Rotational speed = 10000 RPM. Thus, the time for one rotation is

$$\frac{1 \text{ minute}}{10000 \text{ rotations}} * \frac{60 \text{ seconds}}{1 \text{ minute}} = \frac{60 \text{ seconds}}{10000 \text{ rotations}} = 6\text{ms per rotation}$$

*1) What is the read and write service times TS for a single stripe unit on a single disk?*

There are three componenets to any disk access--seek time, rotational delay and finally transfer time. Seek time = average seek time = **4.9ms**

Rotational delay, because we are seeking on only one disk, is 1/2 * 6ms = **3 ms**

One stripe unit is 8KB. Each sector is 512 bytes. Thus, one stripe unit is 16 sectors. Our disk has 200 sectors per track, and performs one rotation in 6ms. Thus, the transfer time is 16/200 * 6ms = **.48ms**

Thus, the total time = 4.9ms + 3ms + .48ms = **8.38ms**

The write service time is the same.

*2) What is the read and write service times TS for a single stripe unit on a RAID level 5 configuration composedof five disks.*

2a) Read time

Despite the fact that we are dealing with a RAID 5 configuration, the read time is identical. When dealing with data the size of a stripe unit or smaller, it is entirely contained on a single disk. Additionally, parity data is neither accessed nor modified on a read.

Thus, the total time = 4.9ms + 3ms + .48ms = **8.38ms**

2b) Write time

Small writes do differ from the single disk case. This is due to the need to updateparity data when writing the new stripe unit. Here is a stripe on a RAID 5 configuration with 5 disks

| D0 | D1 | D2 | D3 | P |
|---|---|---|---|---|

Where each box represents a stripe unit on one of the five disks. D0-D3 represent data stripe units and the P is the parity stripe unit. P is calculated as a function of the data stripe units. If that function is exclusive or (XOR), then

P = D0^D1^D2^D3

Where ^ signifies XOR.

Two steps are necessary to write a new stripe unit--we must overwrite an old stripe unit and we must update the parity. For example, if we are overwriting stripe unit D0, the stripe will look as follows

| | | | | |
|---|---|---|---|---|
| D4 | D1 | D2 | D3 | P |

Where P = D4^D1^D2^D3

Recall that (a^b)^a = b

Thus, if we read the old data and the old parity, it will be possible to cancel out the effect from the old data simply by XORing it with the old parity. Thus

P = D0^D1^D2^D3
(D0^D1^D2^D3)^D0 = D1^D2^D3
D1^D2^D3^D4 = Pnew

Thus, to write a new stripe unit, 4 steps are necessary

1) read the data unit being overwritten
2) read the old parity value
3) write the new data unit
4) write the new parity unit

So we require 4 total disk accesses to two different disks, where one read and one write occurs on each of these two disks. Because of this, we can handle the two reads in parallel and the two writes in parallel (but we must perform reads and writes serially).

The timing is as follows

First, we have to read the data unit and the parity unit.
We again assume average seek time = **4.9ms**
Because we are reading data from two disks, the rotational delay will be different than previously. After seeking to the correct track, when only using a single disk we were equally likely to have to wait any amount of rotation time, from no delay to waiting almost an entire rotation. Because any delay in this range is equally likely, the average delay is simply half of a rotation. When dealing with multiple disks, however, this simple model no longer works. This is because the rotational delay we experience is bounded by the *longest* delay incurred by *any* of the disks. For example, in this case we have to read both the old data unit as well as the parity unit before we can write the new data unit or the new parity unit. Thus, whether we read the data unit or the parity unit first is irrelevant--we cannot proceed until *both* have been read.
The model used, then, for the rotational delay is given as N/(N+1) * rotation time, where N is the number of disks. Thus, the total rotation delay when reading from two disks is, on average, 2/3 * 6ms = **4ms**
Finally, we must read the actual data. The time to read a stripe unit is the same as before, **.48ms**

Having read the data and the parity, it is necessary to write the new data and the new parity. This writing phase can be modeled in two different ways.

