

# **DESIGNING FOR PERFORMANCE**

## **IMPORTANCE OF UNDERSTANDING TIME SCALES AND THEIR INTERACTIONS**

The most important characteristics of computer system performance involve the concept of **time**.

The two most commonly asked performance questions are

- How many tasks were completed during some given interval of time?
- How long did each task take to complete?

The first question asks for a throughput, which is the rate at which jobs get completed.

The second question asks for what is commonly called the *response time* or the *residence time* of jobs in the system.

Performance is judged to be better when throughputs are larger and when response times are smaller.

Typically there is a trade off between these two conditions (which we will see explicitly).

Dependence of performance on time  $\Rightarrow$  must consider all relevant time scales and their interactions when evaluating designs.

This helps to:

- assess designs in terms of performance requirements,
- identify areas for investment, and
- guide decisions on how to allocate resources.

Storage systems provide good examples of these points because they involve time scales that range over many orders of magnitude.

Storage systems are composed of very slow disk drives connected to fast memory through chains of buses of varying speeds.

Firmware executes on fast microprocessors.

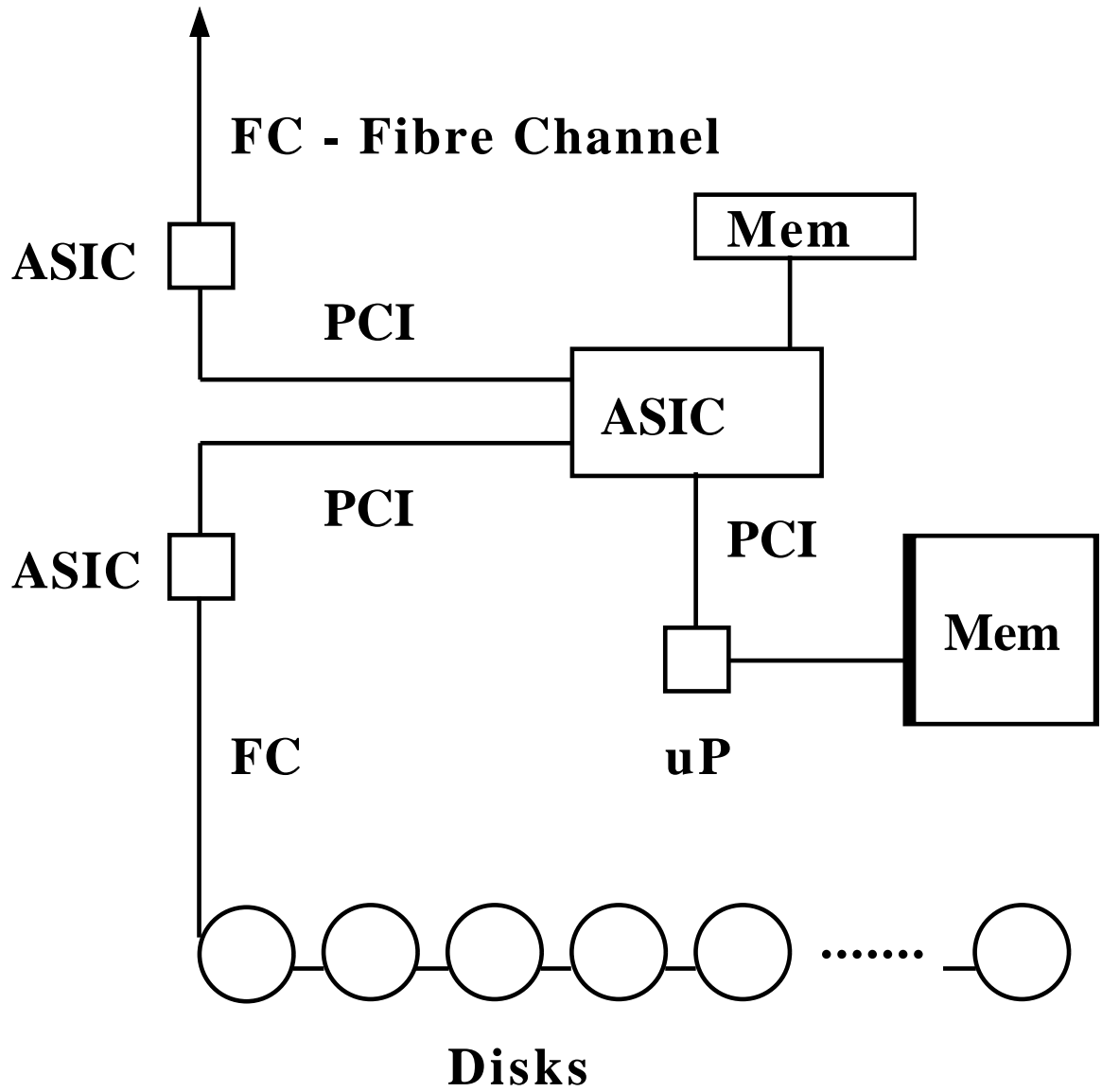


Figure 1: A Hypothetical Storage System

Time scales for a typical storage system.

<b>Event</b>	<b>Duration (s)</b>
$\mu P$ period	$10^{-9}$
