Chapter 10

Disk I/O

10.1. Introduction

Processor and primary memory technology has moved forward rapidly in recent years. Comparable advances have not occurred in the design of I/O subsystems. As a result, I/O subsystems are playing an increasingly critical role in computer system performance. Queueing network models of disk I/O subsystems are the subject of the present chapter.

In any study involving queueing network models, the analyst must begin by determining which system devices should be represented as service centers in the model, and what the service demands at these centers should be. With these parameters as input, the computational algorithms described in Part II use Little's law to calculate the effect of resource contention, yielding performance measures such as utilizations, throughputs, residence times, and queue lengths. Most postulated modifications to the system or to the workload are represented in the model as modifications to the service demands.

The "canonical" queueing network model that we have used throughout the book consists of service centers representing the CPU and the individual disk devices. Such a model is a very abstract representation of the contemporary IBM disk I/O subsystem configuration illustrated in Figure 10.1. The architectural complexity of this subsystem results from difficult compromises between cost and performance. At one extreme, requiring the CPU to monitor directly all phases of I/O activity would lead to poor performance (although low cost). At the other extreme, endowing each disk with sufficient intelligence to transfer data in a fully independent manner would lead to high cost (although good performance). The obvious approach is to introduce some number of shared devices of varying intelligence (channels, controllers, string heads, etc.) on the *path* between the CPU and the disks.

How is it that a simple model, which does not represent explicitly the many I/O path elements, can validate? The answer is that, typically, the effects of these "details" are captured in the disk service demands

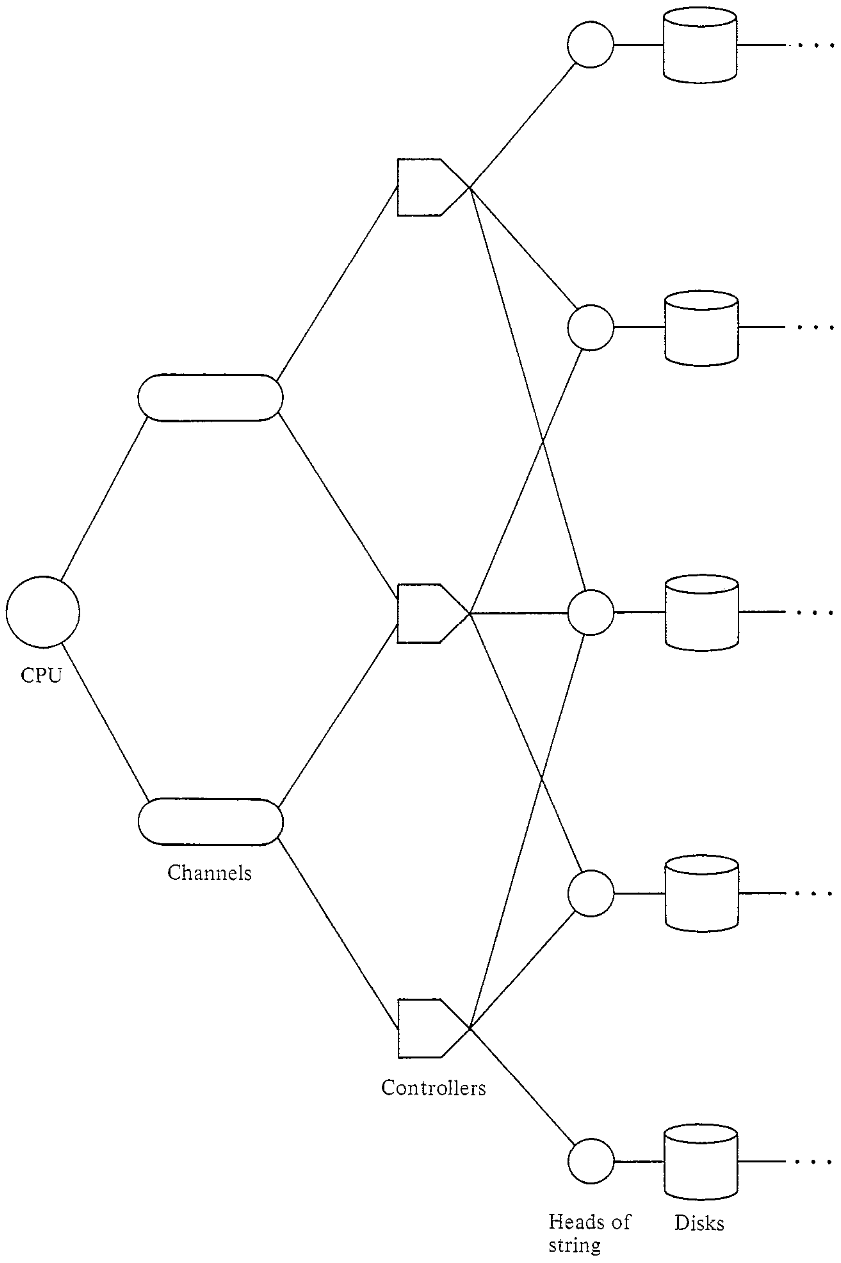

Figure 10.1 – A Contemporary IBM I/O Subsystem

obtained from measurement data. There are three intrinsic components of disk service time: *seek* (the time required to position the arm to the correct cylinder), *latency* (the time required for the start of the data record of interest to rotate under the heads) and *transfer* (the time required for the actual transfer of data). In addition, though, a disk is "held" by a customer during a *contention* period when data cannot be transferred due to the absence of a path back to the CPU. Thus, the result of I/O path contention is an *effective* disk service time (the sum of seek, latency, transfer, and contention times) that is longer than the *intrinsic* disk service time (the sum of seek, latency, and transfer times). Disk busy times increase correspondingly, and so the effect of I/O path contention is reflected in the disk service demand parameter of the queueing network model, which is calculated as $D_{disk} = B_{disk}/C$ (C here is the number of system completions).

How can our canonical model be used to project performance for modified environments? The answer to this question is at once very simple and very complex. On the one hand, many postulated system and workload modifications can be represented by appropriate adjustments to the service demand parameters of the model. For example, the primary effect of a 50% CPU upgrade can be represented by dividing all CPU service demands by 1.5: a customer that required six seconds of service on a 2 MIPS (million instructions per second) CPU will require four seconds of service on a 3 MIPS CPU. Similarly, the primary effect of adding I/O paths and reallocating disks can be represented by reducing the disk service demands, because I/O path contention can be expected to decrease. Unfortunately, it is difficult to quantify the amount of this reduction. The purpose of the I/O modelling techniques to be discussed in this chapter is to allow the analyst to deal with parameters that are meaningful: channels, controllers, strings, paths, disks, intrinsic I/O service requirements, etc. These techniques serve to translate a modification expressed in terms of these parameters into an appropriate modification of the disk service demands.

Our study will progress by introducing ever greater levels of detail into our models. Before proceeding, two remarks:

- For concreteness we will use terminology derived from IBM systems in this chapter. The architectural characteristics that we address and the modelling techniques that we develop, however, are equally applicable to systems of other manufacturers.
- The fact that the computer system under study has a complex I/O subsystem, such as that illustrated in Figure 10.1, does *not* mean that sophisticated I/O subsystem modelling techniques are required. In undertaking any study, the analyst must think carefully about the questions under consideration. If the primary effects of the postulated

modifications can be represented by straightforward adjustments of disk service demands (or by no adjustment, as might be the case for a CPU upgrade), then sophisticated I/O subsystem modelling techniques are not called for.

