1 Introduction

In this project, you will use DiskSim to explore the performance of disk systems. DiskSim is a highly-accurate, highly-configurable storage system simulator that models a number of modern disk drives, controllers, caches, and busses. In this project, you will quantify how the choice of a disk's scheduling algorithm affects I/O performance.

2 Background on disk scheduling algorithms

Disk scheduling algorithms are used to increase performance by reordering queued disk requests. The simplest algorithm is First-Come-First-Served (FCFS), which services requests in the order in which they arrived. Cyclic-LOOK (CLOOK) reduces service time by servicing requests in ascending LBN order; once the head reaches the end of its sweep, it returns to the first cylinder and again services requests in ascending order. Shortest-Seek-Time-First (SSTF) chooses the queued request that is closest to the previous request in seek distance, regardless of the direction that the head is moving. Most real implementations of SSTF and CLOOK assume that distance in LBN space corresponds to physical seek distance on the disk, thus requiring no details about the disk geometry and mapping algorithms. Shortest-Positioning-Time-First (SPTF), on the other hand, requires full knowledge of the layout and positioning characteristics of the disk. It estimates the service time of each request in the queue and chooses that request which will be serviced most quickly. Since this information is not generally known outside of the disk, SPTF is usually used only within disk firmware.

The key metric of scheduler performance is usually response time. This is easy to measure: it includes the service time at the disk and the time spent in the disk's queue. Another metric of interest is the fairness of an algorithm. Greedy algorithms such as SSTF and SPTF can starve requests that are far from the current head position. One measure of fairness is the squared coefficient of variation, which can be computed from the standard deviation σ and the mean response time μ:

$$\text{squared coefficient of variation} = \frac{\sigma^2}{\mu^2}$$

As we'll see below, the mean and standard deviation of the response time is output by DiskSim, so we can measure both the performance and fairness of the various scheduling algorithms.

3 DiskSim

DiskSim is implemented in (mostly) standard C and can be easily built and run on most UNIX and Linux platforms. Included in the package is all of the source code, the parameter files you'll need to do the project, and the DiskSim Users' Guide, which details DiskSim's many options. If you have further questions about using DiskSim, look there first or ask the TA's if you can't find an answer.

Also, DiskSim is a research tool rather than commercial code. You may encounter bugs – let the TA's know if you do. We'll do what we can to fix or avoid these problems.

3.1 Building DiskSim

The version of DiskSim that you will use is distributed as a tar file. First you need to get the distribution from the class webpage at `http://www.ece.cmu.edu/~ganger/546.spring02/disksim_proj1.tar.gz`. Next, untar the file and use `make` to build the DiskSim binary. The unpacked distribution is about 11 MB so make sure you have enough free quota! (If you need a class account or AFS space, please let the TA's know immediately.)

```
% tar -zxvf disksim_proj1.tar.gz
% cd disksim_proj1/src
% make
% ./disksim
Usage: ./disksim paramfile outfile format iotrace synthgen?
%
```

The `disksim_proj1` directory has several subdirectories. The source code is located in the `src` directory, the DiskSim User's Guide is in the `doc` directory, and the parameter files you'll be using are in the `atlas10k` directory.

3.2 Running DiskSim

DiskSim is run from the command line and requires five command-line options and zero or more parameter overrides:

```
disksim parfile outfile tracetype tracefile synthgen? [parameter overrides...]
```

- `disksim` is the name of the executable
- `parfile` is the name of the parameter file to use. To start with, you'll be using atlas10k.par.
- `outfile` is the name of the output file. Output can also go to the terminal window by specifying `stdout` here.
- `tracetype` is the type of the input trace. We'll be using internally-generated workloads, so this parameter is specified but ignored.
- `tracefile` is the name of the input trace file. We'll be using internally-generated workloads, so this should be 0.
- `synthgen?` is a boolean telling DiskSim to generate a synthetic workload. Since this is what we want, this should be 1.
- `[parameter overrides...]` are the optional command-line overrides for parameters. These are described below...

All of the parameter files you will use are in the atlas10k directory. So, to run a simple test of DiskSim, do the following:

```
% cd disksim_proj1/atlas10k
% ../src/disksim atlas10k.par out.atlas10k ascii 0 1
```

This simple test will run DiskSim using the model of the Atlas 10k specified in `atlas10k.par` with a simple synthetic workload of 10,000 random requests. You can look at the results in the file `out.atlas10k`. Use `grep` to find the following lines:

```
IOdriver Requests per second:          40.662599
IOdriver Response time average:        10.551480
IOdriver Response time std.dev.:       7.752829
IOdriver Response time maximum:        65.468647
```

These are the output values that matter for this project. They give the number of requests per second, the average and maximum response time, and the standard deviation. There are many more statistics in the output file, each of which is described in the DiskSim Users' Guide, but these are the ones we'll use for this project.

