Faculty of Engineering and Applied Science

SOFE 4790U Distributed Systems

Homework #2

Name	Student #
Owen Musselman	100657709

Note: Changes made after group discussion are highlighted in red.

Part 1: Networking:
Review your course on computer networks, can you try to summarize what you learned in that course? Focus on the 7 layers of the OSI interconnectivity model? What different detailed functionality does each layer provide? How does this help two applications to communicate?

In the computer networks course, the content learned was the OSI model, the hardware used in creating the networks like routers, switches, hubs, modems, cables (ethernet, coax, fiber), calculating the best path to traverse the network, error checking by using checksums. DHCP and caching servers were also covered which would assist in load balancing and latency.

The Open System Interconnection (OSI) is a seven layer model that is used to communicate over a network. The OSI model's seven layers are as follows:

1. *Physical Layer*: Transmits the raw bit stream over the physical medium, like cables frequencies, pins. Layer also determines throughput latency, and error rates.
2. *Data Link Layer*: Node to node data transfer which is between two connected nodes. Additionally, the data link layer deals with error correction. Interestingly, this layer also has two sub layers which are the Media Access Control layer and the Logical Link Control layer (MAC and LLC respectively). MAC is responsible for interaction with the hardware interactions. LLC is responsible for flow control, and automatic repeat requests.
3. *Network Layer*: Where the layer is responsible for forwarding packets from router to router. This layer assists in increasing the efficiency of packet sending/receiving by finding good routing paths.
4. *Transport Layer*: Layer is responsible for breaking the data from the Session layer (Layer 5) into different segments. This layer also reassembles the segments on the receiving end and puts them together back into the data it was so that it is understood by the Session layer. This layer is also responsible for sending data at a correct rate so that it matches the speed of the device that is receiving the data (flow control). This layer will also error check and request for data to be resent in the case of an erroneous transmission.
5. *Session Layer*: Communication between devices and sessions. It will ensure that a session is open while data is being transferred (send or receive), this layer will also close the session after the job is finished. Checkpoints are an important part of this layer as if the session is stopped the device can start the transfer from the latest checkpoint that was made.
6. *Presentation Layer*: Layer defines how devices should keep their messages secret (encrypted) along with compressing data to ensure its proper transmission. Any data from the application layer is taken here and it is prepared to have it transmitted to the lower layers.
7. *Application Layer*: This layer gets data from the user directly through software like web browsers. This layer provides communication through the lower layers to make connections with different applications.

The OSI model helps two applications communicate by having each layer add its information in the form of headers. Each layer is seen to handle messages from the layers above and below it which is accomplished by protocols. So each layer sends the processed data to the next layer, performs error checking like checksums to ensure the data is correct, ensuring proper communication between applications.

Part 2: Virtualization

Compute Resource Consolidation

Where websites, microservices and virtual machines are running in their own environment, causing inefficient resource usage even when idle, increasing management and runtime costs.

Tasks are able to be grouped together by means of scalability, processing capabilities and their lifetime. When this is done, they are able to scale together. A grouping method to be considered is tasks that need large amounts of CPU power in shorter bursts, and these could be grouped together into a single computational unit. It is noteworthy that there is balance between the need of keeping expensive resources being used and them being over stressed, as the more compute heavy tasks that are longer should not be grouped together as the resource is both being stressed, and used for extended periods of time.

- Scalability & Elasticity: Avoid grouping any tasks that are going to have scalability needs that will conflict within the same compute unit.
- Lifetime: Design tasks with checkpoint system which would let them halt operations in a clean manner, and then allow them to resume their operation before they are halted and the compute unit is restarted. Additionally, when multiple longer tasks that are in a compute unit it will more than likely be necessary to stop the unit from being recycled before these tasks are completed.
- Release cadence: When the config or the implementation of a task is altered frequently, it is most likely that the compute unit needs to be halted so that it can be reconfigured with these new changes and then restarted. If this is done, every other task within that compute unit needs to be restarted as well.
- Security: If there are multiple tasks in the same compute unit they could possibly share access to the same resources. In this case there needs to be a high trust between these tasks, as there is the possibility of having one task corrupt the operation of another task. Each task in the same compute unit is only as secure as the one with that is most vulnerable.
- Fault tolerance: In the case that one task within compute unit malfunctions or fails (even if there are multiple), it can have an effect on the other tasks within that same compute unit.
- Contention: Tasks should not have to compete for the same resources if they are in the same compute unit. For example two heavy computation tasks should not be grouped into the same compute unit as they would be playing tug-of-war with the resources.
- Complexity: When there are various tasks on the same compute unit, the complexity is increased, making the testability, debuggability, and the maintainability less than desired.

- Stable logical architecture: The code for each task should not need to be altered, even in the case of the physical environment it resides in has been altered.