10.2. Channel Contention in Non-RPS I/O Subsystems

In this section we develop a technique to represent the effect of channel contention in an I/O subsystem with disks that do not perform rotational position sensing (RPS). (RPS will be explained in the next section.) Customers cycle through such a system (illustrated in Figure 10.2) as follows:

- queue for the CPU
- when the CPU is available, use it
- queue for access to a specific disk
- when that disk is available, seek
- still holding the disk, queue for access to the channel (contention)
- when the channel is available, use both it and the disk to search for (latency) and transfer data.

Two preliminary remarks:

- In fact, momentary access to elements of the I/O path is required to initiate a disk seek. It is customary (and justified, based on experience) to ignore this in modelling disk I/O subsystems; we will do so throughout this chapter.
- Recall that topology is irrelevant in separable queueing networks; the crucial issue is our choice of service demands, not our placement of the channel relative to the disks in our figures.

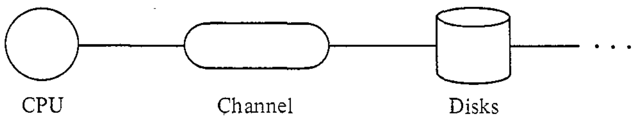

Figure 10.2 – A Highly Simplified I/O Subsystem

As noted in the previous section, it is a straightforward matter to construct a queueing network model of a non-RPS disk I/O subsystem that, given parameters derived from measurements over a specific interval, accurately reproduces the performance observed during that same interval. Each disk should be represented individually, with service demand equal to measured disk busy time divided by measured system completions (in the single class case). The relative contributions of seek, latency, transfer, and contention times are unimportant.

In using the model to project performance for modified environments, it may be necessary to adjust not only the intrinsic service demands at the disks (for example, the substitution of a disk with a higher data transfer rate would result in a smaller transfer time component), but also the channel contention component (this same substitution would result in a decrease in channel holding times, and thus in channel contention). Note that conducting such a modification analysis imposes two requirements beyond those imposed by validating a baseline model:

- It may be necessary to deduce the relative contributions of seek, latency, transfer, and contention times in the measured disk busy times.
- It may be necessary to estimate the changes in each of these components that will result from the proposed modifications.

The emphasis in this section, and in the chapter as a whole, is on the most interesting aspect of these requirements: we will develop techniques that, given information about the intrinsic service requirements of requests at each disk (the seek, latency, and transfer times), will estimate the contention times experienced by requests associated with the various disks, and thus the effective service demands at the disks. In developing our techniques, we will assume that seek, latency, and transfer times are known. (A later section will discuss how to deduce these values from typical measurement data.) In using these techniques to project performance for a modified environment, the analyst would adjust the intrinsic service demands (e.g., transfer times) directly, relying on the algorithms to estimate revised contention components, and thus revised effective service demands. (Chapter 13 discusses modification analysis in more detail.)

Although it is the effective service demand at each disk k, D_k, that we require, it will be convenient to think of D_k as the product of V_k, the number of visits to disk k made by a customer, and S_k, the effective service requirement per visit. S_k, in turn, can be thought of as the sum of $seek_k$, $latency_k$, $transfer_k$, and $contention_k$, each of which are expressed on a per-visit basis. In other words:

$$\begin{aligned} D_k &= V_k\ S_k \\ &= V_k \left[seek_k + latency_k + transfer_k + contention_k \right] \end{aligned}$$

We assume that all of these quantities except for $contention_k$ are known.

We must estimate $contention_k$, the time spent awaiting access to the channel by a request associated with disk k. In the spirit of mean value analysis, this can be viewed as the product of the channel holding time of a request associated with disk k and the number of requests encountered by a disk k request upon arrival at the channel. The channel holding time of a request associated with disk k is simply $latency_k + transfer_k$. To estimate the arrival instant channel queue length, we (falsely) view the channel as a center in an open system. Recall from Chapter 6 that the arrival instant queue length at any center in an open system is equal to $\frac{U}{1 - U}$ where U is the utilization of the center. In the present case, we know that any requests ahead of a disk k request at the the channel must be associated with some disk other than k, so we modify this equation to be:

$$\frac{U_{ch} - U_{ch}(k)}{1 - U_{ch}}$$

where U_{ch} is the utilization of the channel, and $U_{ch}(k)$ is the contribution to this utilization of requests associated with disk k. Thus, if we knew U_{ch} and $U_{ch}(k)$ we could estimate the effective service demand of disk k as:

$$\begin{aligned} D_k &= V_k \left[seek_k + latency_k + transfer_k + contention_k \right] \\ &= V_k \left[seek_k + latency_k + transfer_k + \right. \\ &\qquad \left. \left[latency_k + transfer_k \right] \times \frac{U_{ch} - U_{ch}(k)}{1 - U_{ch}} \right] \\ &= V_k \left[seek_k + \left[latency_k + transfer_k \right] \times \left[1 + \frac{U_{ch} - U_{ch}(k)}{1 - U_{ch}} \right] \right] \\ &= V_k \left[seek_k + \frac{(latency_k + transfer_k)\ (1 - U_{ch}(k))}{1 - U_{ch}} \right] \end{aligned}$$

Unfortunately, the various $U_{ch}(k)$ required to parameterize the model are known only after the model has been evaluated. This suggests the iterative scheme shown as Algorithm 10.1.

As an example, consider a batch computer system with an average multiprogramming level of 10, a CPU at which jobs have an average total

```
1.  Define a queueing network model of the system in which the
    I/O subsystem is represented only by the disks. Initially, as-
    sume that system throughput, $X$, is zero. (This will cause
    the contention component of the disks' effective service
    demands to be set to zero during the first iteration.)
2.  Iterate as follows:
    2.1. For each disk $k$, estimate the contribution to channel
         utilization of requests associated with that disk as:
             $U_{ch}(k) = X \, V_k \left[ latency_k + transfer_k \right]$
         where $X$ is obtained from the previous iteration.
    2.2. Estimate channel utilization: $U_{ch} = \sum_{all\ disks\ k} U_{ch}(k)$
    2.3. For each disk $k$, estimate its effective service demand
         as:
             $D_k = V_k \left[ seek_k + \frac{(latency_k + transfer_k)(1 - U_{ch}(k))}{1 - U_{ch}} \right]$
    2.4. Evaluate the queueing network model using MVA.
    Repeat Step 2 until successive estimates of system
    throughput, $X$, are sufficiently close.
3.  Obtain performance measures from the final iteration.
```

Algorithm 10.1 – Non-RPS Disks

service requirement of 15 seconds, a single channel, and five equally loaded non-RPS disks at each of which jobs have average total service requirements of 8 seconds seeking (i.e., $V_k seek_k = 8$), 1 second searching (latency), and 2 seconds transferring data. (Note that it is not necessary to descend to the "visit" level in order to apply Algorithm 10.1; we did so in our development for consistency with forthcoming sections.) We analyze this system using a queueing network with 10 customers and six service centers, corresponding to the CPU and the five disks. The service demand at the CPU is 15 seconds. The initial service demand at each disk is 11 seconds. (The equally loaded disks are not essential, but are used to simplify the example; they allow single calculations of $U_{ch}(k)$ and D_k to be used for all disks.) Table 10.1 displays the iteration.

The parameter values used in the first iteration correspond to an analysis in which channel contention is ignored. The results (throughput of .056, channel utilization of 84%) differ considerably from those

iter.	input	calculations			output
	X	$U_{ch}(k)$	U_{ch}	D_k	X
1	.0000	.000	.000	11.00	.0557
2	.0557	.167	.836	23.24	.0299
3	.0299	.090	.449	12.96	.0499
4	.0499	.150	.749	18.16	.0376
5	.0376	.113	.564	14.10	.0467
6	.0467	.140	.701	16.63	.0408
7	.0408	.122	.611	14.77	.0449
8	.0449	.135	.674	15.96	.0423
9	.0423	.127	.635	15.18	.0439
10	.0439	.132	.659	15.63	.0430
11	.0430	.129	.645	15.36	.0434
12	.0434	.130	.651	15.48	.0434

Table 10.1 – Execution of Algorithm 10.1

obtained at the end of the iteration (throughput of .044, channel utilization of 65%), when channel contention has been accounted for.