$\mu P$ instruction fetch	$10^{-8}$
$\mu P$ exec time for small task	$10^{-7}$
FPCI transfer time (small)	$10^{-6}$
FC transfer time (small)	$10^{-5}$
$\mu P$ exec time for 1 I/O request	$10^{-4}$
FC transfer time (large)	$10^{-3}$
Disk I/O time	$10^{-2}$

Another perspective: re-scale time so that the  $\mu P$  period corresponds to 1 minute.

<b>Event</b>	<b>Rescaled Duration</b>
$\mu P$ period	1 minute
$\mu P$ instruction fetch	10 minute
$\mu P$ exec time for small task	1.7 hours
FPCI transfer time (small)	17 hours
FC transfer time (small)	1 week
$\mu P$ exec time for 1 I/O request	2.3 months
FC transfer time (large)	1.9 years
Disk I/O time	19 years

To get acceptable performance for a disk array we make use of concurrency.

The time it takes the  $\mu P$  to execute all of the instructions to perform one disk I/O is  $\sim 10^{-4}s$ .

This means that the  $\mu P$  can deliver a throughput of  $T \sim 1/10^{-4}s^{-1} = 10,000s^{-1}$ .

The time it takes a **single** disk drive to satisfy an I/O request is  $\sim 10^{-2}$ .

This means that a disk drive can deliver a throughput of  $T \sim 100s^{-1}$ .

To avoid moving at the slowest (disk) speed we

- use many disk drives and
- launch many requests (execute the  $\mu P$  instructions) to many drives concurrently.

This seems simple enough, but...

How many drives should we use?

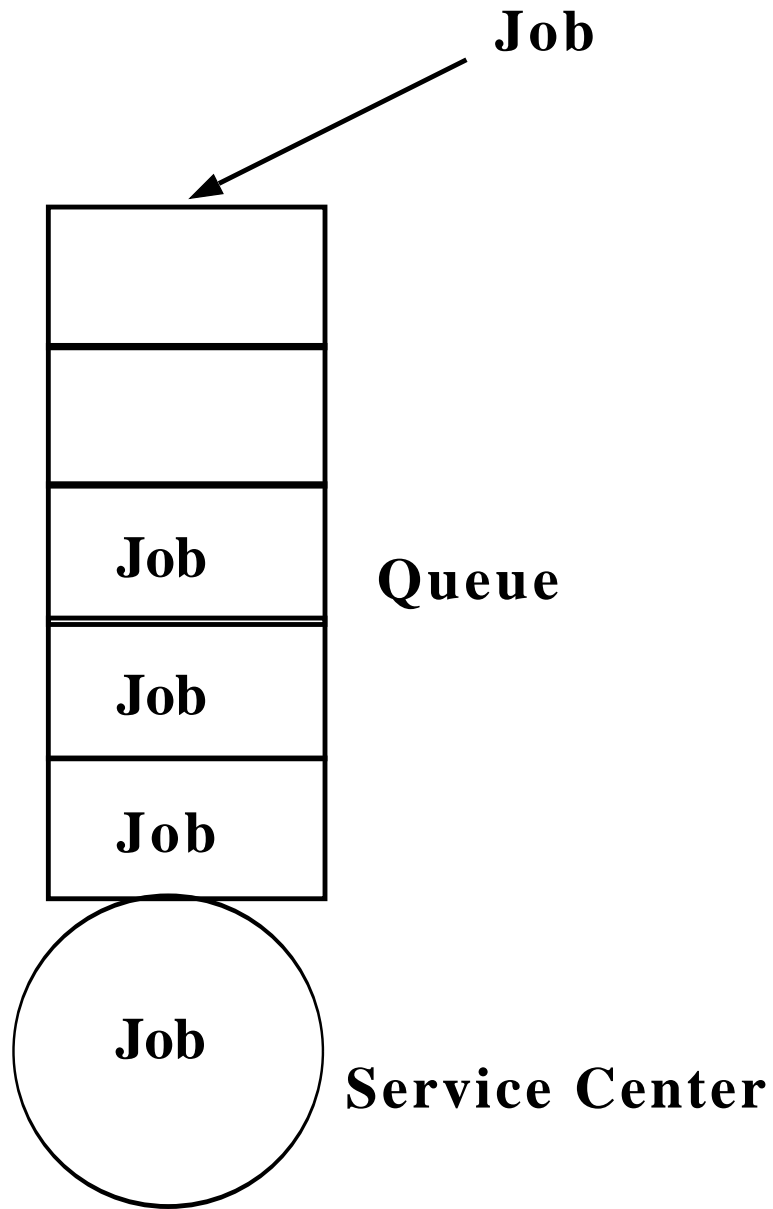
Suppose marketing tells us that we must support some number of drives that prevents us from delivering a performance-optimal solution?

