Problem Set 3 Solution

15-440/15-640 Distributed Systems Spring 2017

Assigned: Tuesday, April 4, 2017

Due: 10:30am on Thursday April 11, 2017

Question 1 (20 points)

As a lead engineer at the startup GREAT_CLOUD, you have convinced your boss that the VM abstraction is valuable. You have done such a good job, that your boss is now enamored of all things VM. He comes up with the idea of combining the VM concept with the classic `fork()` concept of Unix-like systems. Instead of forking a new process, `forkVM()` forks a new virtual machine: in other words, the VM in which `forkVM()` is called pauses, and then creates a copy of itself as another VM; both VMs then continue execution from the current point of execution.

Assume that VM support is provided by a Type II hypervisor such as KVM on a Linux host.

A. An important part of implementing `fork()` is copying memory mapping: the parent process copies the "virtual address → physical address" mapping when it forks a child. What mapping would you want to copy, in terms of memory, if you are implementing `forkVM()`?

B. Suppose a distributed file system that offers strict one-copy semantics (e.g., Lustre) is available across a cluster of machines. What is a major benefit of using `forkVM()` instead of using `fork()` in this context?

C. State and explain two significant disadvantages of `forkVM()` relative to `fork()`.

Solution:

A. The "guest OS physical address (real memory) → host OS physical address" mapping needs to be copied. Note that with type II (hosted) hypervisors abstract guest OS from the host OS. The hypervisor translates from physical to physical.

B. Being able to exploit the computation resources across different machines (which are accessing the same underlying storage through DFS). Forking processes is a good method for running on a single machine only. However, forking VM will have to copy the whole OS state, and this enables users to run the VM copies on multiple machines and leverage parallelism.

C. A few disadvantages:
   1. Huge cost of copying. Not only fd table and memory mapping needs to be copied, the whole OS state code and the status of the applications currently running on it will have to be copied as well. This can take time, and potentially many resources. (If a student splits

this into two categories and answers them separately, “time” and “memory”, it counts as two disadvantages.)

2. Unlike process fork, with `forkVM()` there is no counterpart for IPC (interprocess communication), because the two guest OS are separate. For example, one VM cannot explicitly `wait` on another VM (but a process can).
3. Other valid answers.

Question 2 (20 points)

You are the senior architect of *Muggle.io*, a professional service connecting people with wizards. You've noticed recently that the popularity of the service has skyrocketed, and, being in *15-440*, you decide to go distributed, but need to decide between *scaling up* and *scaling out*.

You want to improve response times, but keep costs low. You are considering the following machine types from a cloud provider, *Cloudr.io*, where they advertise the following servers:

Machine Type	Cost per Month	Speed (billion ops per sec)
Cloudr-Basic	$5	0.5
Cloudr-Pro	$30	2
Cloudr-Enterprise	$150	8
Cloudr-Ultra	$500	16

Here we assume that the price tag is all inclusive: it contains all basic costs for running an instance of the server (license costs, power, cooling, etc.).

Note that the distributed version employs a data parallel approach -- requests are distributed among processing nodes, but it does not split the processing of a single request on multiple nodes. To provide good user experience, you want to ensure the processing latency (excluding queueing and network transfer) is **100ms** or less per request. Keep this in mind when considering these request types:

A. The most common request is for static data (e.g. images). These require 10 million operations per request. What is the cheapest combination (i.e., quantities of each type) of VMs to support 1000 static data requests per second? How about 5000 static requests per second?

B. Requests for dynamic pages require 500 million operations per request. What is the optimal configuration of VMs to support 100 dynamic page requests per second? How about 500 dynamic page requests per second?

C. Update requests use a single master process to coordinate access, and a set of slaves (which can be distributed across machines) to actually do the work. Each update request requires 100 million operations on a slave, but also causes 10 million operations to be executed in the

master. What is the optimal configuration to support 400 updates requests per second? How about 1000 updates per second? How about 2000 updates per second?

Solution:

Key observation: In terms of average cost (per op), Cloudr-Basic is the cheapest and Cloudr-Ultra is the most expensive.