Algorithm 10.1 can be applied to computer systems with multiple channels, each connecting the CPU to a specific set of disks. Each channel subsystem must be considered separately in the algorithm. In Steps 2.1 and 2.2, a separate utilization is calculated for each channel. In Step 2.3, the effective service demand of each disk is estimated using the utilization of the channel to which it is attached.

Two simple modifications are required to generalize the algorithm to multiple class queueing networks. In Step 2.1 the channel utilization due to requests associated with disk k must be estimated as:

$$U_{ch}(k) \;=\; \sum_{c=1}^{C} \left[X_c \, V_{c,k} \left[latency_{c,k} \;+\; transfer_{c,k} \right] \right]$$

In Step 2.3 revised effective service demands must be estimated on a per-class basis as:

$$D_{c,k} \;=\; V_{c,k} \left[seek_{c,k} \;+\; \frac{(latency_{c,k} \;+\; transfer_{c,k})\;(1 \;-\; U_{ch}(k))}{1 \;-\; U_{ch}} \right]$$

Algorithm 10.1 is simple, efficient, and sufficiently accurate. Further, the situations in which its accuracy might be questioned are easily identified: those in which the utilization of the channel is high – certainly greater than 50%, a higher utilization than would be encountered in most applications. The source of this error is our view of the channel as a center in an open system, which we used in calculating the expected queue length encountered at the channel by arriving requests from disk

k. In reality, the number of requests queued at the channel is bounded, rather than unbounded as implied by the open system approximation. For a given utilization, a service center in an open queueing network will have a greater queue length than a service center in a closed network. The open system approximation therefore will tend to overestimate the queue length at the channel, and thus to overestimate channel residence times.

10.3. Channel Contention in RPS I/O Subsystems

Rotational position sensing (*RPS*) increases concurrency in the I/O subsystem by allowing disks to search for data (the latency period) independently of each other and of the channel. When the data record of interest rotates under the heads, the disk attempts to *reconnect*. (*Re*connect rather than *connect* because momentary access to elements of the I/O path was required to initiate the seek and the search; we shall continue to ignore this in our models.) If the path is free, this reconnect succeeds and the data transfer takes place. If not, another reconnect is attempted when the data next rotates under the heads, one disk revolution later. Reconnect attempts are continued in this manner until success is achieved. We refer to all reconnect attempts after the first as *retries*.

As in the previous section, we wish to estimate the effective service demand for each disk k:

$$D_k = V_k \left[seek_k + latency_k + transfer_k + contention_k \right]$$

We assume that all of these quantities except for $contention_k$ are known. In the case of RPS disks, we have:

$$contention_k = retries_k \times rotation_k$$

where $retries_k$ is the number of retries required by disk k before a successful reconnect, on average, and $rotation_k$ is the rotation time of disk k. The latter quantity is known from device characteristics; our objective thus is to estimate $retries_k$.

We assume that for any particular disk k, the probabilities of failure on various reconnect attempts are independent. (This assumption is not strictly correct, but at most a small error is introduced.) We let $P_k[reconnect\ fails]$ denote this probability of failure. Then:

$$
\begin{aligned}
retries_k &= 0 \times (1 - P_k[reconnect\ fails]) + \\
&\quad 1 \times (1 - P_k[reconnect\ fails]) \times P_k[reconnect\ fails] + \\
&\quad 2 \times (1 - P_k[reconnect\ fails]) \times (P_k[reconnect\ fails])^2 + \\
&\quad \vdots \\
&= \sum_{i=1}^{\infty} \left[i\ (1 - P_k[reconnect\ fails]) \times (P_k[reconnect\ fails])^i \right] \\
&= \frac{P_k[reconnect\ fails]}{1 - P_k[reconnect\ fails]} \qquad \text{(a standard transformation)}
\end{aligned}
$$

A reconnect attempt succeeds if the path back to the CPU is free, and fails otherwise. In other words, $P_k[reconnect\ fails]$ is equal to $P_k[path\ busy]$, the probability that disk k finds the path busy when it attempts to reconnect. Presently the channel is the only path element that we are considering, so $P_k[path\ busy]$ is equal to $P_k[channel\ busy]$, the probability that disk k finds the channel busy when it attempts to reconnect. At first glance, we might guess that $P_k[channel\ busy]$ is equal to U_{ch}. In fact, though, disk k will not "see" its own contribution to channel utilization. Thus:

$$
\begin{aligned}
&P_k[reconnect\ fails] \\
&\quad = P_k[path\ busy] \\
&\quad = P_k[channel\ busy] \\
&\quad = P[channel\ busy \mid disk\ k\ not\ transferring] \\
&\quad = \frac{P[channel\ busy\ \&\ disk\ k\ not\ transferring]}{P[disk\ k\ not\ transferring]} \qquad \text{(Bayes's rule)} \\
&\quad = \frac{U_{ch} - U_{ch}(k)}{1 - U_k(transfer)}
\end{aligned}
$$

where $U_k(transfer)$ is the utilization of disk k due to data transfers. This quantity is equal to the utilization of the channel due to requests associated with disk k, $U_{ch}(k)$. Making this substitution and using the result in the expression for $retries_k$, we obtain:

$$
retries_k = \frac{U_{ch} - U_{ch}(k)}{1 - U_{ch}}
$$

Since these utilizations are known only once the model has been evaluated, we employ an iterative scheme, shown in Algorithm 10.2.

```
1. Define a queueing network model of the system in which the
   I/O subsystem is represented only by the disks. Initially, as-
   sume that system throughput, $X$, is zero. (This will cause
   the contention component of the disks' effective service
   demands to be set to zero during the first iteration.)
2. Iterate as follows:
   2.1. For each disk $k$, estimate the contribution to channel
        utilization of requests associated with that disk as:
            $U_{ch}(k) = X \, V_k \, transfer_k$
        where $X$ is obtained from the previous iteration.
   2.2. Estimate channel utilization: $U_{ch} = \sum_{all\ disks\ k} U_{ch}(k)$
   2.3. For each disk $k$:
        – Estimate the average number of retries required
          before a successful reconnect as:
            $retries_k = \frac{U_{ch} - U_{ch}(k)}{1 - U_{ch}}$
        – Estimate an effective service demand as:
            $D_k = V_k \left[ seek_k + latency_k + transfer_k + (retries_k \times rotation_k) \right]$
   2.4. Evaluate the queueing network model.
   Repeat Step 2 until successive estimates of system
   throughput, $X$, are sufficiently close.
3. Obtain performance measures from the final iteration.
```

Algorithm 10.2 – RPS Disks

As an example we return to the system considered in Section 10.2, but assume that the disks are capable of rotational position sensing. Let the rotation time of each disk be 17 msec., and let the number of operations per disk be 120. Table 10.2 displays the iteration. In comparison to the non-RPS case, we note that system throughput has increased by 17% while channel utilization has decreased by 23%.

iter.	input	calculations				output
	X	$U_{ch}(k)$	U_{ch}	$retries_k$	D_k	X
1	.0000	.000	.000	.000	11.00	.0557
2	.0557	.111	.557	1.006	13.05	.0496
3	.0496	.099	.496	.788	12.61	.0509
4	.0509	.102	.509	.830	12.69	.0507
5	.0507	.101	.507	.822	12.68	.0507

Table 10.2 — Execution of Algorithm 10.2

Like its non-RPS predecessor, this algorithm can be applied to computer systems with multiple channels each connecting the CPU to a specific set of disks, by considering each channel subsystem separately in Steps 2.1 to 2.3. It also can be generalized to multiple classes by means of two simple modifications. The equation in Step 2.1 becomes:

$$U_{ch}(k) = \sum_{c=1}^{C} \left[X_c \, V_{c,k} \, transfer_{c,k} \right]$$

and the second equation in Step 2.3 becomes:

$$D_{c,k} = V_{c,k} \Big[seek_{c,k} + latency_{c,k} + transfer_{c,k} + (retries_k \times rotation_k) \Big]$$

(The rotation time of the disk and the average number of retries required before a successful reconnect are independent of the customer class.)