If our solution is not optimal, how far off are we?

This requires that we understand how the disk time scale interacts with  $\mu P$  time scale for executing I/O requests.

But first...

## **Interlude: A Wee Bit o' Queueing Theory**



**mean service time =  $s$**

**mean wait time =  $w$**

**mean response time =  $r$**

$$\mathbf{r = w + s}^2$$

Figure 2: A Single-Queue Queueing System

Let's introduce a few quantities useful for studying queueing systems.

$s \equiv$  the mean service time.

$r \equiv$  the mean response time.

$n \equiv$  the mean number of jobs at a queueing system.

$A \equiv$  the mean arrival rate (jobs arriving per unit time).

$T \equiv$  the mean throughput (jobs completed per unit time).

Assume that queues can be arbitrarily long.

How are these quantities connected?

First, there is the *steady state* assumption (which we adopt now)

$$A = T.$$

Next, there is Little's law that connects  $A$ ,  $n$ , and  $r$  as follows (which is analogous to distance = speed \* time).

$$n = Ar.$$

Third, we need to state a result, that we have already used, which states that the maximum throughput of a queueing system is

$$T_{\infty} = 1/s.$$

As  $T \rightarrow T_{\infty}$ , the system is said to approach *saturation*.

Note that  $0 \leq T = A \leq T_{\infty}$ .

Now,  $n$  is the average number of jobs in the system.

When a new job arrives there will be  $n$  jobs (on average) in front of it.

These  $n$  jobs will take a time of about  $ns$  to clear out, then it will take an additional time  $s$  to service this new job.

So, using the definition of mean response time, we get

$$r = ns + s = (n + 1)s.$$

Using Little's Law ( $n = Ar$ ) and  $T_\infty = 1/s$ , we can eliminate  $n$ , and, with a little rearranging, get

$$r = \frac{s}{1 - \frac{A}{T_\infty}}.$$

The ratio  $u = A/T_\infty$  is called the utilization and varies between 0 and 1.

To make interpretation a bit easier, we'll express the mean response time in units of the mean service time as follows:

