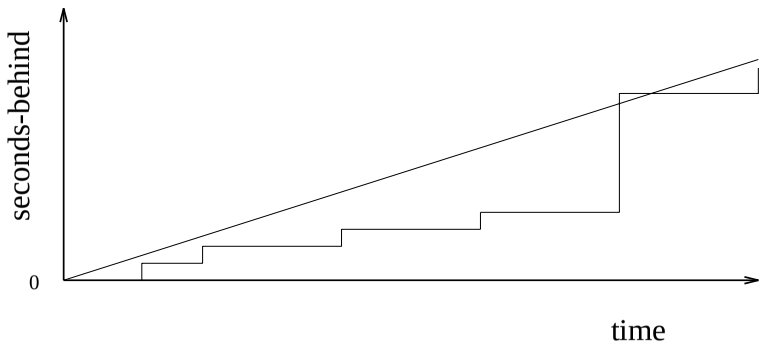

Figure 5: A graph which shows how many seconds a workload unit can get behind for a particular device. Each step coincides with an event. The line is an approximation of the graph.

- If an application is generating requests 24 hours/day, it is not clear when to back up the data on the devices. Also, different applications (i.e., workload units) have different data back up requirements. In this case, one of the reasonable ways to back up the data is, every X requests, a backup read is sent out of the device to the backup device. This affects the performance of the device and should be taken into account when computing the transfer rate and the last-byte latency.

- There are glitches in the transfer rate and last-byte latency from events such as thermal recalibrations, slip sector sparing, seek misses, and read errors. This might be able to be modeled by *slip rate*, the fraction of time lost due to the device processing these types of events. As an example, the value of slip rate can be computed by summing all of the values for the various events that cause performance variations such as thermal recalibrations.[6]

  To compute an approximation of the slip rate of a device, we need to sum the durations due to various events over the longest event interval. The cumulative effect of events can be seen clearly in a graph that plots the amount of time a workload unit request could get behind as a function of time; a sample graph is shown in Figure 5. The value of slip rate line that best fits the graph:

$$\mathsf{Slip\_Rate}[i] \approx \text{slope}. \tag{1}$$

- How do we model complex devices?

[6]In the case of event EV, we get the following if we were to determine the expected values to sum:

$$\mathbf{E}(\text{delay from } EV \text{ in interval of length } t) \quad \approx \quad t\mathsf{freq}_{EV}\mathsf{duration}_{EV}$$

when $\mathsf{duration}_{EV} \ll t$.

- There exist devices that are a slow tape or optical jukebox fronted by a cache disk. Can these be modeled as complex devices? (The access rate depends strongly on the working set: if it fits/hits on disk, looks like a disk. Otherwise somewhere between disk and tape jukebox.)
- [Louis95] has purge and migration policies as attributes where purge policies determine when and what data is purged and migration policies determine how and when the data is moved. Should we have these as part of complex devices?

5 The mapping problem

Now that we have a model for the workload and devices, we need an *assignment* from a set of workloads to a set of devices. The purpose of the assignment is to associate a group of workload units to a device, where the device will be able to service the I/O requests coming from the workload units. An *assignment* is the list of the devices that should be able to meet the needs of the workload. It also includes the mapping of workload units to devices, along with the amount of each device attribute assigned to each workload unit. We sometimes refer to just the assignment of one workload unit as *workload unit assignment.*[7]

5.1 Input/output into the mapping problem

The mapping problem, as stated, is not well-defined. Do all of the workload units have to be supported? Do all of the devices have to be used? These questions can be asked best by determining what the groups of workload units and devices identify. Think of a workload unit class having one of the following "tags":

- "Support all of the workload units in me."
- "Support as many as x of the workload units in me."
- "Support some of the workload units in me (with no upper bound)."

Similarly, a device can be tagged with one of the following:

- "Use me; I'm already bought."
- "You can use me; I'm easy to get."
- "You can use me, but I might have to be ordered."

[7] **Simplification:** One or more workload units can be assigned to one device. That is, there are no partial workload units assignments and workload units cannot span devices.

To support these different ideas, we have defined the following terms. A *set* is a fixed number of items where all are meant to be used or supported. A *pool* is a fixed number of items where some are meant to be used or supported. A *sea* is a variable number of items. It can be thought of conveniently for devices as a pool plus a manufacturing plant that can deliver devices on request. (We have *dynamic* versions of each of the above terms that allow us to define a changing group of workload units or devices. For example, a dynamic workload unit set represents an evolving set of workload units.)

With the groupings of sets, pools, and seas, we can ask a number of different questions:

- set of workload units/set of devices: Can the workload units' needs be met by this set of devices? The answer is yes/no.
- set of workload units/pool of devices: What subset of devices from the devices that I own do I need to use to meet the workload units' needs?
- set of workload units/sea of devices: What devices do I need to buy to meet the needs of these workload units?
- pool of workload units/set of devices: How can I use my spare device capacity that is currently on my machine, i.e., what extra workload units can I run? Help—I just lost half of my machine—what set of workload units can I run?
- pool of workload units/pool of devices: How can I use my spare capacity? (This question makes the most sense if being asked for a number of different workload units/device configurations so that the best choice can be picked.)
- sea of workload units/set (pool) of devices: How much of my workload can I do with these devices that I own?
- sea of workload units/sea of devices: What is the best cost performance I can get from these devices with these types of workload units?

These questions can be seen in Figure 6.

An assignment is determined by a solver, which takes as input: the workload specification, device specifications, and goals which identify the intent of the solver. A sample goal is "cost is more important than performance." The output will be the assignment of workload units to devices, and the amount of resources assigned to each workload unit from the device that it is assigned to.

6 Formalized mapping problem

We formalize our mapping problem by specifying constraint equations/expressions that need to be meet and objective functions that need to be maximized. We use constraint

Number of …		Devices: set	Devices: pool	Devices: sea
Work-load units	set	Can the needs be met?	How many of these devices do I need?	What do I need to buy to meet these needs?
	pool	Help! I just lost half of my machine--what set of streams can I run?	How can I use my spare capacity?	
	sea	How much work can I do with these devices?	How much work can I do with what I own?	What is the best cost performance I can get?

Figure 6: Sample questions that can be asked with different types of input.

equations/expressions and objective functions since that is how the multi-constraint knapsack problem is presented and we feel that approximation algorithms developed for the multi-constraint knapsack problem can be used as the base algorithm for our solver. The following sections present the constraint equations/expressions and objective functions for each attribute. The discussion refers to workload units; the expressions should apply to both objects and streams.