10.4. Additional Path Elements

The path between the CPU and a disk in a contemporary I/O subsystem contains several elements in addition to a channel. The contention component of the effective disk service demands is influenced by each of these path elements. Algorithm 10.2 estimates only the channel's contribution to the contention component. This algorithm can be used in modelling I/O subsystems with additional path elements, provided that a change in the channel's contribution will be the primary effect on the contention component of any contemplated modification. If this is not the case — if significant variations in the contributions to the contention component of other path elements are anticipated — then the algorithm must be extended to estimate these contributions. Such extensions are the subject of the present section.

10.4.1. Controllers

Figure 10.3 illustrates the interposition of a *controller* on the path between the CPU and a disk. Several controllers are attached to a channel, and several disks are attached to a controller. A controller is occupied when any of its associated disks are transferring data.

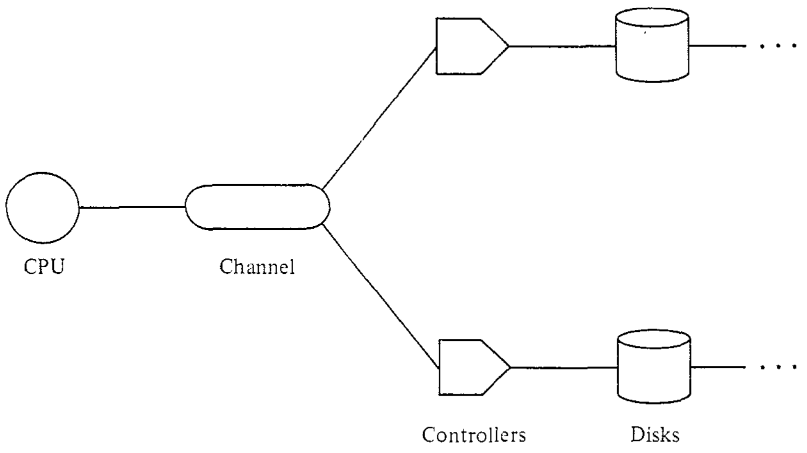

Figure 10.3 – Controllers

As in Section 10.3, our objective is to estimate D_k for each disk k. This requires that we estimate $contention_k$. To do so, we must estimate $retries_k$. This, in turn, requires that we estimate $P_k[reconnect\ fails]$, which is equal to $P_k[path\ busy]$. This quantity can be expressed as:

$$P_k[path\ busy] = P_k[controller\ busy] + P_k[controller\ free\ \&\ channel\ busy]$$

By analogy to the derivation in the previous section, the probability that disk k finds its controller busy when attempting to reconnect is:

$$P_k[controller\ busy] = \frac{U_{ctlr} - U_{ctlr}(k)}{1 - U_k(transfer)}$$

The probability that disk k finds its controller free and its channel busy when attempting to reconnect is:

$$
\begin{aligned}
&P_k[\mathit{controller\ free\ \&\ channel\ busy}]\\
&\quad = P[\mathit{controller\ free\ \&\ channel\ busy} \mid \mathit{disk\ k\ not\ transferring}]\\
&\quad = \frac{P[\mathit{controller\ free\ \&\ channel\ busy\ \&\ disk\ k\ not\ transferring}]}{P[\mathit{disk\ k\ not\ transferring}]}\\
&\quad = \frac{U_{ch} - U_{ch}(\mathit{ctlr})}{1 - U_k(\mathit{transfer})}
\end{aligned}
$$

(In a generalization of our earlier notation, $U_{ch}(\mathit{ctlr})$ is the utilization of the channel by requests associated with the controller to which disk k is attached.) To make our notation more compact, we replace $U_{ctlr} - U_{ctlr}(k)$, which is the utilization of the controller due to requests associated with disks other than k, by $U_{ctlr}(\bar{k})$. Similarly, we replace $U_{ch} - U_{ch}(\mathit{ctlr})$, which is the utilization of the channel due to requests routed through controllers other than the one of interest, with $U_{ch}(\overline{\mathit{ctlr}})$. We obtain:

$$P_k[\mathit{path\ busy}] = \frac{U_{ctlr}(\bar{k}) + U_{ch}(\overline{\mathit{ctlr}})}{1 - U_k(\mathit{transfer})}$$

and:

$$\mathit{retries}_k = \frac{U_{ctlr}(\bar{k}) + U_{ch}(\overline{\mathit{ctlr}})}{1 - U_{ch}}$$

An iterative solution can be obtained, in a manner analogous to Algorithm 10.2.

10.4.2. Heads of String

Some architectures introduce one further path element: a collection of disks constitutes a *string*, which is connected to a controller through a *head of string* (*hos*). Figure 10.4 illustrates this situation.

Like the controller and the channel, the head of string is occupied when any of its associated disks are transferring data. Thus:

$$
\begin{aligned}
P_k[\mathit{path\ busy}] = {} & P_k[\mathit{hos\ busy}] + \\
& P_k[\mathit{hos\ free\ \&\ controller\ busy}] + \\
& P_k[\mathit{hos\ free\ \&\ controller\ free\ \&\ channel\ busy}]
\end{aligned}
$$

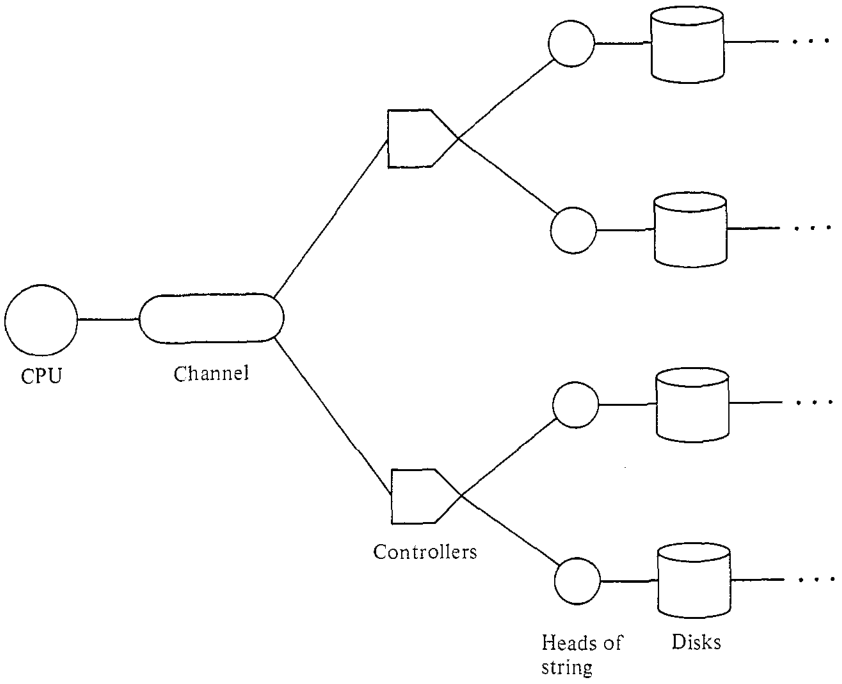

Figure 10.4 − Heads of String

Evaluating these terms yields:

$$P_k[\mathit{hos\ busy}] = \frac{U_{hos}(\bar{k})}{1 - U_k(\mathit{transfer})}$$

$$P_k[\mathit{hos\ free\ \&\ controller\ busy}] = \frac{U_{ctlr}(\overline{hos})}{1 - U_k(\mathit{transfer})}$$

$$P_k[\mathit{hos\ free\ \&\ controller\ free\ \&\ channel\ busy}] = \frac{U_{ch}(\overline{ctlr})}{1 - U_k(\mathit{transfer})}$$