$$\frac{r}{s} = \frac{1}{1 - \frac{A}{T_\infty}}.$$

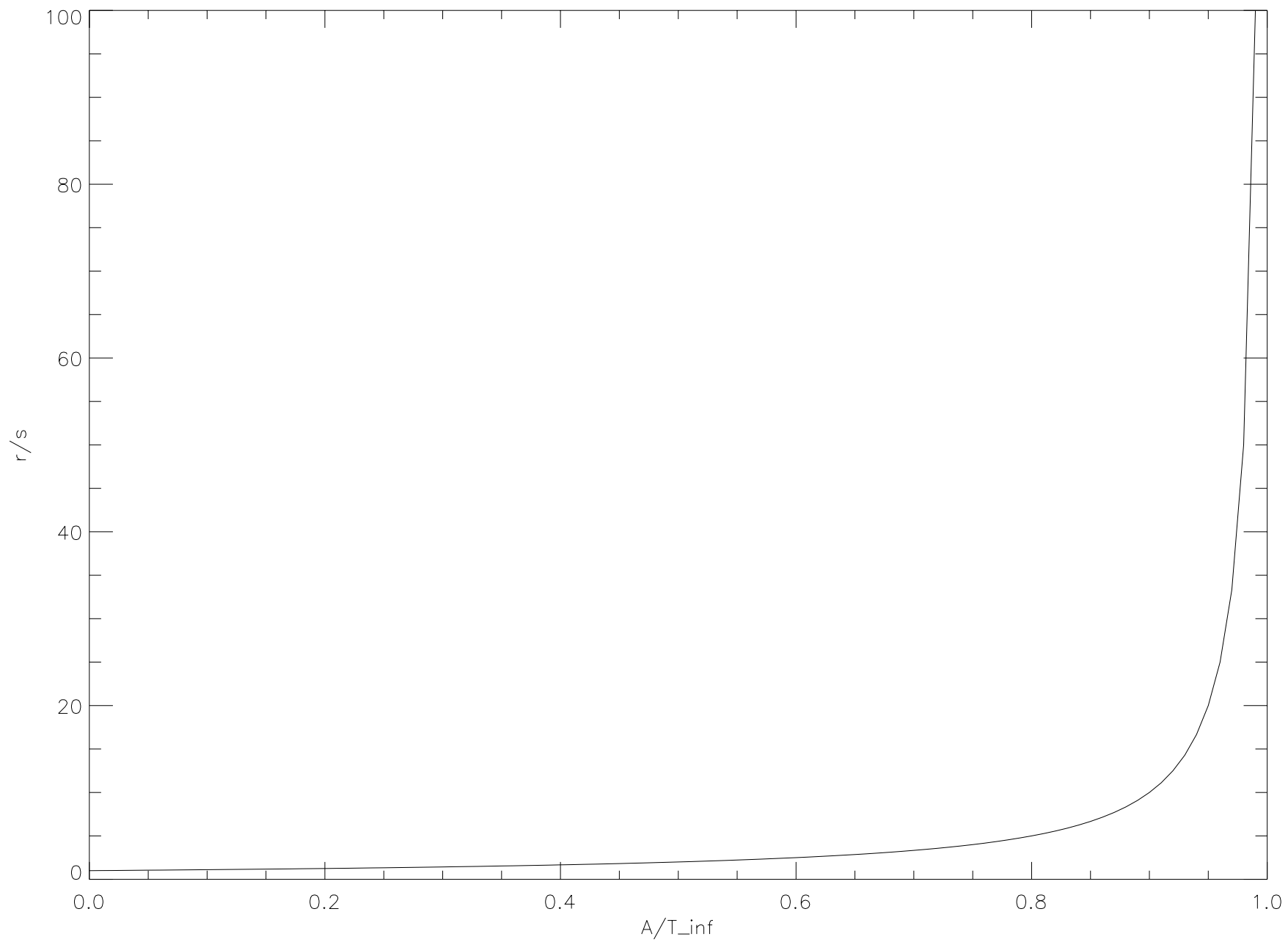
We can use a Taylor series expansion to examine  $r/s$  when  $A/T_\infty \ll 1$ . We get

$$r/s \approx 1 + \frac{A}{T_\infty}.$$

Also, as  $A/T_\infty \rightarrow 1$ ,  $r \rightarrow \infty$ .

**This concludes the interlude**

Saturation Curve



# A Simple Disk Array Model

## Array assumptions

- Array  $\equiv$  a processor subsystem plus  $N$  disks.
- processor subsystem and each disk are treated as a single-queue queueing system.

## Workload assumptions

- I/O requests arrive at a rate  $A$ .
- Each I/O request causes 1 disk access.
- Disks are equiprobable ( $1/N$ ).

## Also

- mean disk service time is  $s_d$  and mean processor service time is  $s_\mu$ .
- system is in a steady state, *i.e.*, ( $A = T$ ).