6.1 Basic definitions

We have the following definitions:

$$\begin{aligned} S &= \text{number of workload unit classes} \\ \mathsf{size}[j] &= \text{number of workload units in the } j\text{th unit class} \\ D &= \text{number of devices} \end{aligned}$$

The output of the mapping problem will be the values $x[i,j]$, $d[i]$,[8] and $\mathsf{quantity}_{\mathsf{at}}[i,j]$ where

$$\begin{aligned} x[i,j] &= \text{number of workload units from the } j\text{th workload unit class} \\ &\quad \text{assigned to } i\text{th device} \end{aligned}$$

[8] If the input for the workload and devices is set/set, then $d[i] = 1$ for all i.

$$
\begin{aligned}
d[i] &= \text{1 if } i\text{th device is used by some workload unit} \\
&\quad\ \text{0 otherwise} \\
\mathsf{quantity}_{\mathsf{at}}[i,j] &= \text{the amount of attribute } \mathsf{at} \text{ that a workload unit} \\
&\quad\ \text{in workload unit class } j \text{ is assigned on device } i
\end{aligned}
$$

The value of $\mathsf{quantity}_{\mathsf{at}}[i,j]$ is discussed in Section 6.2.3.

We use the convention of workload unit attributes beginning with lower case and device attributes beginning with upper case.

We define $x[I,J]$ as a shorthand for the set of $x[i,j]$ for all i such that $1 \leq i \leq D$ and for all j such that $1 \leq j \leq S$. Similarly, $x[i,J]$ is defined as the set of $x[i,j]$ such that $1 \leq j \leq S$.

The definition of $x[i,j]$ implies that the number of workload units from a given workload unit class assigned to all of the devices is equal to the number of workload units needed to be assigned from that class. This can be expressed in the following constraint equation:

$$\sum_{i=1}^{D} x[i,j] = \mathsf{size}[j] \tag{2}$$

for $1 \leq j \leq S$.

The value of $d[i]$ identifies whether device i is used in $x[I,J]$. This can be formalized as either one of the following constraint expressions:

$$d[i] = 1 \iff \exists\, j \text{ such that } 1 \leq j \leq S \text{ and } x[i,j] > 0 \tag{3}$$

$$d[i] = 1 \iff \sum_{j=1}^{S} x[i,j] > 0. \tag{4}$$

6.2 Objective functions

The following 3 sections present varying levels of complex objective functions.

In these sections, the value z is an approximation of the *value of the assignment* $x[I,J]$ (with respect to an objective function), which is the expected value of the objective function across the domain of workloads it will be faced with in real life.

6.2.1 Basic objective function

Let

$$
\begin{aligned}
\mathsf{importance}[j] &= \text{importance of a workload unit in the } j\text{th workload unit class} \\
\mathsf{Cost}[i] &= \text{cost of the } i\text{th device}
\end{aligned}
$$

Formally, our mapping problem can be defined as determining the set of values of $x[i,j]$ where the value of "benefit" of the workload unit set is maximized, which is expressed as

follows

$$z = \sum_{i=1}^{D} \sum_{j=1}^{S} \mathsf{importance}[j]x[i,j] \quad (5)$$

and the cost of the devices is minimized, which is expressed as follows

$$z = \sum_{i=1}^{D} \mathsf{Cost}[i]d[i]. \quad (6)$$

(5) and (6) can be written as one objective function where w represents the weight between the cost of the devices and the benefit of the workload units:

$$z = \sum_{i=1}^{D} \sum_{j=1}^{S} \mathsf{importance}[j]x[i,j] - w \sum_{i=1}^{D} \mathsf{Cost}[i]d[i]. \quad (7)$$

6.2.2 Support cost of devices

The physical cost of a storage system is not just limited to the cost of the devices. There are also costs associated with the power, space, cabinet, etc. These costs do not have to grow linearly with the number of devices supported. Let[9]

$$\mathsf{Support\_Cost}[n] = \text{support cost for the use of } n \text{ devices}$$

Define $\mathsf{Number\_Devices}$ as $\sum_{i=1}^{D} d[i]$. The objective function (7) can be replaced with the following equation to take into account the cost of supporting the devices:

$$z = \sum_{i=1}^{D} \sum_{j=1}^{S} \mathsf{importance}[j]x[i,j] - w \left(\sum_{i=1}^{D} \mathsf{Cost}[i]d[i] + \mathsf{Support\_Cost}[\mathsf{Number\_Devices}] \right). \quad (8)$$

6.2.3 Utility

We use the ideas presented in Section 3.1.2 to determine how the values specified for the workload unit attributes have to be meet. As discussed in Section 3.1.2, each workload unit attribute has a corresponding utility function, which gives the utility of the various possible values of the attribute. (See Figures 2, 3, and 4 for sample utility graphs.) Let

$w_{\mathsf{at}}[j]$ = the weight of attribute at for a workload unit in workload unit class j

$\mathsf{quantity}_{\mathsf{at}}[i,j]$ = the amount of attribute at that a workload unit in workload unit class j is assigned on device i

$\mathsf{utility}_{\mathsf{at}}[j, \mathsf{AT}]$ = the utility of attribute at for a workload unit in workload unit class j with the value of the device attribute being AT

[9] **Simplification:** We assume that all devices have the same cost to support.

All of the attributes for a workload unit must have a non-zero value for an assignment to be made. This is formalized in the following constraint expression:

$$\mathsf{utility}_{\mathsf{at}}[j, \mathsf{quantity}_{\mathsf{at}}[i,j]] < 0 \tag{9}$$

for all workload requirement attributes at, $1 \leq j \leq S$ such that $x[i,j] > 0$ where $1 \leq i \leq D$. (This constraint could be implemented by making the zero utility values have the value negative infinity.)

The utility of a given workload unit assignment is part of the "benefit" of the assignment. Therefore, the sum over all of the requirement attributes is multiplied into the benefit summand of objective function (8), giving us:

$$\begin{aligned} z \;=\; & \sum_{i=1}^{D}\sum_{j=1}^{S}\left(\mathsf{importance}[j]x[i,j] \sum_{\text{all attributes at}} w_{\mathsf{at}}[j]\mathsf{utility}_{\mathsf{at}}[j, \mathsf{quantity}_{\mathsf{at}}[i,j]]\right) \\ & - w\left(\sum_{i=1}^{D}\mathsf{Cost}[i]d[i] + \mathsf{Support\_Cost}\left[\mathsf{Number\_Devices}\right]\right). \end{aligned} \tag{10}$$

This function replaces (8).