As a result:

$$P_k[\mathit{path\ busy}] = \frac{U_{hos}(\bar{k}) + U_{ctlr}(\overline{hos}) + U_{ch}(\overline{ctlr})}{1 - U_k(\mathit{transfer})}$$

and:

$$\mathit{retries}_k = \frac{U_{hos}(\bar{k}) + U_{ctlr}(\overline{hos}) + U_{ch}(\overline{ctlr})}{1 - U_{ch}}$$

10.5. Multipathing

The architectures just described are *single path* architectures: each disk is connected to a single head of string, each head of string to a single controller, and each controller to a single channel, with the result that there is only one path from the CPU to any disk — a particular channel, controller, and head of string must be used. This imposes limitations in several respects:

- *reliability* — The failure of any path element will cause all disks "beneath it" to become inaccessible.
- *performance* — A disk may be unable to transfer data because, for example, although its head of string and its controller are free, its channel is busy transferring data for another disk associated with a different controller. There is no way to utilize another channel that may be free at the time.
- *sharing* — In a single path architecture it is not possible to organize several CPUs as a *loosely-coupled multiprocessor* coordinated by means of shared I/O devices.

Multipathing attempts to overcome these limitations. Figure 10.1 in the introduction to this chapter illustrates a multipathing I/O subsystem. In general, a disk may be connected to several heads of string, a head of string to several controllers, and a controller to several channels, perhaps attached to different CPUs. Each different combination of {channel, controller, head of string} that can be used to access a particular disk constitutes a unique path. The system includes an algorithm that selects a path for each data transfer. Existing algorithms fall into two general classes. In *static reconnection* algorithms, any free path is used to initiate an I/O sequence, but the disk must reconnect over this same path to transfer data. In *dynamic reconnection* algorithms, the reconnect may occur over any free path. (Interestingly, multipathing with static reconnection typically results in a performance *degradation* relative to the single path case, which is tolerated for the sake of reliability and sharing.)

In modelling multipathing, our basic approach remains unchanged, but the process of estimating the probabilities of reconnect failure for the various disks (the P_k[*reconnect fails*]) becomes more involved. Three factors contribute to this complexity:

- To estimate the utilizations of the various path elements, the path selection algorithm must be considered, because at any "level" of the I/O subsystem hierarchy (i.e., at the level of the channels, the controllers, or the heads of string) the utilization due to requests associated with a particular disk is divided among several path elements in a manner determined by this algorithm. This problem is discussed in Section 10.5.1.

- Once the utilizations of the various path elements are known, it still is not straightforward to estimate the probability of reconnect failure for a particular disk. This is the case because several paths are available to each disk. The probability that each of these paths is found busy must be estimated. Then, the path selection algorithm must be considered to determine probability of reconnect failure given these path busy probabilities. This problem is discussed in Section 10.5.2.
- In the expression for the probability that a particular disk finds a particular path busy, additional terms must be introduced due to multipathing. This problem is discussed in Section 10.5.3.

Algorithm 10.3 shows the general structure of a technique for representing multipathing in queueing network models.

10.5.1. Estimating the Utilizations of Path Elements

In a single path architecture, the utilization of any particular path element (any channel, controller, or head of string) is equal to the sum of the data transfer utilizations of all disks "beneath it". In the case of multipathing, though, it may be possible to route the data transfers of any particular disk through several different {channel, controller, head of string} paths. Thus, the utilization of any path element is the sum of portions of the data transfer utilizations of a number of disks.

Even if we "know" the utilization of each disk due to data transfer (an improved estimate is obtained each time we iterate through all of Step 2 of Algorithm 10.3), the proportion routed through each path element can be estimated only once we have represented the behavior of the path selection algorithm. And, in order to represent the behavior of the path selection algorithm, we must know the utilizations of the path elements, because the path selection algorithm is driven by the probabilities that the various paths are found busy. In other words, estimating the utilizations of path elements, Step 2.2 of Algorithm 10.3, itself is an iterative process.

This iterative process would be relatively straightforward if I/O subsystems were fully interconnected — if every disk could use every head of string, controller, and channel. Unfortunately this is not the case. Both physical and logical constraints exist. These constraints could turn the estimation of the utilizations of path elements into a nasty combinatorial problem. Fortunately, though, interconnection structures tend to be quite limited and quite regular in practice, and various simplifying approximations can be introduced without significant loss of accuracy.

One possible approach (there are several) is suggested by the fact that in handling I/O operations for any particular disk k, the path selection

1. Define a queueing network model of the system in which the I/O subsystem is represented only by the disks. Make an initial estimate of system throughput, X.
2. Iterate as follows:
   - 2.1. Estimate the utilization of each disk k due to data transfer:

     $$U_k(\mathit{transfer}) = X\ V_k\ \mathit{transfer}_k$$

   - 2.2. Estimate the utilizations of the various path elements, by apportioning the data transfer utilizations of the disks among these path elements in a way that is consistent with the system's path structure and with the path selection algorithm. (See Section 10.5.1.)
   - 2.3. Estimate the effective service demand of each disk k:
     - For each path that can be used by disk k, estimate $P_k[\mathit{path\ busy}]$, the probability that disk k finds this path busy when it attempts to reconnect. (See Section 10.5.2.)
     - Considering these probabilities along with the system's path selection algorithm, estimate $P_k[\mathit{reconnect\ fails}]$, the probability that disk k fails to reconnect. (See Section 10.5.3.)
     - Given this probability, estimate $\mathit{retries}_k$ and D_k in the usual manner:

     $$\mathit{retries}_k = \frac{P_k[\mathit{reconnect\ fails}]}{1 - P_k[\mathit{reconnect\ fails}]}$$

     $$D_k = V_k\Big[\mathit{seek}_k + \mathit{latency}_k + \mathit{transfer}_k + (\mathit{retries}_k \times \mathit{rotation}_k)\Big]$$

   - 2.4. Evaluate the queueing network model.

   Repeat Step 2 until successive estimates of system throughput, X, are sufficiently close.
3. Obtain performance measures from the final iteration.

Algorithm 10.3 — Multipathing in the Rough

algorithm will choose among the possible paths in proportion to the probability that it finds them free. Thus:

- Establish initial estimates (say, zero) for the utilization of each path element.
- Iterate as follows:
  - Treat each disk k in turn:
    - For each path i to disk k, let $P_k[\textit{path } i \textit{ selected}]$ denote the proportion of disk k's transfers that use path i. Set the $P_k[\textit{path } i \textit{ selected}]$ to be proportional to the probabilities that disk k finds each path i free: $(1 - P_k[\textit{path } i \textit{ busy}])$, where $P_k[\textit{path } i \textit{ busy}]$ is calculated as in Section 10.5.2, using the current estimates for path element utilizations.
    - Update the estimates of the utilizations of the various path elements to include the new assignment of disk k's transfers.

  Once each disk has been considered, iterate, modifying previous values.

This procedure will not reproduce exactly the behavior of the path selection algorithm, but will provide a reasonable approximation.

10.5.2. Estimating the Path Busy Probabilities

As in the case of single path architectures, the probability that disk k finds any particular path busy when it attempts to reconnect is:

$$\begin{aligned} P_k[\textit{path busy}] &= P_k[\textit{hos busy}] + \\ &\quad P_k[\textit{hos free \& controller busy}] + \\ &\quad P_k[\textit{hos free \& controller free \& channel busy}] \end{aligned}$$

where *hos*, *controller*, and *channel* refer to the particular head of string, controller, and channel of interest — those that constitute the path in question.

