CMPSCI 677 Distributed and Operating Systems **Spring 2018**

Lecture 7: Process, Code and VM Migration

Lecturer: Prashant Shenoy *Scribe:* ***Amber Madvariya***

7.1 Server Design Factors

7.1.1 Address Resolution

How does client figure out the port number to send the request to? For some standard services like ssh and https, the ports are standard like 22 and 80 respectively. However, for custom endpoints this is not possible. One obvious way to identify port is to hard-code it in the client code, however this is rarely feasible and is not advised. Directory services is the most standard way for port resolution in most cases (this is implemented in nginx). Sometimes for large clusters, having a meta-server dedicated exclusively for directory services is also advisable.

7.1.2 Request Processing

There are two types of request processing methods in modern servers:

- **Iterative**: The server sits in a while loop serving requests. This method doesn't enable any concurrency and all requests are queued. This method is preferred in single-core single process servers.
- **Concurrent**: The server executes each request as a separate thread. Preferred in multi-core multi-process systems.

7.1.3 State

: Servers can be classified into three categories in terms of state:

- **State-full:** Maintains state for clients in memory or cache. Can be used to maintain sessions of user activity.
- **Stateless:** Doesn't maintain any state for clients.
- **Soft State:** Maintains state for a certain amount of time by persisting data in cache. Discarding the existing state does not change the correctness of the system.

7.1.4 Clustered Servers

For high volume services, scaling is achieved through tiering/clustering of servers. Each tiered level may be replicated and load is balanced across the clusters through a load balancer which assigns each incoming request to a server in the cluster. There are two methods of request assignment in clusters:

- **Round Robin:** Requests are assigned in a round robin fashion. This approach isn't ideal or optimal for implementing user based sessions (activity by a single user in a small continuous time period).
- **Session based:** Requests from a user session are routed to the same server in order to maintain state about that session.

For communication between the system and the client in a multi-tiered architecture, there are two methods:

- **Switch Server:** Tier 1 hands off the client request to a server in the cluster and all further communication between the client and the server happens directly.
- **Strict Multi-tier:** All the communication between the client and the cluster happens via tier 1.

7.2 Server Architectures

There are primarily four type of server architectures:

- **Pure-Sequential**: Serves request one at a time in a single process server. Requests are queued in this case.
- **Thread based Concurrent:** Spawns a new thread for each request.
- **Process based Concurrent:** Spawns a new process for each request.
- **Event-based Sequential:** Single process server, but is asynchronous and non-blocking to I/O. Since, I/O is not blocked while a request is being processed, other requests can be processed simultaneously.

Out of the four types, pure-sequential, thread based and process based are relatively easier to program as compared to event-based sequential.

Efficiency:

For a multi-core multi-process server, the efficacy of the four types can be ordered as:

thread based > process based > event based > pure sequential

Thread based, process based and event based are better than pure sequential because of concurrency. Thread based and process based are better than event based because of higher parallelism. Thread based is better than process based because of low context switch overhead among threads as compared to processes.

For a single core single process server, the efficacy can be ordered as:

event based > thread based > process based > pure sequential

While the reasoning for the ordering of thread based, process based and pure sequential is the same as for the previous one, the efficacy for event based is highest in this case because of no context overhead and concurrency.

7.3 Process, Code and VM Migration

The motivation behind developing techniques for code, process and VM migration is that migrating these components of a system helps in improved performance and flexibility.

There are two types of migration models:

- **Process Migration:** Also known as strong mobility, this includes the migration of all the components of a process i.e. code, resources and execution state. Restarts from last saved state upon migration and involves significant transfer of data over the network.
- **Code Migration:** Also known as weak mobility. In this model only the code is migrated and the process is restarted from the initial state on the destination machine. The network transfer overhead is low since only the code is transferred. Sender initiated migration is where the sender sends the code e.g. search queries whereas receiver initiated migration involves the reciever sending the code e.g. app requesting JS/Java code from server.

To decide whether to migrate a resource attached to a process or not, we look at the binding of the resource to the process. There are three types of resource to process bindings:

- **Identifier:** Tightest possible binding. E.g. files
- **Value:** Not as tight as Identifier. E.g. libraries used in Java
- **Type:** Weakest binding, not necessary to be transferred as long as functionality is fulfilled. E.g. local devices.

Apart from the resource binding, it is also necessary to look at the cost of moving resources which can also be classified into three categories:

- **Unattached:** Very low cost of moving e.g files
- **Fastened:** High low cost of moving e.g databases
- **Fixed:** Can't be moved e.g. local devices, common end points.

The migration actions depending on the resource binding and their cost of transfer are tabulated below for different combinations:

	Unattached	**Fastened**	**Fixed**
By Identifier	MV (or GR)	GR (or MV)	GR
By Value	CP (or MV, GR)	GR (or CP)	GR
By Type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

where GR means establishing global system-wide references, MV means moving the resources, CP means copying the resource and RB means rebinding process to locally available resource.

7.3.1 VM Migration

Process and code migration in heterogeneous systems (i.e. destination architecture is different than source architecture) pose a challenge. Migration via interpreted codes is one possibility but it is often clumsy in practice and is almost never used. Moreover, interpreted code migration only supports weak mobility. In such cases, it is beneficial to look at techniques for VM migration. VMs can be migrated from one machine to another irrespective of architectural differences, without any noticeable down-time. There are two methods for VM migration:

- **Pre-copy Migration:** The process of pre-copy migration can be listed down as:

  1. Copy all memory pages to destination
  2. Copy memory pages which were changed during the previous copy
  3. Repeat step 2 until number of memory pages is small.
  4. Stop VM, copy rest of memory pages at destination and start VM at the destination.

  The IP address of the source is also copied to the destination (since IP address is a logical entity). All I/Os are assumed on disk, otherwise disk resources also need to be copied.

- **Post-copy Migration:** The process of post-copy migration can be listed as:

  1. Stop VM and move basic amount of pages to destination
  2. Start executing on new machine
  3. In case of page faults in the new VM, copy page from the source machine.

  Migration is very quick and the pages which were not transferred initially can be fetched asynchroously in the background.