ACS 2015
Assignment 1

Oliver Hammer Boda, pgt135

November 18, 2015

1 Question 1: Fundamental Abstractions

1.1

Implement RAID 4 for K disks: $K-1$ disks are 1:1 sequential mapping of the single address space. The largest disk works as a parity disk. Each disk including the parity disk can be recreated using *XOR* in case of failure, but if more than 1 disk fails, the system will fail and data is lost.
Alternatively, the parity information can be distributed among all disks, this gives better write performance, since each parity update will not be written to the same disk. This is equivalent to RAID 5. 2 disks can fail if RAID 6 is implemented.

1.2

The API exposed to the user is *read* and *write*. Here's the implementations in pseudo-code:

```
diskLimits = [(d1Lower,d1Upper) , (d2Lower,d2Upper), ..., (d<k-1>
    Lower, d<k-1>Upper)]

read(address) {
  diskN = determineDisk(address, diskLimits) // which drive is '
      address' located on?
  addressOnDisk = address-diskLimits[diskN,0] // relativeAddress -
      <base address of disk>
  waitFor(diskN)       // finish writes before we read
  data = fetchData(diskN, addressOnDisk)
  return data
```

```
}

write(address, val) {
  diskN = determineDisk(address, diskUpperLimits) // which drive
     is 'address' located on?
  addressOnDisk = address-diskLimits[diskN,0] // relativeAddress -
      <base address of disk>
  waitFor(diskN) // wait for readers and other writers
  waitFor(diskParity)

  valOld    = fetchData(diskN, addressOnDisk)
  parityOld = fetchData(parityDisk, addressOnDisk)
  parityNew = valOld XOR val XOR parityOld

  writeData(diskParity, parityNew)
  writeData(diskN, addressOnDisk, val)

  resourceIsFree(diskParity)  // inform anyone waiting, that
     parity disk is now free
  resourceIsFree(diskN)       // inform anyone waiting, that diskN
      is now free
  return
}
```

Function *determineDisk* takes an 'address' and a structure of disk-upper and -lower limits and returns which disk 'address' is located on. This could be implemented as a optimal binary search tree. The parity disk should be initialized before use and its size should be greater or equal to the largest of the other disks.

1.3

The read- and write-operations should be atomic; no readers should read old data and no writers should update the same data at the same time. This is achieved by only allowing 1 writer at a time. This way the readers will never read old data, and the writers can not write simultaneously. There can be an unlimited number of concurrent readers.

1.4

The number of machines is not needed to be known a priori. It is possible at any time to connect more machines, but one requirement is, that the largest disk should be the parity disk, if this is not the case then the parity disk must be migrated to the largest disk available.

2 Question 2: Techniques for Performance

2.1

2.2

2.3

3 Question 3: Programming Task