In the multipathing case, additional terms are involved in expressing these probabilities in terms of the utilizations of path elements. The probability that disk k finds the path's head of string busy is unchanged:

$$P_k[\textit{hos busy}] = \frac{U_{hos} - U_{hos}(k)}{1 - U_k(\textit{transfer})}$$

The probability that disk k finds the path's head of string free but its controller busy has one additional term:

$$P_k[\textit{hos free \& controller busy}] = \frac{U_{ctlr} - U_{ctlr}(hos) - U_{ctlr}(k \rightarrow \overline{hos})}{1 - U_k(\textit{transfer})}$$

where $U_{ctlr}(k \rightarrow \overline{hos})$ is the utilization of the controller of interest due to requests associated with disk k routed through heads of string other than the one of interest. The probability that disk k finds the path's head of string and controller free but its channel busy has two additional terms:

$$P_k[\mathit{hos\ free\ \&\ controller\ free\ \&\ channel\ busy}]$$
$$= \frac{U_{ch} - U_{ch}(ctlr) - U_{ch}(hos \rightarrow \overline{ctlr}) - U_{ch}(k \rightarrow \overline{hos} \rightarrow \overline{ctlr})}{1 - U_k(transfer)}$$

where $U_{ch}(hos \rightarrow \overline{ctlr})$ is the utilization of the channel of interest due to requests routed through the head of string of interest but through controllers other than the one of interest, and $U_{ch}(k \rightarrow \overline{hos} \rightarrow \overline{ctlr})$ is the utilization of the channel of interest due to requests associated with disk k routed through heads of string and controllers other than the ones of interest.

10.5.3. Estimating the Probability of Reconnect Failure

In a single path architecture, $P_k[\mathit{reconnect\ fails}]$ is equal to $P_k[\mathit{path\ busy}]$. This simple relationship does not hold in the case of multipathing. Each disk k now has a number of paths to choose from. In determining the probability of reconnect failure, the busy probabilities of each possible path must be considered, along with the strategy used by the path selection algorithm.

With a static reconnection algorithm, the reconnection is attempted over whichever path was chosen for the initiation of the I/O sequence. Thus:

$$P_k[\mathit{reconnect\ fails}] = \sum_{i \in \substack{possible \\ paths}} P_k[\mathit{path\ i\ selected}] \times P_k[\mathit{path\ i\ busy}]$$

where $P_k[\mathit{path\ i\ selected}]$ is the proportion of disk k transfers that use path i (from Section 10.5.1) and $P_k[\mathit{path\ i\ busy}]$ is the probability that disk k finds path i busy when attempting to reconnect (from Section 10.5.2).

With a dynamic reconnection algorithm, the reconnection can take place over any free path. Thus:

$$P_k[\mathit{reconnect\ fails}] = P_k[\mathit{all\ possible\ paths\ busy}]$$
$$\approx \prod_{i \in \substack{possible \\ paths}} P_k[\mathit{path\ i\ busy}]$$

(This equation assumes that the probabilities of various paths being busy are independent of one another. This assumption is not strictly correct, but any error introduced is apt not to be substantial.)

10.6. Other Architectural Characteristics

In this section we provide brief treatments of two additional architectural characteristics: shared disks and cached devices.

10.6.1. Shared Disks

As noted in Section 10.5, one virtue of multipathing is that it allows disks to be shared among several systems. Such a configuration often is referred to as a *loosely-coupled multiprocessor.* In principle the systems could be joined at any level in the I/O subsystem hierarchy. Figure 10.5 illustrates a typical case, in which a single controller is attached to two channels connected to different CPUs.

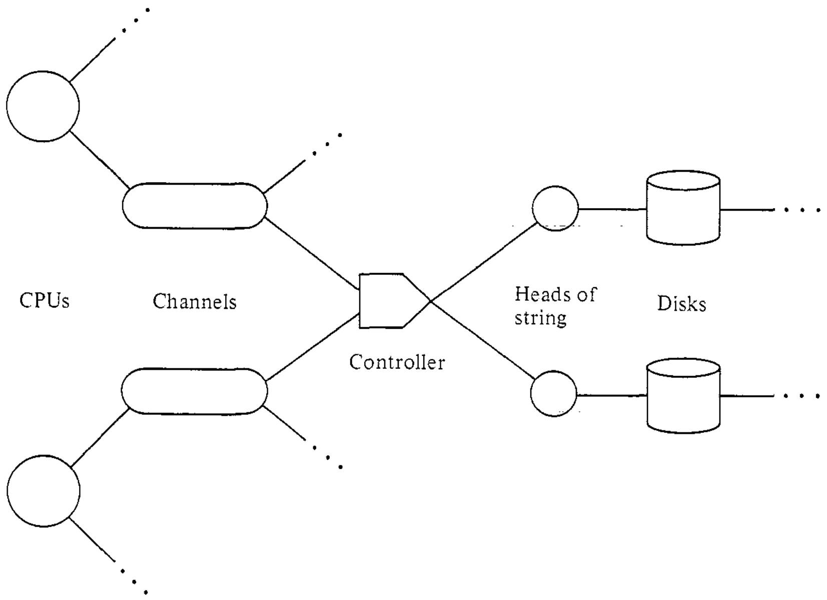

Figure 10.5 — Shared Disks

A loosely-coupled multiprocessor can be viewed in two ways: as a single system that happens to have multiple CPUs, or as a collection of separate systems that happen to share disks. The distinction is important, for the two views lead to different modelling approaches. The choice of view depends upon the way in which a particular processing complex actually is used, and the nature of the performance questions under consideration.

The first view, that of a single system that happens to have multiple CPUs, leads to a single large queueing network model that includes all devices and all workload components. The advantage of this modelling approach is its conceptual simplicity: no new ideas are involved. For this reason we will discuss this view no further.

The second view, that of a collection of separate systems that happen to share disks, leads to a collection of small queueing network models, one corresponding to each system. The advantage of this modelling approach is its modularity: a modification whose primary effect will be felt by one system can be investigated by defining, parameterizing, and evaluating one relatively small model. Conducting such an analysis is the subject of the remainder of this subsection.

Consider the queueing network model of any of the systems. The I/O subsystem component of this model will include service centers corresponding to all disks used by customers on that system, whether those disks are dedicated or shared. Certainly, contention in the I/O subsystem due to requests associated with other systems must be represented. If not, throughput of requests associated with the system of interest would be over-estimated. We will represent this contention in our model, but will do so in a way that is determined from measurement data. In modifying the model for purposes of performance projection, we will assume that the utilizations of disks and path elements due to requests associated with other systems remain unchanged.

In estimating the effective service demand at each disk in the model, we represent the effect of requests associated with other systems in two ways:

- *accounting for additional reconnect delay experienced because of path contention due to "foreign" requests* — In evaluating the expressions for the probabilities that various paths are found busy, the measured utilizations due to requests associated with other systems are added to the calculated utilizations due to customers in the model, for each shared path element and shared disk. This adjustment results in a realistic estimate for the contention component of effective service demand.
- *accounting for delay in acquiring the disk due to its use by "foreign" requests* — For each disk, the contention component calculated above is added to the seek, latency, and transfer components. This total is divided by one minus the measured utilization of the disk due to requests associated with other systems. The rationale is the same used in estimating channel contention for non-RPS disks (Section 10.2).

We recommend this approach whenever it is possible to assume relative stability in the utilizations of disks and path elements due to requests associated with other systems, in the presence of postulated modifications to the system of interest.