3.3 Configuration files

DiskSim uses a set of configuration files to define the parameters of the disk, bus, controller, interconnect, and workload. In this project, we are really only concerned with the parameters for the disk and the workload. For details about each parameter, see the DiskSim Users' Guide. We will be using parameters for a Quantum Atlas 10k disk drive; all of the configuration files you need are in the `atlas10k` directory.

The main configuration file is `atlas10k.par`, which (among other things) tells DiskSim which components to simulate and how they are connected. In this case there is a device driver, two busses, a controller, and the disk. The device driver is very simple – it just passes requests through across the first bus to the controller. The controller is also very simple – it just passes requests across the second bus to the disk. The `atlas10k.par` file also configures the synthetic workload generator which we'll use to drive the simulator. The parameters for this are described below. The parameters for the disk are in `atlas10k.diskspecs`. Again, there are full descriptions of each parameter in the Users' Guide.

Parameters can also be specified on the command line. This is useful for running DiskSim repeatedly (like with a script) with different parameters without keeping around lots of parameter files for each setup.

Scheduling Policy

This parameter in the `atlas10k.diskspecs` file is an integer which tells DiskSim which scheduling policy the disk should use. A number of algorithms are implemented, but we will be concerned with the four described above. The line from the parameter file, shown below, specifies the algorithm using the numbers shown in Table 1. Watch out, because this line will appear in several places in the various files. Make sure the one you edit is in `atlas10k.diskspecs` or you use a command-line override.

Algorithm	Number
FCFS	1
CLOOK	3
SSTF	4
SPTF	8

Table 1: Scheduling policies

```
[...]
Scheduling policy:                    1
[...]
```

This parameter can also be overridden on the command line as follows:

```
% disksim atlas10k.par out.atlas10k.CLOOK ascii 0 1 disk -1 ioqueue_schedalg 3
```

This will override the Scheduling Policy parameter given in atlas10k.par (FCFS) with CLOOK (policy 3).

Inter-arrival times

The `atlas10k.par` file also specifies the synthetic workload that DiskSim will use. By default, request inter-arrival times are drawn from an exponential distribution with a mean of 25 milliseconds:

```
[...]
General inter-arrival times (in milliseconds)
Type of distribution: exponential
Base value: 0.0
Mean:        25.0
[...]
```

You can change the mean in the parameter file directly or override it on the command line:

```
% disksim atlas10k.par out.atlas10k.50 ascii 0 1 synthio -1 genintr_mean 50.0
```

For workloads with locality, you will also have to scale the inter-arrival time of local requests:

```
[...]
Local inter-arrival times (in milliseconds)
Type of distribution: exponential
Base value: 0.0
Mean:        25.0
[...]
```

This can also be overridden on the command line with the `locintr_mean` parameter.

Notice that increasing inter-arrival time leads to a lighter workload. It usually makes sense to consider scheduling performance as a function of inter-arrival *rate*, the inverse of inter-arrival time, as in Worthington's paper, *Scheduling Algorithms for Modern Disk Drives.* Graphs of response time, say, then show workloads growing heavier from left to right on the X axis.

Locality

By default, the starting address of the requests from the synthetic workload generator will be uniformly distributed throughout the disk's capacity. You can add locality to the workload by setting these parameters in the `atlas10k.par` file:

```
[...]
Probability of sequential access:           0.0
Probability of local access:                0.0
[...]
```

For each request, the generator uses these probabilities to choose whether a request is sequential, local, or random. Sequential requests start at the address immediately following the current request. Local requests start some short distance away from the current request. The distance is computed from a random variable describe below. Lastly, a random request starts at an address uniformly distributed across the storage space of the disk.

The inter-arrival times of sequential and local requests are determined by two random variables:

```
[...]
Sequential inter-arrival times (in milliseconds)
Type of distribution: exponential
Base value: 0.0
Mean:     25.0
Local inter-arrival times (in milliseconds)
Type of distribution: exponential
Base value: 0.0
Mean:       25.0
[...]
```

The fraction of sequential and local requests can be specified as command-line overrides, `seqintr_mean`, and `locintr_mean`, respectively.