It is not intuitive how to use the above objective function without having to explore all possible values of $\mathsf{quantity}_{\mathsf{at}}$; we suggest that the following idea be implemented. The system should determine what reasonable values are for each attribute for each workload unit class, set the values and run the solver. The result of the objective function (i.e., the value of z) should be compared with the previous best value of the objective function and the best should be saved. Then, the values of the workload unit attributes should be adjusted, and the procedure iterated.

The attributes are set (and adjusted) so that the only range tested is where the value of the utility is non-zero. The adjustment mirrors the type of function that the utility function is, e.g., if the utility function is linear for a specific attribute, then the value set will be increased at a linear rate. The value set also should depend on the importance of the workload unit and the weight of the attribute.[10]

6.3 Transfer rate

A method is needed to verify that a device has adequate bandwidth to support the workload units assigned to it. Let

$$\begin{aligned} \mathsf{Transfer\_Rate}[i] &= \text{the nominal transfer rate of } i\text{th device} \\ \mathsf{Position\_Time}[i,j] &= \text{the time to position the head of device } i \end{aligned}$$

[10] **Simplification:** This maps, for every set of workload units that are part of a workload unit class, the same amount of resources, i.e., $\mathsf{quantity}_{\mathsf{at}}[i,j]$ is the same for all i. But, different devices should be able to support different amounts of resources for the same workload unit class. Therefore, more work on the algorithm is needed.

when it is servicing request from a workload
in workload unit class j

$\mathsf{Position\_Time}[i,j]$ is defined in (22).

The amount of time that it takes to service a request from the current workload unit is the time to position the head and to transfer the data:

$$\mathsf{Service\_Time}[i,j] = \mathsf{Position\_Time}[i,j] + \frac{\mathsf{request\_size}[j]}{\mathsf{Transfer\_Rate}[i]}. \tag{11}$$

The $\mathsf{Long\_Term\_Transfer\_Rate}[i,j]$ is the rate at which device i services requests from workload unit class j:

$$\mathsf{Long\_Term\_Transfer\_Rate}[i,j] = \frac{\mathsf{request\_size}[j]}{\mathsf{Service\_Time}[i,j]}. \tag{12}$$

The workload unit long-term transfer rate equation. The long-term transfer rate that the workload unit gets must be greater than or equal to what it needs. This gives us the following constraint equation:

$$\mathsf{long\_term\_transfer\_rate}[j] \leq \mathsf{Long\_Term\_Transfer\_Rate}[i,j] \tag{13}$$

for $1 \leq i \leq D$ and for $1 \leq j \leq S$ such that $x[i,j] > 0$. We call this constraint equation the *workload unit long-term transfer rate* equation.

The service time utilization equation. The number of bytes needed by all of the workload units in a time interval must not exceed what the device can process in that time interval; this constraint is just the simple utilization equation:

$$\sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{Service\_Time}[i,j] < 1 \tag{14}$$

for $1 \leq i \leq D$. We call this constraint equation the *service time utilization* equation. This equation replaces (13) since $\mathsf{long\_term\_transfer\_rate}[j] = \mathsf{request\_size}[j] \cdot \mathsf{request\_rate}[j]$.

The device long-term transfer rate equation. We also must verify that the device's bandwidth is not over-exceeded. That is, in any one second, the amount of data that all of the workload units assigned to the device need can be produced by the device. To do this, we calculate the fraction of time that will be spent positioning the head for the current assignments made to the device. This is

$$\sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{Position\_Time}[i,j].$$

If we multiply the above value by the transfer rate of the device, we compute the amount of bandwidth that can be considered positioning time overhead for the current set of assignments. This is

$$\mathsf{Transfer\_Rate}[i] \sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{Position\_Time}[i,j].$$

The amount of data that all of the workload units need (in one time interval) is

$$\sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{long\_term\_transfer\_rate}[j]\frac{\mathsf{request\_size}[j]}{\mathsf{Long\_Term\_Transfer\_Rate}[i,j]}.$$

Therefore, to verify that the devices can service the requests without exceeding their bandwidth, we have the following constraint equation:

$$\begin{aligned} &\sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{long\_term\_transfer\_rate}[j]\frac{\mathsf{request\_size}[j]}{\mathsf{Long\_Term\_Transfer\_Rate}[i,j]} \\ \leq\ & \mathsf{Transfer\_Rate}[i] - \\ & \mathsf{Transfer\_Rate}[i] \sum_{j=1}^{S} x[i,j]\mathsf{request\_rate}[j]\mathsf{Position\_Time}[i,j] \end{aligned} \tag{15}$$

for $1 \leq i \leq D$. We call this constraint equation the *device long-term transfer rate* equation. This constraint equation does not need to be supported; it is implied by (13) and (14).

6.4 Last-byte latency

We define the *last-byte latency* to be the amount of time that it takes from request time to the time that the last byte is received.

Let

$\mathsf{last\_byte\_latency}[j]$	=	last-byte latency that can be tolerated by a workload unit in the jth workload unit class
$\mathsf{Last\_Byte\_Latency}[i,j,x[i,J]]$	=	the last-byte latency that is seen from a workload unit from the jth workload unit class assigned to device i when the assignments $x[i,J]$ are made
$\mathsf{request\_size}[j]$	=	the request size of a workload unit in the jth workload unit class
$\mathsf{Transfer\_Rate}[i,j]$	=	transfer rate of the ith device given a request from a workload unit in workload unit class j

$$\mathsf{Position\_Time}[i,j] \;=\; \text{the time to position the head of device } i \text{ so that a request from workload unit class } j \text{ can begin}$$

$\mathsf{Position\_Time}[i,j]$ is defined in (22). (The term $x[i,J]$ had been defined in Section 6.1.)

When a new workload unit is assigned to a device, the last-byte latency of the device changes for the workload units already assigned to the device. (This is because the last-byte latency depends on the positioning time of the disk head and the positioning time is workload-dependent.) So, the requirements for an assigned workload unit might not be able to be met anymore. Therefore, the constraint expression must verify the requirements for all workload unit classes, not just the one currently being considered. This is reflected in our constraint expression by "$1 \leq j' \leq S$." (We call this a *feedback loop*.)

A workload unit can only be assigned to a device if the current last-byte latency for the device is less than or equal to the workload unit's needs **and** if adding the workload unit does not violate the latency-requirements for the workload units currently assigned to the device (as discussed above). Thus we get the following constraint expression

$$\mathsf{last\_byte\_latency}[j'] \;\geq\; \mathsf{Last\_Byte\_Latency}[i,j',x[i,J]] \tag{16}$$

for $1 \leq j' \leq S$ such that $x[A,j'] > 0$, $1 \leq j \leq S$ such that $x[A,j] > 0$, and $1 \leq i \leq D$.

