Assignment1 ACS

Cristian Marius-Florin wdx186
Fernando José Faustino Alves xgd375

1 Fundamental Abstractions

For this question we will assume that our cluster has less than 2^{15} machines. We will try to build a RAID system in order to tolerate data loss in case a machine fails. The address will be on 64-bits as it follows: the first bit in the address will represent the memory-lock (1 if there is a writing operation on that address or 0 otherwise); the next 15 bits will represent the machine number on wich the first cell is and the next 32 bits will represent the address inside the machine; the last 16 bits will represent the number of segments to be stored.
By storing duplicates of the data on different machines we increase the fault-tolerance and in case of a fail (a machine disconects or is "too busy") it will switch to another instance.
If the number of the machines exceedes 2^{15} the system won't support new ones as there are no bits left in the machine memory space, thus we must use extra bytes.

API:
All the Read/Write calls on client are made on the Single Address Space

```
//returns a value;
client_read(address)
{
value=address_controller(address,self_reference);
return value;
}
//returns 1 or -1
client_write(address,message)
{
flag=address_controller(address,self_reference,message);
return flag;
```

}

The address controller that sends request to machine controller
Designed as an event handler

```
//value represents the value to be writen or read (if it's null)
address_controller(address, reference, value)
{//we are working with uint64
if(value is null)//read branch
        {
                seg=(address)&0xFFFF;//last 16 bits
                next_addr=null;
                r_value=null;
                value=null;
                while(seg!=0)
                {
                        //machine address
                        machine=(address>>48)&0x7FFF;
                        phisical_addr=(address>>16)&0xFFFFFFFF;
                        send(machine,physical_addr);
                        if(receive(machine,&r_value,&next_addr) timeouts)
                                lookup_duplicate_machine();
                        else
                        {
                        value+=r_value;
                        adress=next_addr;
                        seg--;
                        }
                        if(sig_fail)
                                {
                                send(reference,-1);
                                break;
                                }
                }


        }
else //write branch
{
        success=-1;
        //retry write if its blocked or untill timeout
        while(success=-1)
        {
        if(address>>63)//check if writing lock is set
                {
                set_writing_lock();
                split_value(value,blocks);
                machines=look_up_free_machines();
                for(block in blocks)
                {
                for(machine in machines)
                {
                        //very abstract
                        concatenate(block,next_machine_addr);
                        try {
                        send(machine,physical_address ,block);
                        } if (fail)
                                machine=next_machine();

                }
                }
        else
                if(timeout)
                break;
        }
        unset_writing_lock();
        //sends if write was succesfull or not
        send(reference , success);
}
}
```

.

The machine controller that reads writes blocks into the specific machine

```
//low level machine memory read
machine_controller(address , ref,block)
{
        if(block is null)
        {
        value=read(address);
```

```
          send(ref, value);
          }
          else
          {

          if(write(address, block)==sig_fail)
                  send(ref, sig_fail);
          else
                  send(ref, success);
          }
}
```

.

The read operation is not atomic as there may be a write while still reading, but the write operation is blocking, so there may not be another write while writing on that address. In order to make both read and write atomic, either we use the write-lock for also reading, or we try to update the read if we know it started as a read while the write-lock bit was set. It depends on the file being shared if the operations should be atomic or not.
Some of the data is duplicate on other machines (did not explicity write it in my pseudocode, but i presumed so when I chose a RAID architectures). But if machines leave and enter the system on a regular basis there must be more duplicate data, or else there will appear segfaults or certain blocks can't be reconstructed.

2 Techniques for Performance

2.1

Concurency helps reduce the latency by splitting a large task into smaller subtasks that run in paralel, in an ideal example if a process can be parallelized perfectly then this plan may speed up the process by a factor of n. The downside of using concurency is that many applications are difficult to parallelize, and then again it is difficult for the programmer who must coordinate the activities of different subtasks. Another aproach is to use concurency on splitting the bottleneck stage in n instances and run them concurently.
The influence is mostly positive if you consider that there are physical and engineering limitations on the hardware (such as the heating dissipation) and thus more convenably to parallelize more processors rather than mak-

ing a single concurent one faster.

2.2

Batching is performing several requests as a group to spare the setup overhead of doing them one at a time. Dallying is delaying a request either on the chance that it won't be needed (for example if a disk call gets overwritten by another) or to create more oportunities for batching.
For example instead of sending several messages from one stage to another we can combine all of the messages in one spare the system the expense of making more requests with just one. In this way batching provides an oportunity for the stage to avoid work. Dallying may increase latency of a request with the hope that more requests will come and form a batch with the delayed one.

2.3

Yes caching can be viewed as a fast path optimization. Data that has be used and might be used in the near future is kept in the cache and thus reducing the disk calls (it has to travel a shorter path).

3 Programming Task

We added some test for our implementation of rateBooks,getTopRatedBooks and getBooksInDemand , our tests verified the following properties:
-Negative values
-Invalid ISBN
-Null case parameter
-Positive cases
Note that getTopRatedBooks and rateBooks tests are on BookStoreTest class while getBooksInDemand tests is on StockManagerTest Our JUnit test are made to maintain the at-most-once semantic.

4 Questions for Discussion on Architecture

4.1

a) Our implementation addresses it by checking if any of the cases fails then whole test method fails as well.

b) In order to do so we use JUnit tool environment, this way we can test it if it behaves like it should.

4.2

a) It is strongly modular in the sense that each function/module does one specific task.
b) There is no type of isolation, let's assume we have a manager M and a customer C that do some transactions in the following order of requests on the same book with 0 number of copies. Now if M adds 1 book and M removes 1 book, then C tries to buy 1 book, C will fail to buy the book, as it will no longer be there. If M adds 1 book and C buys 1 book and M tries to remove the book, it will fail thus not maintaining the same order of transaction. The protection offered is in the means that every type of client is limited to the operations assigned to them (for example a customer cannot add books into the store).
c) Yes.

4.3

a) Yes, it exposes the name of the operation on the client side and is used in the BookStoreHTTPMessageHandler to link to method calls.
b) Internet Protocol for the client to communicate with the server.

4.4

At-MostOnce. In the ClientHTTPProxy we set a timeout, if there is no reply the request expires.

4.5

a) It is safe since it uses HTTP. A well configured web proxy server can be used in this architecture since it uses HTTP Requests/Responses between client server. In this case the proxy server would act as an intermediate.
b) It is safe because if it is well configured it will protect the server system from attacks since the client doesn't know the servers direct address. It should be deployed between the HTTPClient and the HTTPServer so it receives the requests from the client and forwards them to the server.

4.6

a) Yes.
b) The machine on wich the server runs has limited resources.

4.7

a) Yes, because they might have their request served from the proxy cache memory.
b) Yes, in the same way the web server masks the servers identity, the proxy server also does the same about server failure, since they can still fullfill the clients request.
c) No, since we always need to maintain an at-most-once semantics.