The distance of a local request from the current request is also generated from a random variable:

```
[...]
Local distances (in blocks)
Type of distribution: normal
Mean:     0.0
Variance: 40000.0
[...]
```

Request size

Request size is determined by a random variable:

```
[...]
Sizes (in blocks)
Type of distribution: exponential
Base value: 0.0
Mean:       8.0
[...]
```

Combining command line overrides

Command line overrides can also be combined:

```
% disksim atlas10k.par out.atlas10k.50 ascii 0 1 \
  disk -1 ioqueue_schedalg 3 \
  synthio -1 genintr_mean 50.0
```

will run DiskSim with CLOOK and a mean inter-arrival time of 50 ms.

4 Project

Use DiskSim to generate graphs of average response time and squared coefficient of variation for FCFS, CLOOK, SSTF, and SPTF for increasing inter-arrival rates (decreasing inter-arrival times). These should look similar to the graphs shown in Worthington's paper *Scheduling Algorithms for Modern Disk Drives* – each graph should include all four algorithms and be plotted as a function of inter-arrival *rate*, the inverse of inter-arrival time.

The best way to do this project is to write a script. Use Perl or some other scripting language to run DiskSim repeatedly, changing the command-line overrides. The script can then collate the results and then output them to a graphing tool like Gnuplot or Excel. If you've never used Perl before, it's one of the most useful tools you can learn – take my word for it! Once you have a script to automate this first part of the project, the other parts will be pretty straightforward.

Remember that all of your graphs should be made as a function of request *arrival rate* or requests per second. DiskSim requires that you specify inter-arrival time in milliseconds, so remember to convert. Your rates should go up to around 200-300 requests per second, with enough resolution to see the differences in the curves clearly, especially for the squared coefficient of variation graphs. Use a 100 ms response time as the cutoff on the Y axis for the response time graphs.

Once you have a script (or scripts) which generate the graphs, alter the following parameters and show how they differ from the base case. You should produce a brief writeup which includes each of the graphs and explains how the results differ from the base case any why.

1. The first graphs you generate of response time and squared coefficient of variation will provide the basis for the rest of the project. From looking at these baseline graphs, which scheduling algorithm gives the best performance? Which is the most fair? Find the points at which each algorithm saturates – when it cannot service requests quickly enough and response time increases dramatically. Also, find the point at which the scheduling algorithm starts to matter – why does it not matter at lower response times?

2. Introduce some locality to the synthetic workload by setting the probability of local requests to 0.50. Make a graph of average response time with the new workload.

3. By default, the generated requests have a mean size of 4KB (8 512B blocks). What is the effect of larger requests? Show the same average request time graph again, but for 64KB and 512KB requests. With larger requests, the simulation can start taking longer, so you can reduce the number of requests if you need to.

4. Show the effect of prefetching. Prefetching is enabled by default and can be disabled in `atlas10k.diskspecs`:

   ```
   [...]
   Buffer continuous read:          4
   Minimum read-ahead (blks):       0
   Maximum read-ahead (blks):       354
   [...]
   ```

   The `Buffer continuous read` parameter is explained in the DiskSim User's Guide – a value of 4 sets it to prefetch to the end of the track that follows the track containing the last requested sector. A value of 0 will disable all read-ahead.

Show the average request time graph for a disk with no prefetching using the original, random workload. Then show it again with a workload with some locality – use the same 0.50 probability of a local request.

5. In your homework, you studied the performance of zero-latency or "immediate" access. By default, the Atlas10k model performs zero-latency reads, as seen in `atlas10k.diskspecs`:

```
[...]
Immediate buffer read:            1
[...]
```

Disable zero-latency access by setting this parameter to 0. For this graph, modify your script to plot response time as a function of request size. Change the workload such that all requests are reads and fall only inside the first zone (band) of the disk. Find the size in blocks of the first zone from the band information in `atlas10k.diskspecs` and use this as the `Storage capacity per device (blocks)` parameter in `atlas10k.par`. This will force the generator to only generate requests within this range. Use the track size in blocks as the maximum mean request size on your graph.

Given what you learned in the homework, do the results match what you would predict? You should find that the benefit of zero-latency access is small, even at large mean request sizes. Why might this be so? Consider what happens when a large request spans a track boundary, and the cost of this in both the non-immediate and immediate access cases.