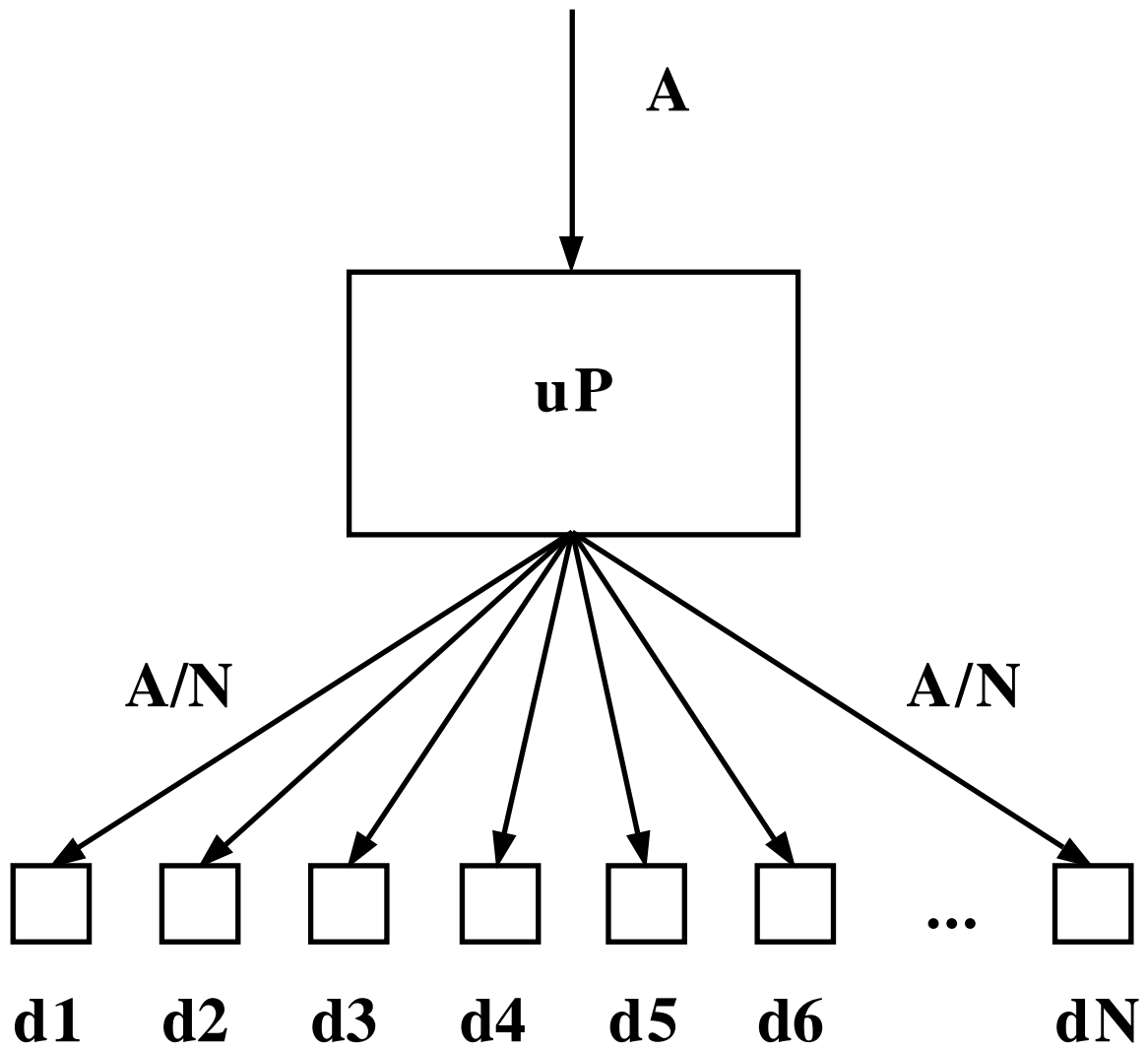


Figure 3: A Simple Model of a Disk Array + workload

Mean residence time for an I/O request (job) is

$$\frac{s_{\mu}}{1 - \frac{A}{T_{\mu}}}$$

Mean residence time at disk is

$$\frac{s_d}{1 - \frac{(A/N)}{T_d}} = \frac{s_d}{1 - \frac{A}{NT_d}}$$

So the mean response time is

$$r = \frac{s_{\mu}}{1 - \frac{A}{T_{\mu}}} + \frac{s_d}{1 - \frac{A}{NT_d}}$$

or, normalizing to units of the disk service time,

$$\frac{r}{s_d} = \frac{1}{1 - \frac{A}{NT_d}} + \frac{s_{\mu}}{s_d} \frac{1}{(1 - \frac{A}{T_{\mu}})}$$

Keep in mind that

$$\frac{s_{\mu}}{s_d} \approx \frac{10^{-4} s}{10^{-2} s} = 10^{-2}.$$

Just for reference...