10.6.2. Cached Devices

A *cache memory* is a relatively small, relatively high speed memory that is used as a staging area for data. For many years cache memories have been interposed between processors and their primary memories. Very recently they have been introduced into I/O subsystems, typically by augmenting controllers with storage capacity (on the order of millions of bytes) and processing capacity. In this subsection we will take a brief look at modelling cached devices.

The cache contains duplicate copies of some of the disk-resident data. If the cache is well managed, the vast majority of the data that is referenced by I/O operations will be resident in the cache. Two parameters are crucial in determining the effectiveness of the cache. The first is the *hit ratio*: the proportion of I/O operations that refer to data residing in the cache. The second is the *read ratio*: the proportion of I/O operations that are reads rather than writes. These parameters are crucial because a *read hit* (a read operation referencing data resident in the cache) can be serviced without accessing the disk. Thus, it has a service time roughly equal to the data transfer time, with no seek or latency components. On the other hand, a *read miss*, a *write hit*, and a *write miss* each require that the disk be accessed. Furthermore, because of the overhead involved in managing the cache, a disk access in a cached environment is somewhat slower than a disk access in a conventional environment. Thus, a performance *degradation* can result from conversion to a cached I/O subsystem if a low read ratio exists, regardless of the hit ratio. A performance improvement will result if high hit and read ratios exist.

Let us consider a modelling study whose objective is to estimate the effect of converting an existing system to a cached I/O subsystem. We adhere to the basic model structure and evaluation techniques used in previous sections, and assume that a validated baseline model exists.

- We can reflect any changes in the seek, latency, and transfer times due to device characteristics in a straightforward manner.
- To account for the fact that a read hit can be serviced with no disk access, we adjust the effective service demands of the disks in the obvious way:

$$D_k = V_k \Big[(1 - (\mathit{hit\ ratio} \times \mathit{read\ ratio})) \times (\mathit{seek}_k + \mathit{latency}_k) + \mathit{transfer}_k + \mathit{contention}_k \Big]$$

  The hit ratio is not apt to be site-dependent in a significant way, so typical values can be obtained from manufacturer's data. The read ratio is not apt to change as a result of the conversion, so measurement data from the existing system can be used.

- The overhead of managing the cache may cause the utilizations to increase at various path elements, especially controllers. These increased utilizations should be represented, because they will affect path contention. Manufacturer's data is available that provides multiplicative factors to be used in estimating this overhead, given the basic transfer time. These factors can be used within the model in calculating the path busy probabilities.

10.7. Practical Considerations

Two practical considerations immediately arise in contemplating the application of the techniques we have described:

- How can the relatively detailed parameters required by these techniques be inferred from the measurement data that typically is encountered?
- How can these techniques be embedded in queueing network modelling software?

These related concerns are the subjects of the present section.

10.7.1. Inferring Parameter Values from Measurement Data

The techniques we have presented require that the following information be provided as input:

- a specification of the path structure of the I/O subsystem
- for each disk:
  - the visit count
  - the average seek, latency, and transfer times per visit
  - the average rotation time

Given this information, these techniques iteratively estimate the average contention time per visit at each disk, and thus the effective service time per visit, S_k, and the effective service demand, D_k.

In this section we consider the common situation in which the values of some of these parameters are not available directly, so must be inferred before our techniques can be applied. Inevitably, the visit counts and utilizations of the disks are known from measurement data. From these, the *actual* effective service times per visit and effective service demands can be calculated. We know that the actual effective service demands, if used to parameterize a model, would yield excellent results without the use of the techniques described in this chapter. (These techniques are required to conduct a modification analysis in which a change to the contention component of the effective service demands is anticipated to be a primary effect.) A fruitful way to view our task is that we must partition the actual effective service times per visit into seek, latency, transfer, and contention components, in such a way that when the seek, latency and transfer components are provided as inputs to the model (along with path structure, visit counts, and rotation times), the techniques that we have developed will calculate effective service times per visit and effective service demands that are roughly the same as the actual values. Once this has been achieved, we will consider the baseline model to be validated and will be prepared to use it for performance projection.

We denote the actual effective service time per visit at disk k by S_k^*, and the actual effective service demand by D_k^*. We proceed as follows:

- To estimate $latency_k$, we refer to the device characteristics.
- To estimate $transfer_k$ we employ the utilizations and visit counts of the channels, which are available readily from measurement data. From these, the service time per visit to each channel can be obtained. In the single path case, we set $transfer_k$ to this value (for the appropriate channel, of course). In the multipathing case, we take an average of the values of the channels accessible from disk k. Estimating $transfer_k$ on the basis of measured channel service times is important. The various path elements are processors rather than wires, and overhead is associated with each transfer. Estimating $transfer_k$ by considering block sizes and transfer rates would ignore this overhead, yielding an optimistic value. In stating our approach, we have made the homogeneity assumption that the data transfer service requirements of all disks on a particular channel are the same. Adjustments are possible if block size information is available.
- To estimate $seek_k$ it is tempting to refer to the device characteristics. Unfortunately, this approach is notoriously unreliable. We know that:

  $$seek_k + contention_k = S_k^* - latency_k - transfer_k$$

  where each of the quantities on the right hand side is known. In order to obtain consistent estimates for the two quantities on the left hand side, we will evaluate the queueing network, using either Algorithm 10.2 (for the single path case, augmented as in Section 10.4) or Algorithm 10.3 (for the multipathing case), and let the results determine the estimates. More specifically:

  - In Step 2.1 of either algorithm, we use the values of $transfer_k$ estimated above.
  - In each iteration of Step 2 in either algorithm, we use D_k^* as the effective service demand of disk k. (Fixing this value does not entirely eliminate iteration, because the throughput of the model will differ slightly from the throughput of the system.)

– When the algorithm terminates, it will have estimated P_k[*reconnect fails*] and $retries_k$ for each disk. Since $rotation_k$ is known (from the device characteristics), this means that an estimate for $contention_k$ has been obtained. We set our estimate for $seek_k$ to:

$$seek_k \ = \ S_k^* - latency_k - transfer_k - contention_k$$

We now are prepared to use the model for performance projection.

10.7.2. Incorporation in Queueing Network Modelling Software

The preceding discussion provides a number of insights concerning the support that a queueing network analysis software package might provide for modelling complex I/O subsystems.

The package might provide a convenient syntax for specifying the path structure of the I/O subsystem. As input, the analyst would provide this path structure, plus the effective service demands and visit counts at each disk, and the service demands and visit counts at each channel. The package might make use of internal information concerning various device types to provide quantities such as average latency and rotation times.

The analyst would indicate when the model has been specified fully. At this point, the package would evaluate the model, inferring the detailed parameter values and storing them internally.

At this point, it is possible to undertake modification analyses. The package might support this process in a number of ways. For example, the path structure might be modifiable using the same syntax in which it was specified, with the package adjusting the detailed parameter values.

Chapter 16 contains a more extensive discussion of software support for queueing network modelling.

10.8. Summary

In this chapter we have presented a single model structure that can be used to represent complex contemporary I/O subsystems at varying levels of detail. In this model structure, the I/O subsystem is represented by service centers corresponding to the various disks, each with an *effective service demand*, D_k, equal to:

$$V_k \left[seek_k + latency_k + transfer_k + contention_k \right]$$

We have developed algorithms for estimating the contention component of the effective service demand under a number of different assumptions about the structure of the I/O subsystem and the level of detail of the model. We have discussed various practical considerations, such as obtaining the necessary parameters for these algorithms from typical measurement data and incorporating these algorithms in queueing network modelling software.

For a variety of reasons the material in this chapter should not be viewed as definitive: the I/O subsystem architectures of various vendors differ substantially in their details, these architectures are evolving rapidly, and techniques for representing these architectures in queueing network models are an area of current research activity. Our algorithms should be viewed as an indication of what can be done, and as a set of techniques that can be used directly and also can be tailored as necessary to the requirements of specific systems.