The device last-byte latency is easy to model when just considering a simple device since the last-byte latency is just the time needed to service the request ($\mathsf{Service\_Time}[i,j]$ as defined in (11)):

$$\mathsf{Last\_Byte\_Latency}[i,j,x[i,J]] \;=\; \mathsf{Service\_Time}[i,j]. \tag{17}$$

If, for example, the device has a queue, the last-byte latency would include the *queue delay*, the time the request is on the queue.

6.5 Capacity

Devices have limited capacity to store data. Some workload units have well-defined requirements for the amount of data that needs to be stored. Let

$\mathsf{capacity}[j]$ = the capacity needed by a workload unit in the jth workload unit class
$\mathsf{Capacity}[i]$ = the capacity of the ith device

A device should have enough capacity to service all of the workload units' data assigned to it. This can be expressed as the following constraint expression

$$\sum_{j=1}^{S} \mathsf{capacity}[j]x[i,j] \leq \mathsf{Capacity}[i] \tag{18}$$

for $1 \leq i \leq D$.

6.6 Availability and reliability

Let

$$\begin{aligned} \mathsf{availability}[j] &= \text{availability needed by a workload unit in the } j\text{th workload unit class} \\ \mathsf{reliability}[j] &= \text{reliability needed by a workload unit in the } j\text{th workload unit class} \\ \mathsf{Availability}[i] &= \text{availability of the } i\text{th device} \\ \mathsf{Reliability}[i] &= \text{reliability of the } i\text{th device} \end{aligned}$$

We treat availability and reliability as values that do not change based on the load on the device. Therefore, the necessary constraint expressions do not depend on what workload units are assigned to the device, but just on the absolute values for the device. We see this as follows:

$$\mathsf{availability}[j] \leq \mathsf{Availability}[i] \tag{19}$$

$$\mathsf{reliability}[j] \leq \mathsf{Reliability}[i] \tag{20}$$

for $x[i,j] > 0$, $1 \leq j \leq S$, and $1 \leq i \leq D$.[11]

6.7 Correctness violations

Let

$$\begin{aligned} \mathsf{correct\_viol}[j] &= \text{the fraction of data that a workload unit from the} \\ &\quad\ j\text{th workload unit class can tolerate incorrectly transfered} \\ \mathsf{Correct\_Viol}[j] &= \text{the fraction of data that device } i \text{ incorrectly transfers} \end{aligned}$$

The correctness violations do not change based on the load on the device. Therefore, the necessary constraint expressions do not depend on what workload units are assigned to the device, but just on the absolute values for the device. We see this as follows:

$$\mathsf{correct\_viol}[j] \leq \mathsf{Correct\_Viol}[i] \tag{21}$$

for $x[i,j] > 0$, $1 \leq i \leq D$, and $1 \leq j \leq S$.

6.8 Position time

The time to position the head on the desired block is the seek time plus the rotational latency delay:

$$\mathsf{Position\_Time}[i,j] = \mathsf{Seek\_Time}[i,j] + \mathsf{Rot\_Lat}[i,j] \tag{22}$$

[11] **Simplification:** We do not discuss independent failure modes.

where $\mathsf{Seek\_Time}[i,j]$ is the time to move the device head into place and $\mathsf{Rot\_Lat}[i,j]$ is the time to rotate the device so that the desired block is under the head. Position time also includes the protocol overhead and the time to perform a head switch if needed. The protocol overhead is the time needed to decode the command and to check the ECC. We are not modeling either one of these at this time; they are insignificant when compared to the seek time and the rotational latency. For example, [Hospodor95] states that the time to perform a head switch is les than 1 ms and the seek time was 3–25 ms.

The seek time and rotational latency are workload dependent. This can be seen by considering two workloads—one where the requests are consecutive and the other where the requests are random. The mean seek time for the consecutive workload will be very small since the device head will not have to move very frequently; the mean seek time for the random workload is much higher. The mean rotational latency for the both workloads depends on the request rate.

We analyze the values of $\mathsf{Seek\_Time}[i,j]$ and $\mathsf{Rot\_Lat}[i,j]$[12] based on the assigned workload units' access pattern attributes. We are limited by our lack of knowledge of which cylinders the workload unit's data will be located on. (We also do not know what zones the data are on.) We assume the blocks of the file being accessed by a workload unit are stored consecutively on a cylinder and then on neighboring cylinders.

In the following sections, we analyze the seek distance, seek time, and rotational latency based on the access patterns of the workload units assigned to the devices. The access patterns that we consider are: uniform random access, consecutive access, sequential access, and single hot spots.

6.8.1 Seek distance

The seek distance represent the integer number of cylinders traversed between the current cylinder and the requested cylinder. We represent it as a discrete random variable. In this section, we formalize the expression of expected seek distance when servicing a request. (The following analysis is similar to what is presented in [Chen92].)

Let $\Pr(\mathsf{dis} = k)$ be the probability that the distance that needs to be seeked on device i to service a request from the jth workload unit class is equal to k tracks. We have the following three cases:

- If the workload unit assigned to device i has a uniform random access pattern, we have

$$\Pr(\mathsf{dis} = k) = \begin{cases} 0 & k = 0 \\ \frac{2(\mathsf{Num\_of\_Cylinders}[i]-k)}{\mathsf{Num\_of\_Cylinders}[i](\mathsf{Num\_of\_Cylinders}[i]-1)} & \text{otherwise} \end{cases} \tag{23}$$

[12] It would be more correct to represent the parameters of Seek_Time and Rot_Lat as $[i, x[i, J]]$ since we do not think it is worth it to model seek time and rotational latency to the point that we could determine different values for different workload units assigned to the same device.

The $\frac{2(\mathsf{Num\_of\_Cylinders}[i]-k)}{\mathsf{Num\_of\_Cylinders}[i](\mathsf{Num\_of\_Cylinders}[i]-1)}$ quantity comes from the fact that the location of the first request and the location of the next request are randomly located on disk.

Note that the above discussion assumes that the data accessed is randomly distributed across the device. This would only be true if the device has one zone. Since the zones have varying amounts of data stored on them, a multiple zoned disk does not have the data randomly distributed across the cylinders.

- If the workload unit assigned to device i has a consecutive access pattern, we have

$$\Pr(\mathsf{dis} = k) = \begin{cases} 1 - \frac{\mathsf{request\_size}[j]}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]} & k = 0 \\ \frac{\mathsf{request\_size}[j]}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]} & k = 1 \\ 0 & \text{otherwise} \end{cases}$$

  The above approximation does not take into account the time needed to settle the head when moving between platters.