The disk locations we need to write to with the new data and parity units are the *same* locations that we read from before. Thus, to write to these locations, we don't require any seeking, since they occur in the current track.

The rotational delay is equal to the time it takes to rotate back to the beginning of these stripe units = (200 - 16)/200 * 6ms = **5.52ms**

Finally, the new data and parity must be written, which takes **.48ms**

Thus, the complete time to write a single stripe unit is 8.38ms + 6ms = **14.38ms**

However, this problem could also be done a different way. Under a more aggressive disk controller, we might permit additional disk requests to be handled between the reading of old data and the writing of new data in this example. In that case, the write time would be equal to the read time (since we would have to seek back to the appropriate track and pay the rotational delay cost again), giving a total write time of **16.76ms**

Either answer is acceptable

*3) What is the read and write service times TS for a 1 MB operation on a single drive?*

1MB = 1048576 bytes = 2048 sectors = 10.24 tracks

Because our accesses are over multiple tracks, we have a decision to make regarding the rotational delay we model. A smart disk controller might begin reading or writing data immediately upon seeking to the correct track, regardless of whether we had yet rotated to the "first" sector. The controller would reorder the data in an internal buffer before it is delivered to the processor. Because the cost of such a buffer is small, but provides good performance benefits, we assume such a configuration, and neglect rotational delay as a result.

Because the data we are concerned about occupies 10.24 tracks and is contiguous, it is located on no more than 11 tracks. Thus, because we don't know specifically how the data is laid out, we model the transfer time as an access of 11 *full* tracks.
Seek time = average seek time = **4.9ms**

Rotational delay = **0 ms**

To read 11 tracks, the time is 6ms + 11 = **66ms**. However, this is not the complete time. In addition to reading each track, we must seek between the adjacent tracks. Because we have 10 'small' seeks (we modeled the seek time to the first track as the average seek time), we also have to pay the additional 10 * 1.1ms = **11 ms** penalty.

Thus, the total access time to read or write 1MB = 4.9ms + 66ms + 11ms = **81.9ms**

*4) What is the read and write service times TS for a 1 MB operation on the RAID level 5 configuration composed of five disks.*

When reading large data on a RAID 5 configuration, we have the advantage of being able to read from multiple disks in parallel. However, the reading bandwidth is is only that of 4 disks rather than 5. This is because, on *each* disk, every 5th stripe unit that passes under the read head is a parity unit. Even if this parity is ignored, the disk head still passes over these parity units, which reduces our total disk bandwidth.

The total data to be read is 10.24 tracks worth of data. However, 1/5 of what we read will be parity information, so we actually have to access a total of 12.288 tracks. This works out to 2.45 tracks per disk. We again choose to model this as accessing a total of 3 tracks per disk.

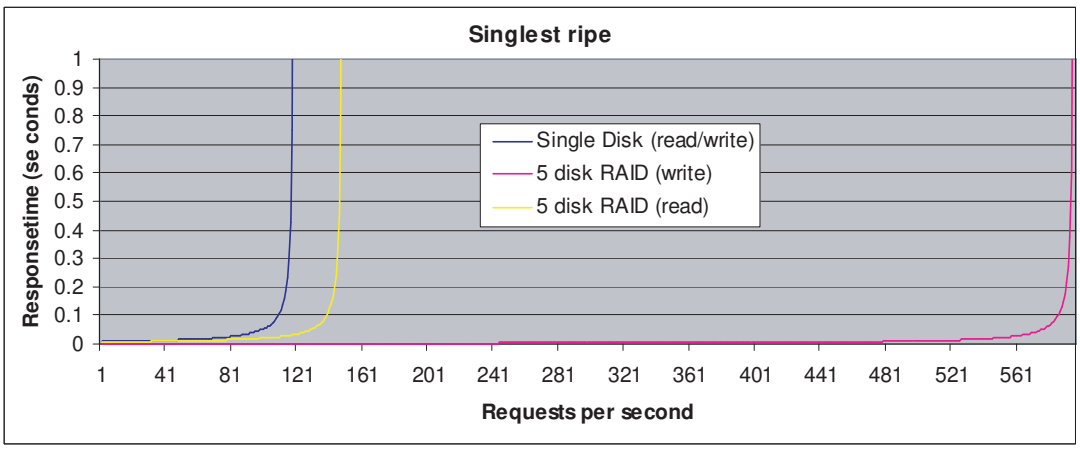
Seek time = **4.9ms**
Rotaional latency = **0ms**
Transfer time = 3*6ms + 2*1.1ms = **20.2ms**