Compute resource consolidation should be used when the tasks needed to run cannot do so cost effectively in their own separate compute units. For example tasks that spend vast amounts of time idle should not be in their own compute unit. This method of resource consolidation can potentially not be suitable if tasks are performing actions with sensitive data, or critical operations, in this case they are better off running in their own separate compute unit.

Data Partitioning

In distributed systems it is sensible to logically partition the resources into separate computational units, but this can lead the system to consume more power when not needed, like when idle. This problem is solved by using Compute Resource Consolidation and by doing so we can increase utilization, and decrease overheads and costs.

There are various reasons to perform data partitioning some are as follows:

- Improve scalability: Databases will eventually reach their limit in terms of hardware, this can be solved by having separate partitions located on different servers.
- Improve performance: performance is increased by having access operations occur over a smaller volume of data, like breaking up a large dataset into smaller sets instead of having to search through a single massive dataset.
- Improve security: More sensitive data can be separated in their own partitions which can then have different security measures put upon it.
- Provide optional flexibility: Can minimize the cost of operations, or maximizing the administrative efficiency. Additionally strategies used could be for management, backups/restoring, monitoring.
- Match the data store to the pattern of use: Partitioning lets each individual partition to have a different type of data store. In other words, files could be stored using a document database, and binary data would be put into blob storage.
- Improved availability: If the data is partitioned across multiple locations/serves, there is no single point of failure.

There are different types of partitioning methods three of which are:

- Horizontal partitioning (sharding): Each partition has its own data store, but those data stores have identical schemas. Each partition here has its own subset of the data. This reduces the number of rows in the tables since the tables are now spread out. The difficulty of partitioning horizontally is the sharding itself, as we want the databases to be accessed evenly as well as distributed evenly.
- Vertical partitioning: Kind of the opposite of Horizontal partitioning, where the partitions hold a subset of the field items in the data store. Where each of these fields are divided in terms of their pattern of use like how frequently they are used.
- Functional partitioning: Data is aggregated in terms of how it is used. Like in the case of an invoice, it would be stored in one partition and the stock of a product would be stored in another partition.

The above strategies can be combined, for example Functional partitioning and Vertical partitioning can be combined for a more specific use case.

Cloned VMs (SnowFlock)

Cloned virtual machines are missing most of their memory state by the time they are created by the descriptor:

Memserver process stops the VM so that its memory can settle. Once copied, clones can reconnect to the external world in their new enclosed VM. The memserver provides the bits of information the clone needs from its parent. Copy on write is used to circumvent the issue of corrupted memory, where the writing access from pages in memory are removed. A page fault will occur if a write is tried, and that notifies the hypervisor to copy the page. Importantly the parent VM can write to that page in order to keep running, but the memserver needs to use the copied read only page.

Mcdist is used to distribute a packet to various receivers. It does this through IP multicasting which allows for network hardware parallelism, which in turn lowers the memory server load. Mcdist does not need to be a reliable method as the packets do not need to be received in a specific order, where the server multicasts for responses, and the clients timeout if/when they do not get a reply for a request they made and have to try again. There are 3 optimizations of mcdist:

- Lockstep: when temporal locality occurs, various clones will request the same page, and it will be done almost at the same time, and mcdist will ignore all but requests but the first one that reaches.
- Flow control: The server will limit its sending rate to be a weighted average of the client's receive rate. If this is not done, then the receiver will be overwhelmed with too many pages sent from the server.
- End game: At the time the server has transmitted the majority of its pages, the server will go back to unicast responses. Usually anything at this point are requested retries, and having retries being sent to every clone is a waste.

With the use of lLazy propagation, virtualization, page faults, copy on write, multicast. With all of these present it is possible to reduce the stress on the cloud servers as they do no need to reboot new instances continuously. This helps with serving web pages, and parallel computing as well.

Generic Questions:

- What resources are managed and how?
  The resources managed in the system are CPU, memory, storage, VMs. They are managed through data partitioning, vm cloning, and resource consolidation.
- How does the system described in the article provide the following requirements:
  - Resiliency in case of failures: Data partitioning can allow for no single point of failure, allowing for other partitions to be in use if another fails. If a VM were to fail a new clone can be made in a respectable amount of time to fill in for the failure.
  - Scalability in the case of expansion in various parameters: More VMs can be cloned if needed (scalable and elastic). With data partitioning it allows the data to be hosted on separate servers. With consolidation, compute units should not have conflicting scalability requirements within them.
- How are transparencies provisioned within the described image sharing system?

Transparency within data partitioning is that the users do not know how or where the data is partitioned, all they know is that it is a seamless experience and it seems like it is a single implementation. For VM cloning with lazy state replication, the users do not see what VM they are in. These cloned VMs with lazy replication can increase the efficiency as only the state that is used is copied, but the user is not concerned, just if the service is usable. As for compute resource consolidation, when a user is actively using the system, they don't see or care to know what compute unit they are utilizing, they only want their workload to be dealt with, hence what resources are apart of the compute unit, and what compute unit they are using is hidden from the user.