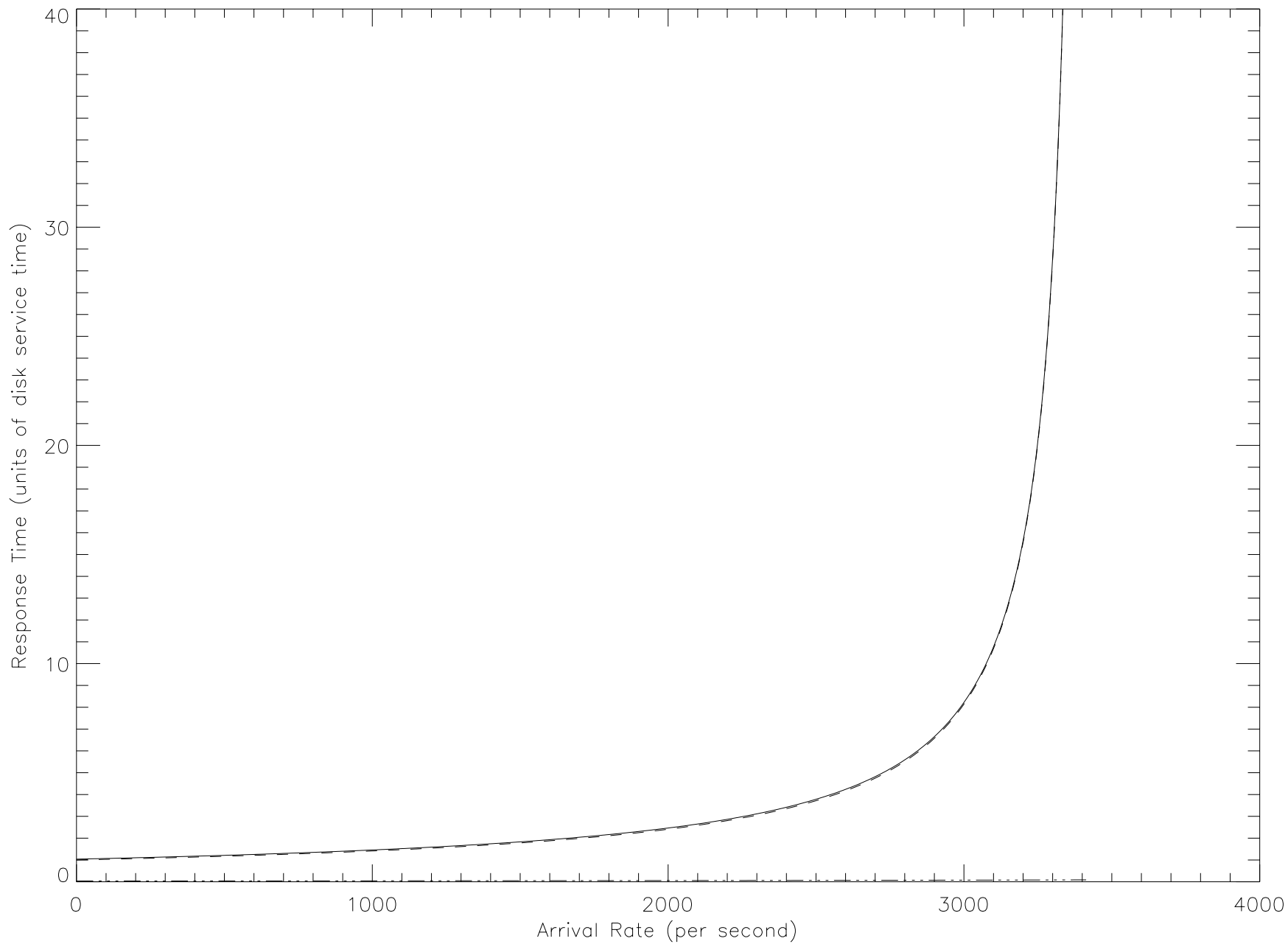
$$\frac{r}{s_d} = \frac{1}{1 - \frac{A}{NT_d}} + \frac{s_\mu}{s_d} \frac{1}{\left(1 - \frac{A}{T_\mu}\right)}$$