- If the workload unit assigned to device i has a sequential access pattern, we have

$$\Pr(\mathsf{dis} = k) = \begin{cases} \frac{\text{num\_left}}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]} & k = \lfloor \text{num\_skipped} \rfloor \\ 1 - \frac{\text{num\_left}}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]} & k = \lceil \text{num\_skipped} \rceil \\ 0 & \text{otherwise} \end{cases}$$

  where

$$\begin{aligned} \text{num\_skipped} &= \frac{\mathsf{run\_stride}[j] - \mathsf{run\_length}[j]}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]} \\ \text{num\_left} &= \mathsf{run\_stride}[j] - \mathsf{run\_length}[j] \bmod \mathsf{Ave\_Bytes\_per\_Cylinder}[i]. \end{aligned}$$

The above values for $\Pr(\mathsf{dis} = k)$ are based on one workload unit being assigned to a device. In reality, a device supports many workload units. We, therefore, need to approximate the seek distance in such instances. Studies on interactive timesharing, office automation, transaction processing and financial batch systems [Bates91] have shown that the average seek distance is $\frac{\mathsf{Num\_of\_Cylinders}[i]}{10}$ and about half of the requests result in seeks of less than one cylinder. [Hospodor95] states that the average seek distances for a multi-user Unix system is .155 of a stroke. Thus, we suggest that instead of using the analytic method of determining the seek distance, the empirical distance of $\frac{\mathsf{Num\_of\_Cylinders}[i]}{10}$ be used.

6.8.2 Seek time

There are two predominant methods used to specify device seek time: a single average seek time (i.e., $\mathsf{Ave\_Seek\_Time}[i]$) or the seek time needed as a function of the number of cylinders being seeked (i.e., $\mathsf{Seek\_Time}[i, \mathsf{dis}]$ where dis is the number of cylinders being seeked).

We discuss the seek time needed for one workload unit assigned to a device for uniform random access, consecutive access, sequential access, and single hot spots. We then discuss the seek time needed for the empirical method of approximating the seek distance when multiple workload units are assigned to a device.

Uniform random access

We use standard approximation for $\mathsf{Seek\_Time}[i,j]$ when the values of $\mathsf{run\_length}[j]$, $\mathsf{run\_stride}[j]$, and $\mathsf{data\_skew}[j]$ of the assigned workload units identify the workload units as having uniform random access.

If the average seek time is specified for the device ($\mathsf{Ave\_Seek\_Time}$), we can use that value to approximate our seek time:

$$\mathsf{Seek\_Time}_{\mathrm{ran}}[i,j] = \mathsf{Ave\_Seek\_Time}[i]. \tag{24}$$

If the average seek time for the device is not specified, but the seek curve is, we can calculate the average seek time. The average seek time is the weighted average of the seek time for each possible seek distance:

$$\mathsf{Ave\_Seek\_Time}[i,j] = \sum_{k=1}^{\mathsf{Num\_of\_Cylinders}[i]} \mathbf{Pr}(\mathsf{dis}=k) \cdot \mathsf{Seek\_Time}[i,k].$$

Using the value of $\mathbf{Pr}(\mathsf{dis}=k)$ from (23), we get

$$\mathsf{Ave\_Seek\_Time}[i] = \frac{2\sum_{\mathsf{dis}=1}^{\mathsf{Num\_of\_Cylinders}[i]}(\mathsf{Num\_of\_Cylinders}[i]-\mathsf{dis})\mathsf{Seek\_Time}[i,\mathsf{dis}]}{\mathsf{Num\_of\_Cylinders}[i](\mathsf{Num\_of\_Cylinders}[i]-1)}.$$

Consecutive access

If there is only one workload unit assigned to the device, and the values of $\mathsf{run\_length}[j]$, $\mathsf{run\_stride}[j]$, and $\mathsf{data\_skew}[j]$ identify the workload unit as having a uniform consecutive access pattern, we can get tighter approximations to the seek time than for the case of uniform random access.

Assuming that the disk must access all of the sectors on one track, and then access the sectors on the next platter (same track), etc., the average seek time for all of the accesses to read the data on one cylinder is

$$\begin{aligned}
&\mathsf{Seek_Time}_{\mathrm{cons1}}[i,j] \\
&= \frac{\text{amount of time spent performing seeks}}{\text{number of requests}} \\
&= \frac{\mathsf{Seek_Time}_{\mathrm{ran}}[i,j] + \mathsf{Num_of_Platters}[i] \cdot \mathsf{Head_Switch_Time}[i]}{\left\lfloor \frac{\mathsf{Ave_Bytes_per_Cylinder}[i]}{\mathsf{request_size}[j]} \right\rfloor}.
\end{aligned}$$

If the amount of data accessed is larger than a cylinder (i.e., $\mathsf{run\_length}[j] \geq \mathsf{Ave\_Bytes\_per\_Cylinder}[i]$), then there is a seek that occurs when the head moves to the next cylinder. The amount of time spent performing seeks to read the data that is stored on n cylinders is contributed by the seek to the first cylinder, $n-1$ seeks to the next cylinder, and the seeks within one cylinder for each of the n cylinders. This is

$$\begin{aligned}&\mathsf{Seek\_Time}_{\mathrm{ran}}[i,j] + (n-1)\mathsf{Seek\_Time}[i,1] \\ &+ n \cdot \mathsf{Num\_of\_Platters}[i] \cdot \mathsf{Head\_Switch\_Time}[i]. \end{aligned} \tag{25}$$

The total number of requests that can be serviced by the data stored on the n cylinders is

$$n \left\lfloor \frac{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]}{\mathsf{request\_size}[j]} \right\rfloor. \tag{26}$$

We can put these together to get

$$\mathsf{Seek\_Time}_{\mathrm{cons}}[i,j] = \frac{\text{formula (25)}}{\text{formula (26)}} \tag{27}$$

where $n = \frac{\mathsf{run\_length}[j]}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]}$.

Sequential access

If we have one workload unit assigned to the device and it has uniform sequential access (defined by the values of $\mathsf{run\_length}[j]$, $\mathsf{run\_stride}[j]$, and $\mathsf{data\_skew}[j]$), our $\mathsf{Seek\_Time}[i,j]$ derivation becomes more complex.

When analyzing $\mathsf{Seek\_Time}[i,j]$, we perform case analysis on the different possible sizes of the runs.