A. Since each request takes only 10M ops, they can be handled within 100ms. So in this case we should scaling out with as many Basic machines as necessary. For 1000 requests, we need 20 Cloudr-Basic machines and for 5000 requests, we need 100 Basic machines. (6 pts)

B. Since each request needes 500M ops now, the Basic (1s) and Pro (250ms) machines can no longer satisfy our needs. The next cheapest one is Cloudr-Enterprise, which we need 7 to handle 100 requests per second. For 500, we need 32 such machines. (6 pts)

C. Must scale up master (because there is only one master), Scale out slaves. Note that the master can be used to do slaves' work as well. Basic can't meet latency, so scale out with Pro machines.

   400 update requests --> need 4G ops for master, 40G ops for slaves, so 1*Enterprise + 18*Pro; (1*Enterprise + 20*Pro is also accepted) (3 pts)
   1000 update requests -->1*Ultra+47*Pro; (1*Ultra + 50*Pro is also accepted) (3 pts)
   2000 update requests --> need 20G ops for master. This is impossible. (2 pts)

Question 3 (20 points)

Consider a rack-mounted cluster of machines that is used for executing a very long-running scientific application. When any machine in the cluster fails, the operator performs the following repair steps:

01. The entire rack of machines is powered off.
02. The faulty machine is replaced.
03. The rack of machines is powered up.
04. The user application software performs crash recovery, and then continues execution.

On average, there is a failure of some machine in the cluster once every 80 hours. Nothing else in the cluster fails. Steps 01 through 03 above take two hours in total. The time for Step 04 has a normal distribution, with a mean of four hours.

A. What is the availability of the cluster? Also, what is the yearly failure cost if each hour of downtime costs $92,500?

B. Suppose, the cluster is modified to support rapid recovery after a crash. Upon failure of any machine in the cluster, the following steps happen:

B-01. Software failover to non-failed machines is automatically invoked.

B-02. In parallel with B-01, service personnel can remove the failed machine without disrupting the rest of the cluster by using a hot-swap capability. They can then repair and re-insert the machine into the cluster, and then power it up.

B-03. The powered up machine joins in application execution with no further delay.

Step B-01 (failover) takes 10 minutes and is the only time when during which the application is not running. What is the availability of the cluster now? What is the annual cost due to failures?

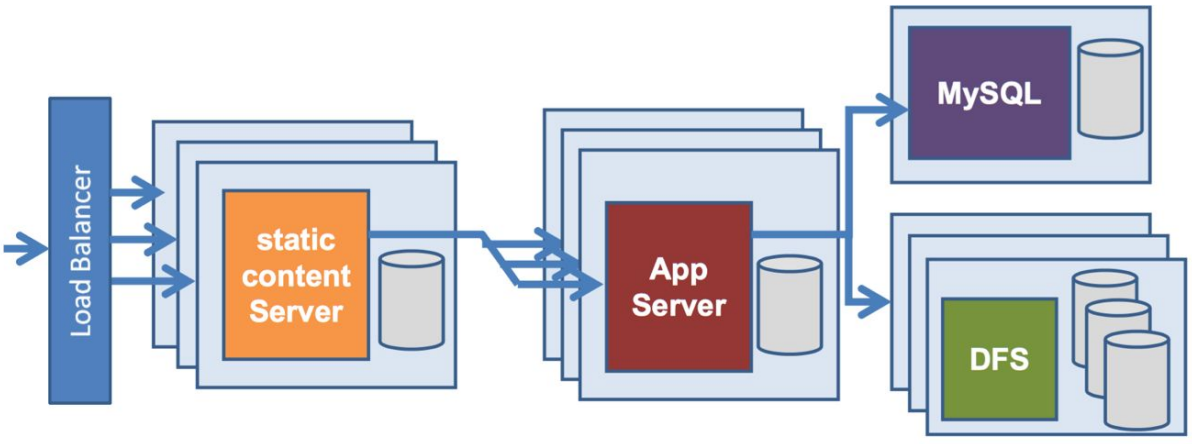
Solution:

A. MTTF = 80 hours. MTTR = 2+4 = 6 hours. Therefore the availability is MTTF/(MTTF+MTTR) = **93.02%**. The total cost is then 365 * 24 * (1 - 93.02%) * 92500 = 56,532,550 ≈ **57 million**.