Note that

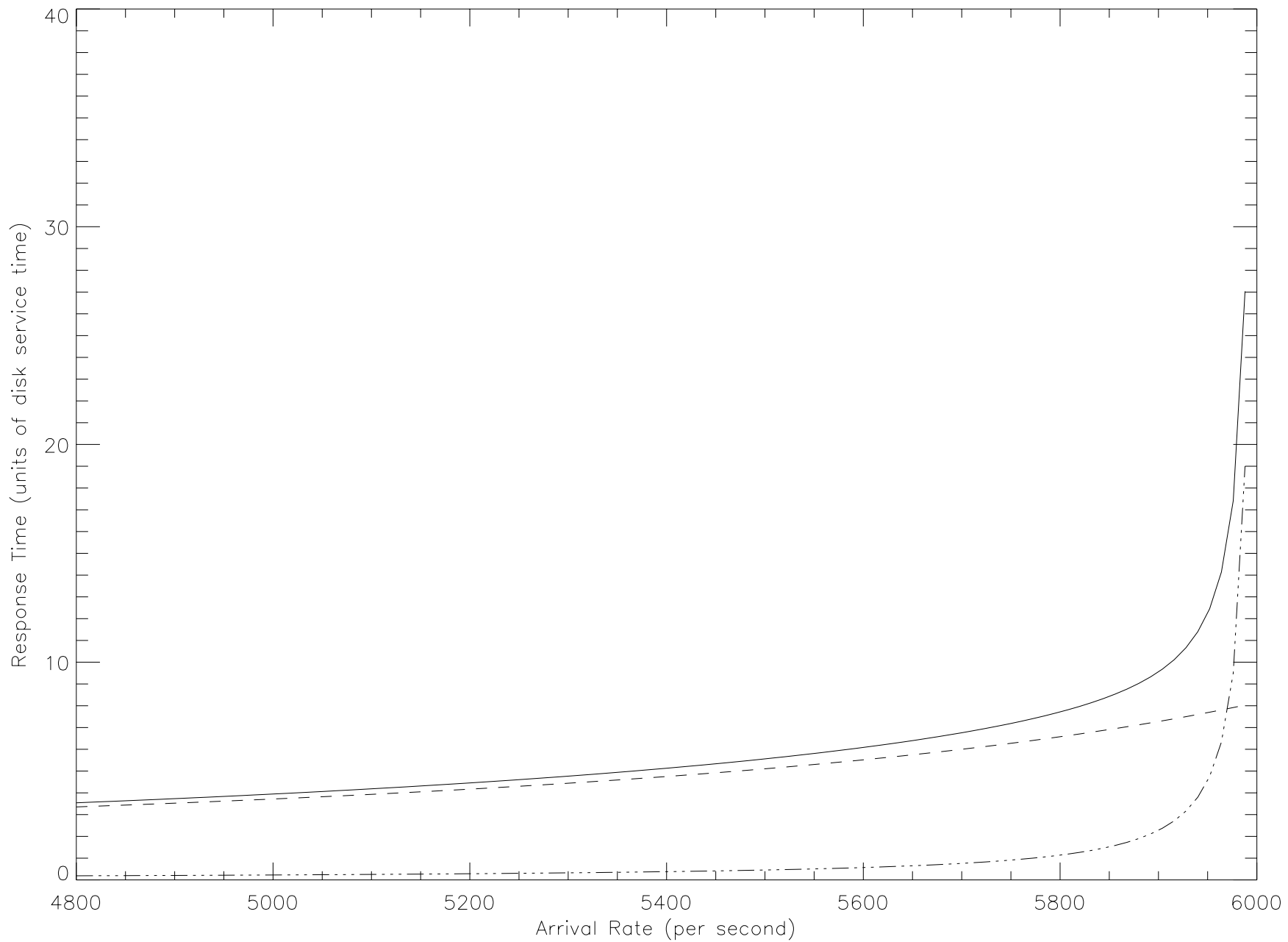
if  $NT_d \leq T_\mu$ , the disks dominate response time inflation for all  $A$ , and

if  $NT_d > T_\mu$ , the  $\mu P$  dominates response time inflation for sufficiently large  $A$ .