- $\mathsf{run\_length}[j] \geq \mathsf{Ave\_Bytes\_per\_Cylinder}[i]$ (i.e., the runs are contained on one or more cylinders)

  If the runs are contained on one or more cylinders, the amount of time spent performing seeks to read one run that is stored on n cylinders is contributed by the seek to the first cylinder, the $n-1$ seeks to the next cylinder, and the seeks within one cylinder for each of the n cylinders. Formally, we have

  $$\begin{aligned}&\mathsf{Seek\_Time}_{\mathrm{ran}}[i,j] + (n-1)\mathsf{Seek\_Time}[i,1] \\ &+ n \cdot \mathsf{Num\_of\_Platters}[i] \cdot \mathsf{Head\_Switch\_Time}[i] \end{aligned} \tag{28}$$

  where $n = \frac{\mathsf{run\_length}[j]}{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]}$.

  The calculation of the number of requests that can be serviced from n cylinders must take into account that some of the bytes are skipped. Approximately, $\mathsf{run\_length}[j]/\mathsf{run\_stride}[j]$ of the bytes are desired, so the number of requests is

  $$\left\lfloor \frac{\left\lfloor \frac{n \cdot \mathsf{Ave\_Bytes\_per\_Cylinder}[i]}{\mathsf{run\_stride}[j]} \right\rfloor \mathsf{run\_length}[j]}{\mathsf{request\_size}[j]} \right\rfloor. \tag{29}$$

- $\mathsf{Ave\_Bytes\_per\_Track}[i] \leq \mathsf{run\_length}[j] \leq \mathsf{Ave\_Bytes\_per\_Cylinder}[i]$ (i.e., multiple runs fit on a cylinder, but not on one track)

  The amount of time spent performing seeks to read the data stored on one cylinder is

  $$\mathsf{Seek\_Time}_{\mathrm{ran}}[i,j] + \mathsf{Num\_of\_Platters\_Accessed}[i,j] \cdot \mathsf{Head\_Switch\_Time}[i] \tag{30}$$

  where $\mathsf{Num\_of\_Platters\_Accessed}[i,j]$ depends on whether the run stride is greater than the number of bytes on a track or not. If $\mathsf{run\_stride}[j] \geq \mathsf{Ave\_Bytes\_per\_Track}[i]$, the number of platters that are accessed is the product of the number of platters that a run is across and the number of runs there are on the cylinder:

  $$\begin{aligned} &\mathsf{Num\_of\_Platters\_Accessed}[i,j] \\ &\quad = \mathsf{run\_length}[j]/\mathsf{Ave\_Bytes\_per\_Track}[i] \\ &\qquad \cdot \mathsf{Ave\_Bytes\_per\_Cylinder}[i]/\mathsf{run\_stride}[j]. \end{aligned}$$

  If $\mathsf{run\_stride}[j] \leq \mathsf{Ave\_Bytes\_per\_Track}[i]$, every platter is accessed:

  $$\mathsf{Num\_of\_Platters\_Accessed}[i,j] \quad = \quad \mathsf{Num\_of\_Platters}[i].$$

  The calculation of the number of requests that can be serviced from one cylinder must take into account that some of the bytes are skipped. Approximately, $\mathsf{run\_length}[j]/\mathsf{run\_stride}[j]$ of the bytes are desired, so the number of requests is

  $$\frac{\left\lfloor \frac{\mathsf{Ave\_Bytes\_per\_Cylinder}[i]}{\mathsf{run\_stride}[j]} \right\rfloor \mathsf{run\_length}[j]}{\mathsf{request\_size}[j]}. \tag{31}$$

- $\mathsf{run\_length}[j] \leq \mathsf{Ave\_Bytes\_per\_Track}[i]$ (i.e., multiple runs fit on a track)

  The amount of time spent performing seeks to read the data requested from one cylinder is

  $$\mathsf{Seek\_Time}_{\mathrm{ran}}[i,j] + \mathsf{Num\_of\_Platters}[i] \cdot \mathsf{Head\_Switch\_Time}[i]. \tag{32}$$

  The number of requests that can be serviced by the data stored on one cylinder is

  $$\left\lfloor \frac{\left\lfloor \frac{\mathsf{Ave\_Bytes\_per\_Track}[i]}{\mathsf{run\_stride}[j]} \right\rfloor \mathsf{run\_length}[j]}{\mathsf{request\_size}[j]} \right\rfloor \mathsf{Num\_of\_Platters}[i]. \tag{33}$$

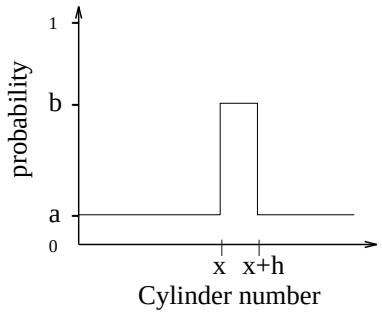

Figure 7: A sample distribution of uniform accesses with a hot spot of size h.

Hot spots/highly skewed random access

If one workload unit is assigned to device i and the spatial access patterns define the workload unit as having a hot spot of size $h = \lceil \mathsf{hot\_spot\_size}[j]/\mathsf{Ave\_Bytes\_per\_Cylinder}[i] \rceil$ cylinders, then the amount of time spent doing seeks is the weighted sum of the time doing seeks when a request and the following request are both in the hot spot and when the requests are not both in the hot spot:

$$\mathsf{Seek\_Time}_{\text{hs-ran}}[i,j] \quad = \quad p \cdot \mathsf{Seek\_Time}[i,h] + (1-p) \cdot \mathsf{Seek\_Time}_{\text{ran}}[i,j]$$

where p is the probability that both a request and the following are in the hot spot. If the probability density curve is as shown in Figure 7, the value of p is $((b-a)h)^2$.

Empirical seek time

If the average seek distance is $\frac{\mathsf{Num\_of\_Cylinders}[i]}{10}$ as specified above, the seek time is:

$$\mathsf{Seek\_Time}[i,j] \quad = \quad \mathsf{Seek\_Time}[i, \mathsf{Num\_of\_Cylinders}[i]/10]. \tag{34}$$

6.8.3 Rotational latency

The standard approximation for rotational latency is 1/2 the time to rotate the disk a full revolution:

$$\mathsf{Rot\_Lat}[i,j] \quad = \quad \mathsf{Ave\_Rot\_Time}[i, \mathsf{Ave\_Bytes\_per\_Track}[i]/2] \tag{35}$$

where $\mathsf{Ave\_Rot\_Time}[]$ and $\mathsf{Ave\_Bytes\_per\_Track}[]$ are defined in Section 4.1.

If the access patters are consecutive or sequential, we can develop a closer approximation.