B. MTTR = 1/6 hour now. So the availability is MTTF/(MTTF+MTTR) ≈ (80/80.16667) ≈ **99.79%**. The new total cost should be 365 * 24 * (1 - 99.79%) * 92500 ≈ **1.68 million**.

Question 4 (20 points)

Consider the 3-tier Web system shown below, which is similar to what was discussed in class. You are using this for an online bookstore. For each book, the MySQL database stores dynamic information about its current inventory and current price, as well as static information such as title, author, publisher, and year of copyright.



A. On Cyber Monday, 2017 at 00:00 AM, you start a sale. There are 160 user requests arriving at your system per second to buy books. You have 50 tier-1 servers (i.e., static content servers) and 50 tier-2 servers (app servers). 80% of the requests are for browsing, and take 0.1s for an

app server to process. 20% of the requests are purchases. 95% of the purchases can be processed by an app server in 1.0 seconds, but their distribution is long-tailed: 4% of the purchases take 5.0 seconds to process, and the remaining 1% take 10 seconds to process. You may ignore the time for processing by tier-1 servers (content servers). You may also ignore network delays and assume that no requests are dropped. Under these assumptions, what will be the queue length in front of each tier-2 server (i.e., an app servers) 3 minutes after the sale starts?

B. After many days of debugging, you discover that the long tail of app server response times is due to the MySQL database. A very complex real-time pricing model leads to complicated query processing in the database. You would like to scale out the database to shorten the tail of app server response times. However, your friend warns you that the database tier is considered to be the most difficult to scale out. State and explain two problems that might occur if you try to dynamically scale out the database across multiple machines.

C. Since dynamic scale-out of the database tier is risky, you decide to settle for static scale out. You split the book inventory onto 26 database servers according to the first letter of the book title. Now each server is in charge of only a small portion of the book inventory. However, in some cases, the problem is not fixed and tail latencies are still very long. What could be a possible reason for this? How would you fix this?

Solution:

A. The average time to process a request can be calculated as:

80% * 0.1 + 20% * (95% * 1 + 4% * 5 + 1% * 10) = 0.33 s/request

Each app server can process 1/ 0.33 = 3 request / s
50 app server can process 150 requests / s.
3 minutes there will be (160 - 150) * 3 * 60 / 50 = 36 requests in queue for each app server

B. DB contains all the inventory and history, so it has to be consistency across the who application.

Problems: user may see old record because of consistency.
Transaction may fail or block for a long time if two phase commitment is used.

C. Load imbalancing. The letter is not a good key. Use hash instead.

Question 5 (20 points)

Consider two simple transactions that execute concurrently on two different nodes. Each transfers one hundred dollars to the account A. Initially, there are one thousand dollars in the account A. The steps executed by the two transactions at Node 1 and Node 2 are shown in the table below.

Time	Node 1	Node 2
1	START_TRANSACTION	START_TRANSACTION
2	X = read value from account A	Do nothing
3	Do nothing	Y = read value from account A
4	X = X + 100	Y = Y + 100
5	set value of account A to X	Do nothing
6	Do nothing	set value of account A to Y
7	Do nothing	Do nothing
8	Do nothing	Do nothing
9	END_TRANSACTION	END_TRANSACTION

Assume your transaction system uses intentions lists to implement transactions.

A. Suppose Node 1 crashes at time 6. When your system recovers from the crash, what should the intentions list at node 1 look like? How should your crash recovery handle this intentions list?

B. Suppose the Node 1 delays its read of account A (currently at time 2) to time 7. The order of the following steps at Node 1 are preserved, so the transaction commits later. What will be the amount of money in account A after both nodes have ended their transactions?

C. Is this code correct in the absence of crashes? If your answer is yes, give an informal proof. If your answer is no, give your reasons and briefly explain how you would fix the code to make it correct.

Solution:

A. Intentions list should be X=1100. Will discard/abort it because the DONE flag hasn't been set, which means the whole transaction is considered incomplete.
B. 1100, since node 2 hasn't yet committed when node 1 attempts to update its value. It will try to commit with value 1100.
C. No. Clearly, synchronization is needed in this case, and the account balance is considered a critical section that should be carefully handled. One possible solution is to apply mutex or lock

mechanism so that only one node at a time is permitted to execute a whole transaction. Other solutions may be accepted, if with correct reasoning.