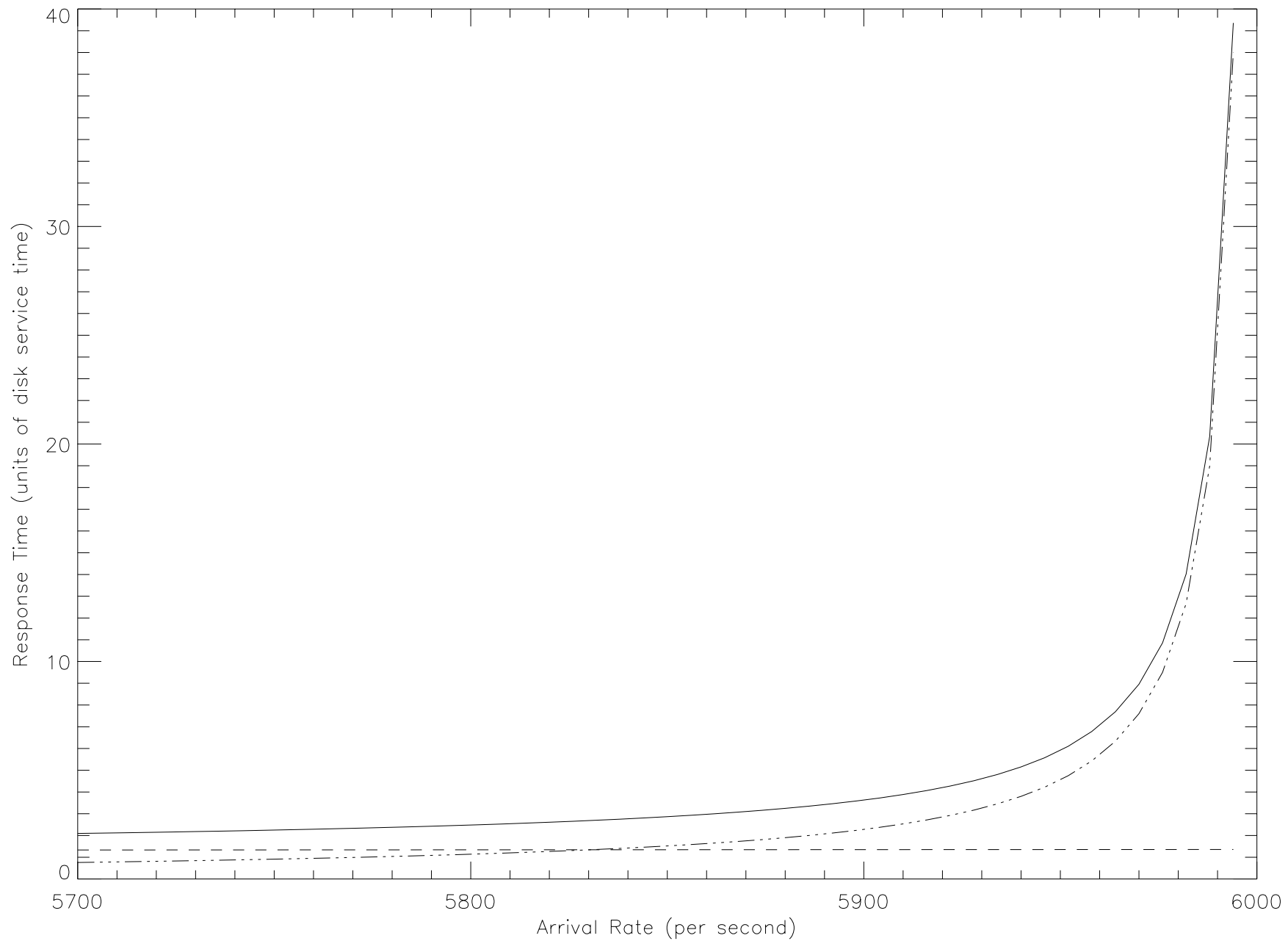
$N = 15$



$N = 30$



N = 100



Conclusions from the example:

It is sufficient to tune the code so that the execution time results in a saturation throughput that equals that of all the drives to be supported.

If the number of disk drives to be supported is large enough that their aggregate throughput is greater than the code can support, either a code tuning investment needs to be made or performance expectations must be lowered.

If expectations are to be lowered, the lowering can be quantified.