Consecutive access

If the workload unit has a consecutive access pattern, we must calculate the amount of time that it takes to rotate the disk once the next request arrives to determine the amount of time spent for rotational latency. Since the desired data is the next data on the disk, the disk has to at least make a complete revolution before servicing the next read. Since the device is rotating during the think time of the workload unit, we do not count this to the rotational latency needed to service the request. We get

$$\begin{aligned}&\mathsf{Rot\_Lat}_{\text{cons}}[i,j]\\ &\quad = \mathsf{Ave\_Rot\_Time}[i, \mathsf{Ave\_Bytes\_per\_Track}[i]]\\ &\qquad - (\mathsf{think\_time}[j] \bmod \mathsf{Ave\_Rot\_Time}[i, \mathsf{Ave\_Bytes\_per\_Track}[i]]).\end{aligned} \tag{36}$$

Sequential access

There are two different cases to rotational latency: when the data is being accessed consecutively (since a run can last for many requests) and when data is skipped.

If the data is being accessed consecutively, the rotational latency is what (36) gives us. The number of bytes that is skipped that affects the rotational latency is $b = (\mathsf{run\_stride}[j] - \mathsf{run\_length}[j]) \bmod \mathsf{Ave\_Bytes\_per\_Track}[i]$. Therefore, the rotational latency when data is skipped is

$$\begin{aligned}&\mathsf{Rot\_Lat}_{\text{seq-sk}}[i,j]\\ &\quad = \mathsf{Ave\_Rot\_Time}[i, b]\\ &\qquad - (\mathsf{think\_time}[j] \bmod \mathsf{Ave\_Rot\_Time}[i, \mathsf{Ave\_Bytes\_per\_Track}[i]]).\end{aligned} \tag{37}$$

The rotational latency is the weighted average of the two cases:

$$\mathsf{Rot\_Lat}_{\text{seq}}[i,j] = w \cdot \mathsf{Rot\_Lat}_{\text{cons}}[i,j] + (1-w) \cdot \mathsf{Rot\_Lat}_{\text{seq-sk}}[i,j] \tag{38}$$

where

$$\begin{aligned} w &= \frac{\#\text{ of requests that seem consecutive}}{\#\text{ of requests}}\\ &= \frac{\left\lfloor \frac{\mathsf{run\_length}[j]}{\mathsf{request\_size}[j]} \right\rfloor}{\frac{\mathsf{Ave\_Bytes\_per\_Track}[i]}{\mathsf{run\_stride}[j]}}. \end{aligned}$$

6.9 Open issues for the mapping problem

There are a number of open issues in the mapping problem:

- How are blank workload units and device attributes handled when the mapping is being done?

- How should disk quotas and roles be modeled?
- If there are X workload units specified but we know that only Y of them will be run at once, how can the mapping be done so that percentile percentile of the combinations of workload units will have their requirements satisfied by the devices prob probability?
- We feel that the specification of attributes for both workload units and devices should be in the form of distributions. (Some work that has been done on distributions.) But, how to do the mapping is still an open question.
- Garth Gibson mentioned that the jitter depends on the zone of the device that the data is in. How should this be modeled?

7 Previous work

Our approach of specifying application attributes and mapping this to resources is similar to [Franken95]; Franken et. al. define applications and resources in terms of *properties*, where a property is a characteristic of an element that can be identified by examining the element. Properties of multimedia applications and computing and communication resources are used to determine the computing and communication resources that maximize the probability that a failure does not occur while the resources are being utilized for the stated applications.

7.1 Previous workload model work

Most people have developed workload models by examining workload traces and determining what low-level operations can be used to distinguish between different trace results or by some other method of determining low-level operations or commands ([Calzarossa94] is an example of this). This type of approach is not helpful to us; we need a method that identifies the workload by its requirements of the storage system.

There has been recent work on characterizing the workload of scientific applications in terms of file system usage on parallel machines [Nieuwejaar95]. Unfortunately, the results have not included a general workload model.

One model that considers the I/O is SynRGen [Ebling94] which uses parameterized *micromodels* to model user actions against a file system. Since SynRGen was developed to stress-test a file system, the number of files, symbolic and hard links, and the directory hierarchy are important; these things are not important to us.

The idea of specifying requirements of streams is not new; [Schill95] specifies the *quality of service* requirements of various multimedia data streams to test high performance transport systems.

7.2 Previous device model work

There has been some work in device modelling, with two major approaches: practical and theoretical. [Ruemmler94] and [Worthington94, Worthington95] are simulator-based methods of disk modelling. [Ruemmler94] models sector size, cylinders, tracks per cylinders, data sectors per track, number of zones, track skew, cylinder skew, among others. [Worthington95] presents a disk drive simulator which simulates such features as zoned recording, spare regions, defect slipping and reallocation, disk buffers and caches, various pre-fetch algorithms, fast writes, bus delays, control and communication overheads and command queuing. Both of these models are too detailed for our needs.

Louis and Teaff [Louis95] presents a storage class metadata structure which contains many device-dependent attributes which are similar to the ones in our model as seen in the following section.

The parallel disk model [Vitter94] is an example of a theoretical device model. It states that the data are stored on D devices in a round-robin fashion in constant-sized blocks. Data is accessed from the storage system in terms of *parallel I/Os*, which reads or writes D blocks, one per device. The model has been used to develop I/O-optimal algorithms where the number of parallel I/Os was counted.

8 Future work

There are many areas that we have not yet had time to explore and that seem like interesting areas for future work.

- How can the device model and mapping be enhanced so that the *flexibility* of the resulting system can be measured (and maximized)? The flexibility of a storage system is some measure of the amount of resources that are not assigned—the more resources non-assigned, the greater the flexibility. (It seems that the measure of flexibility should be summed over the consumable attributes with the generatable attributes subtracted.)

- How can this method of objective functions and constraint equations/expressions be used to solve the *self-managing storage system problem*, i.e., some workloads units have been assigned and new ones are coming in; how should it be determined if the assigned workload units should be moved around?

  The ideas for dynamic data placement from [Weikum90] might be helpful. [Wolf89] uses a variable which they call tweak limit that presents the number of files which can be moved from their current disks when their system is changing assignments as a result of a changing workload; we might want to do something similar.

9 Conclusions

This document gives our current view on the workload and device models, and the necessary objective functions and constraint expressions needed to formalize the mapping of workload units to devices. An assumption made throughout is that single values are used to specify both the workload and the devices. We state the instances where we are making simplifying assumptions.

We learned a number of things:

- Even though many parts of the problem seem hard, there are easy parts. (We think we solved most of these.)
- The objective function and constraint expression approach seems like a good way to specify and model the problem.
- There are a number of different questions that can be asked for attribute mapping; we modeled these with sets, pools, and seas.