In closing this chapter, we reiterate an important point made in its introduction. The fact that the computer system under study has a complex I/O subsystem does *not* mean that sophisticated I/O subsystem modelling techniques are required. If the primary effects of the postulated modifications can be represented by straightforward adjustments of disk service demands, then sophisticated I/O subsystem modelling techniques are not called for. The benefits of omitting sophistication include a simpler parameterization and fewer assumptions.

10.9. References

In this chapter we have developed models in which service centers of the load-independent queueing type are used to represent each disk, iteratively estimating the effective service demands at these centers. Two equally reasonable alternate approaches exist. The first of these can be described as follows:

- Define a queueing network model of the system in which the I/O subsystem is represented only by the disks, and each disk is represented by a service center of the delay type.
- Iterate as follows:
  - For each disk:
    - Estimate the effective service demand.
    - Use this value in a formula from queueing theory to estimate the average residence time at the disk.
    - Substitute this value into the corresponding delay center.
  - Evaluate the queueing network model.

  Repeat until successive estimates of system throughput are sufficiently close.

This approach is common in practice. Although its origins are unknown, it has been used by Bard [1980, 1982], by Wilhelm [1977], and by Zahorjan, Hume, and Sevcik [Zahorjan et al. 1978].

The second alternate approach, due to Brandwajn [1981], involves multiple applications of the principles of flow equivalence and hierarchical modelling described in Chapter 8:

- Consider each string (the disks attached to a particular head of string) in turn. Define an FESC by evaluating (for each feasible population) a submodel in which each disk on the string is represented by a service center of load-independent queueing type.
- Consider each controller subsystem (the heads of string and disks attached to a particular controller) in turn. Define an FESC by evaluating (for each feasible population) a submodel in which each string is represented by the FESC defined in the previous step.
- Consider each channel subsystem (the controllers, heads of string, and disks attached to a particular channel) in turn. Define an FESC by evaluating (for each feasible population) a submodel in which each controller subsystem is represented by the FESC defined in the previous step.
- Evaluate a high-level model consisting of the CPU and the channel subsystem FESCs defined in the previous step.

Two of the three model structures described above, including the one adopted in this chapter, require that effective service demands be estimated for each disk. The treatment of non-RPS disks (Section 10.2) belongs to the folklore of queueing network modelling. The treatment of RPS disks (Section 10.3) also is difficult to attribute. Wilhelm [1977] and Zahorjan, Hume, and Sevcik [Zahorjan et al. 1978] are responsible for two accessible renditions. The latter analysis incorporates the fact that the probabilities of failure on successive reconnect attempts are not independent.

Bard is responsible for the original work on multipathing, both in the case of static reconnection algorithms [Bard 1980] and in the case of dynamic reconnection algorithms [Bard 1982]. Bard's approach relies on a *maximum entropy* formulation of the problem.

Buzen and von Mayrhauser [1982] present an interesting analysis of various considerations affecting the modelling and the performance of the IBM 3880-13 cached storage controller. The discussion in Section 10.6.2 is based partially on their work.

Hunter [1982] explores the process of parameterizing queueing network models of I/O subsystems from typical measurement data, in the context of IBM's MVS operating system. The discussion in Section 10.7.1 is based partially on his work.

[Bard 1980]
Yonathan Bard. A Model of Shared DASD and Multipathing. *CACM 23*,10 (October 1980), 564-572.

[Bard 1982]
Yonathan Bard. Modeling I/O Systems with Dynamic Path Selection, and General Transmission Networks. *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (1982), 118-129.

[Brandwajn 1981]
Alexandre Brandwajn. Models of DASD Subsystems: Basic Model of Reconnection. *Performance Evaluation 1*,3 (November 1981), 263-281.

[Buzen & von Mayrhauser 1982]
Jeffrey P. Buzen and Anneliese von Mayrhauser. BEST/1 Analysis of the IBM 3880-13 Cached Storage Controller. *Proc. CMG XIII International Conference* (1982), 156-173.

[Hunter 1982]
David Hunter. Modelling Real DASD Configurations. In R.L. Disney and T.J. Ott (eds.), *Applied Probability - Computer Science: The Interface, Vol. I.* Birkhauser, 1982, 451-468. Also appears as IBM T.J. Watson Research Center Report RC-8606.

[Wilhelm 1977]
Neil C. Wilhelm. A General Model for the Performance of Disk Systems. *JACM 24*,1 (January 1977), 14-31.

[Zahorjan et al. 1978]
J. Zahorjan, J.N.P. Hume, and K.C. Sevcik. A Queueing Model of a Rotational Position Sensing Disk System. *INFOR 16*,3 (October 1978), 199-216.

10.10. Exercises

1. The example of Section 10.2 involves a CPU, five equally loaded disk devices, and a channel utilized roughly 65%. Clearly the channel represents a performance problem. Suppose a second channel were added to the system, and two of the five disks moved to it.
   a. Use the iterative technique of Section 10.2 to estimate system throughput under the assumption that the disks do not have rotational position sensing capability. Compare the channel contention component of effective disk service demand with the new configuration to that shown in Table 10.1 for the single channel configuration.

   b. Perform the same calculations under the assumption of RPS disks. Compare your results to those shown in Table 10.2.

2. Consider the simple models of channel contention discussed in Section 10.2 (Algorithm 10.1 for non-RPS disks) and Section 10.3 (Algorithm 10.2 for RPS disks). Show that for fixed seek times, rotation times, data transfer times, and visit counts, the "effective service demand" will be lower with rotational position sensing than without it, for any disk throughput that does not saturate the channel. (Assume a single transaction workload, and a latency equal to one half of a rotation.)

3. Consider a new disk technology in which each disk contains a one track buffer. Assuming a simple channel/disk view of the I/O subsystem (i.e., ignoring other path elements), the disk would operate as follows. When performing a read operation, seek and initial latency would be performed independently of the channel. If the channel was idle when the data to be read rotated under the heads, the disk would gain control of the channel and perform the data transfer. If the channel was busy when the data became available, the entire track would be copied into the disk's buffer, and the disk would queue in a FCFS manner for the channel. When the channel became available, the data would be transferred from the buffer. When performing a write operation, the buffer would not be used (i.e., the disk would operate as a standard RPS device).

   a. Give an expression for the effective disk service time. What input parameters are required?

   b. Describe an (iterative) approximation technique for modelling this disk technology.

4. In deriving the expression for $retries_k$ (the average number of retries required by device k), we have assumed that the probability that a reconnect attempt fails is independent of the number of attempts made so far. However, it appears that in practice the probability that the second and subsequent attempts fail is slightly larger than the probability that the first attempt fails.

   a. What does this indicate about the tendency of the procedures described in this chapter to under- or over-estimate system response time?

   b. Suppose you knew that the probability of a reconnect attempt failing was 10% higher on the second and subsequent attempts than on the first attempt. Give an expression for the average number of retries required.

   c. In practice, an unlimited number of reconnect failures is not possible. After some fixed number of failures, the disk queues for the channel, and reconnects as soon as possible regardless of the position of the desired data relative to the heads. What does this indicate about the tendency of the procedures described in this chapter to under- or over-estimate system response time? Does this amplify or diminish the effect indicated by your answer to (a)?

5. The complex approach to modelling multi-element I/O paths taken in this chapter was necessary for two reasons. First, a single job may use more than one path element at a time. Such simultaneous resource possession cannot be modelled directly by separable queueing networks. Secondly, measurement tools frequently do not provide sufficient information about the usage of the I/O path elements.

   a. What sorts of measurement information would be useful in modelling complex I/O subsystems?

   b. How could you modify the procedures given in this chapter to take advantage of such information?