Thus the total read time = **25.1ms**

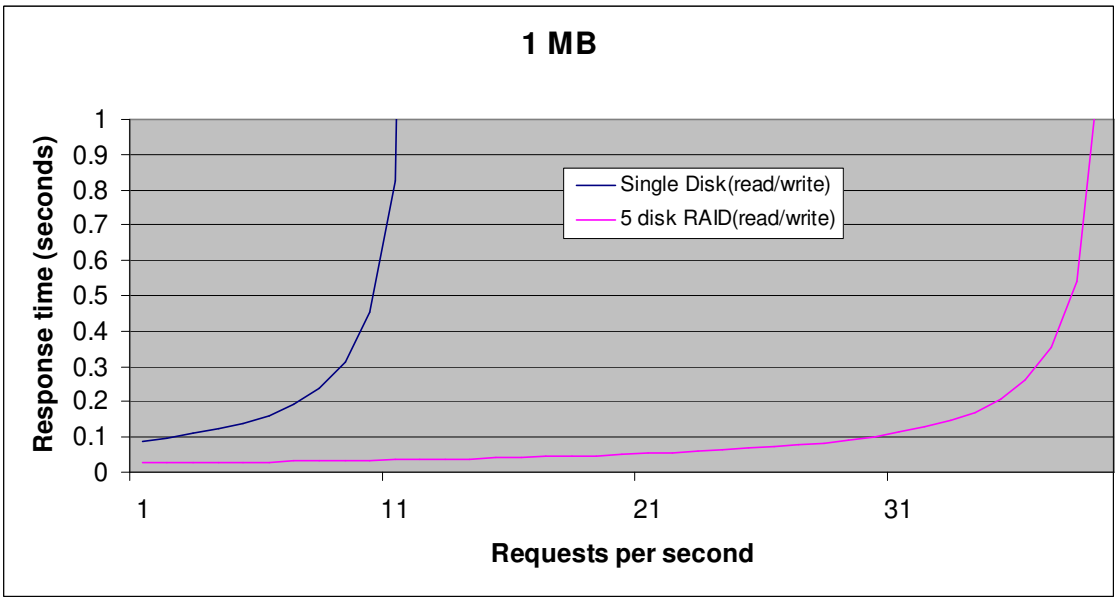
The write time is the same. This is because we don't have to read the old parity when performing a large write. When writing to *all* disks, all stripes units which determine parity values are being overwritten. Thus, our disk controller can compute the parity units based solely on the data being written to the disk without accessing any data already on the disk.

*5) Graphs of response times*

Response times can be modeled by the formula 1/(1/Ts - arrival_rate)
For Ts we use the answers to the previously solved problems. The graph for times assuming a single stripe is being accessed is shown below




We see that in the case of single stripe reads, RAID 5 significantly outperforms a single disk. This is because the RAID configuration is able to exploit the parallelism between disks, exposing parallelism that does not exist in the single disk case. For writes, the RAID configuration also outperforms the single disk case, but by a much smaller margin.

For the RAID configurations, the Ts was divided by the amount of parallelism that could be exposed. When graphing for read times, the Ts previously computed was divided by 5, and when graphing write times it was divided by 1.25.

Below are graphed the response times for the 1MB cases. Notice again that RAID significantly outperforms the single disk case.