References

[Bates91] Ken Bates. *VAX I/O subsystems: optimizing performance.* Professional Press Books, Horsham PA 19044, 1991.

[Calzarossa94] M. Calzarossa and G. Serazzi. Construction and use of multiclass workload models. *Perf. Evaluation*, **19**(4):341–52, 1994.

[Campbell96] Andrew Campbell, Cristina Aurrecoechea, and Linda Hauw. A review of QoS architectures. *Proceedings of 4th International Workshop on Quality of Service (IWQoS96)* (Paris, France), pages 173–95, Jan de Meer and Andreas Vogel, editors, 7-8 March 1996.

[Chen92] Shenze Chen. *Design, modeling, and evaluation of high performance I/O subsystems.* PhD thesis. Department of Computer Science, University of Massachusetts, September 1992.

[Degermark95] M. Degermark, T. Kohler, S. Pink, and O. Schelen. Advance reservations for predictive service. *Proceedings of 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'95)* (Durham, NH), pages 3–14, 18–21 April 1995.

[Denning80] P. J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, **SE–6**(1):64–84, January 1980.

[Ebling94] Maria R. Ebling and M. Satyanarayanan. SynRGen: an extensible file reference generator. *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems 1994* (Nashville, TN). Published as *Performance Evaluation Review*, **22**(1):108–17, 16–20 May 1994.

[Franken95] Leonard J. N. Franken, Peter Janssen, Boudewijn R. H. M. Haverkort, and Gidi van Liempd. Quality of service management in distributed systems using dynamic routation. *Proceedings of 3rd International IFIP TC6 Conference on Open Distributing Processing (ICODP '95)* (Brisbane, Australia), pages 367–78, Kerry Raymond and Liz Armstrong, editors, 20–24 February 1995.

[Gibson93] Garth A. Gibson and David A. Patterson. Designing disk arrays for high data reliability. *Journal Parallel and Distributed Computing*, **17**(1–2):4–27. Academic Press, Incorporated, January/February 1993.

[Golding95] Richard Golding, Elizabeth Shriver, Tim Sullivan, and John Wilkes. Attribute-managed storage. *Workshop on Modeling and Specification of I/O* (San Antonio, TX), 26 October 1995. Available as Technical Report HPL–SSP–95–11, Storage Systems Program, Hewlett-Packard Laboratories.

[Gray86] Jim Gray. Why do computers stop and what can be done about it? *Proceedings of 5th Symposium on Reliability in Distributed Software and Database Systems*, pages 3–11. IEEE Computer Society Press, catalog number 86CH2260–8, 1986.

[Hospodor95] Andy Hospodor. Mechanical access time calculation. *Advances in Information Storage Systems*, **6**:313–36, 1995.

[Jensen91] E. Douglas Jensen. Alpha: a non-proprietary experimental operating system for distributed mission-critical real-time applications—an overview of its objectives and kernel abstractions. Concurrent Computer Corporation, Draft of 17 March 1991. Provided by the author.

[Louis95] S. Louis and D. Teaff. Class of service in high performance storage system. *Proceedings of 3rd International IFIP TC6 Conference on Open Distributing Processing (ICODP '95)* (Brisbane, Australia), pages 307–18, Kerry Raymond and Liz Armstrong, editors, 20–24 February 1995.

[Low93] Steven Low and Pravin Varaiya. Burstiness bounds for some burst reducing servers. *12th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM'93).* IEEE Computer Society, Cat. No.93CH3264-9, 1 April 1993.

[Majumdar84] Shikharesh Majumdar. *Locality and file referencing behaviour: principles and applications.* MSc thesis published as technical report 84–14. Department of Computer Science, University of Saskatchewan, Saskatoon, August 1984.

[Nieuwejaar95] Nils Nieuwejaar and David Kotz. Low-level interfaces for high-level parallel I/O. *3rd Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS'95)* (Santa Barbara, CA), pages 47–62, 25th April 1995.

[Patterson93] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *Operating Systems Review*, **27**(2):21–34, April 1993.

[Plagemann96] Thomas Plagemann. Protocol configuration—a flexible and efficient approach for QoS provision. *Proceedings of 4th International Workshop on Quality of Service (IWQoS96)* (Paris, France), pages 235–7, Jan de Meer and Andreas Vogel, editors, 7-8 March 1996.

[Ruemmler94] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, **27**(3):17–28, March 1994.

[Schill95] ASchill, C. Mittasch, T. Hutschenreuther, and F. Wildenhain. A quality of service abstraction tool for advanced distributed applications. *Proceedings of 3rd International IFIP TC6 Conference on Open Distributing Processing (ICODP '95)* (Brisbane, Australia), pages 353–64, Kerry Raymond and Liz Armstrong, editors, 20–24 February 1995.

[Siewiorek92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems: design and evaluation*. Digital Press, Second edition, 1992.

[Sreenan96] Cormac J. Sreenan. QOS support in a broadband multipoint server. *Proceedings of 4th International Workshop on Quality of Service (IWQoS96)* (Paris, France), pages 209–17, Jan de Meer and Andreas Vogel, editors, 7-8 March 1996.

[Vitter94] Jeffrey S. Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory I: two-level memories. *Algorithmica*, **12**(2/3):110–47, August and September 1994.

[Weikum90] Gerhard Weikum, Peter Zabback, and Peter Scheuermann. *Dynamic file allocation in disk arrays*. Technical report 147. Department of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland, December 1990.

[Wolf89] Joel Wolf. The placement optimization program: a practical solution to the disk file assignment problem. *1989 ACM SIGMETRICS and Performance '89 International Conference on Measurement and Modeling of Computer Systems* (Berkeley, California, May 1989). Published as *Performance Evaluation Review*, **17**(1):1–10. ACM, May 1989.

[Wolf95] L. Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig. Issues of reserving resources in advance. *Proceedings of 5th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'95)* (Durham, NH), pages 27–37, 18–21 April 1995.

[Worthington94] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt. *Scheduling for modern disk drives and non-random workloads.* Technical report CSE–TR–194–94. Department of Computer Science and Engineering, University of Michigan, March 1994.

[Worthington95] Bruce L. Worthington. *Aggressive centralized and distributed scheduling of disk requests.* PhD thesis, published as Technical report CSE–TR–244–95. Department of Computer Science and Engineering, University of Michigan, June 1995.

[Zipf49] George K. Zipf. *Human behavior and the principle of least effort.* Addison-Wesley, Reading, MA, 1949.