Toy DB Project Report
Disk Storage Simulator
Project No. 2

Rohit Kumar
120050028
rohit@cse.iitb.ac.in

Deependra Patel
120050032
pateldeependra06@gmail.com

1 Introduction

This report talks about a strategic simulation of a disk system. The simulation includes an efficient arm movement strategy for processing the requests on a disk system, providing reliability, high performance through implementation of the **RAID-5** technology and caching unit for faster access.

This disk-system is accessed by the paged file module provided at the starting of the project. The statistics related to various performance parameters are gathered and various conclusions and effect of the technologies used are displayed through graphs etc.

2 Structure of the disk-system

A main controller controls and executes all the functionalities provided. This is the structure that is accessed by the paged file module and acts as an interface between the PF layer and the disk.

The main controller provides a mapping of page number to disk addresses. It has an access to a cache controller which manages the disk cache and interfaces between the disk-system and the main controller. The cache controller checks if the data is present in the cache and returns it. If not present, it passes the request to a disk controller which simulates the disk-system. The disk controller passes this request to a request buffer which in turn processes these requests using the elevator algorithm.

While processing the requests various logs and data relating to measure the performance parameters etc. are also being stored.

The details of the various data structures and algorithms used in these classes are described in the below section.

3 Data Structures and Algorithm used for various classes

3.1 Main Controller (mainController.cpp & mainController.h)

The main controller stores a mapping of page numbers of a particular file to the disk address. For implementing this functionality, the map data structure provided by the STL of c++ has been used. Whenever a read request for a particular page in particular file is made

by the paged file module to this, it checks in the map if an address corresponding to this page is present in the map. If present, a read request is made to the cache controller, otherwise if there is no such mapping, it denotes that the page wasn't written in the disk earlier and it is an invalid request. In case of the request is invalid, it is dropped.

Whenever a write request is made, it is checked that the mapping is already present in the map or not. In case it is present, it implies that the file is already present and is being overwritten. In case if it is not present (implying writing for the first time), a new mapping is assigned to the page randomly from the set of available free addresses in the disk and this mapping is put in the map.

There might be a case when the file is being deleted from the disk. In such a case the mapping is no longer necessary and it is deleted from the map.

This main controller passes the requests to the cache controller for performing the operations.

3.2 Main Controller (cacheController.cpp & cacheController.h)

3.3 Disk-System Controller (diskController.cpp & diskController.h)

3.4 Request Buffer and Elevator algorithm (diskRequestHandler.cpp & diskRequestHandler.h)

This contains a class which places the requests into a request buffer and processes it using the elevator algorithm. A data-structure for every request is created which also stores the time-stamp of the request.

These requests are stored in the set data structure provided by the STL of C++.

Upon serving a particular request, the arm will now move to the next closest request in the direction of movement. This request will be obtained by iterating over the tree data structure. The first element of a tree is always the leftmost one. The next element after an element is the first element of its right subtree. If the element does not have a right child, the next element is the element's first right ancestor. If the element has neither a right child nor a right ancestor, it is the rightmost element and it's last in iteration. In this manner the requests will be served until the end/start of the tree has been reached (depending on the direction of the movement). At this movement the direction of iteration will be reversed.