L05c. Latency Limits

Scan me

Introduction:

- Network communication is a key factor in determining the performance of a distributed system. Hence, the OS has to reduce the latency for the network services.
- Latency vs. Throughput:
  - Latency is the elapsed time for an event.
  - Throughput is the number of events that can be executed per unit time.
  - Higher bandwidth means higher throughput but doesn't necessarily result in lower latency.
- Overhead on RPC:
  - Hardware overhead: How the network is interfaced to the CPU.
  - Software overhead: What the OS takes to prepare the message for transmission.

RPC Latency:

- These are the steps needed to perform an RPC:
  - Client call: Setting up the arguments for the procedure call, and makes the call to the kernel. The kernel validates the call, marshales the arguments into a network packet and sets up the controller to do the transmission.
  - Controller: The controller moves the message to its own buffer, and then put it on the bus.
  - Time on wire: Depends on the available bandwidth.
  - Interrupt handling: The message arrives to the server as an interrupt. The OS moves the message to the server's controller buffer, and then to the node's memory.
  - Server setup to execute the call: Locating and dispatching the server procedure, unmarshel the arguments from the network packet.
  - Server execution: The server executes the call and sends the results to the client in the same way described above.
  - Client setup to receive the results and restart execution.

Marshaling and Data Copying:

- Making an RPC call involves three data copies:
  - First copy: The client stub takes the arguments of the RPC and converts it to an RPC message.
  - Second copy: The kernel has to copy the RPC message from the memory space it resides on to its own buffer.
  - Third copy: The network controller will then copy the RPC message from the kernel buffer to its interneal buffer using DMA. This is an unavoidable hardware action.
- The copying overhead is the biggest source of overhead for RPC latency.

- To reduce the number of copies, one of two approaches can be used:
  - Eliminate the stub overhead by moving the stub the stub from the user space to the kernel and marshaling directly into the kernel buffer.
  - Leave the stub in the user space, but have a shared descriptor between the client stub and the kernel. This descriptor provides information to the kernel about the layout of the arguments on the stack. The kernel these information to create a similar layout in on its space.

Control Transfer:

- This includes the multiple context switches that have to happen to execute an RPC call:
  1. When the client makes and RPC call, it blocks till it receives the results. The kernel makes a context switch to serve another process. This is important to make sure that the client node is not underutilized.
  2. When the call arrives to the server's side, it has to make a context switch from the process it's currently serving to the incoming interrupt (RPC call). This is essentially to the RPC latency.
  3. After executing the call, the server will switch to another process.
  4. When the results arrives at the client's side, the kernel has to switch to serve that client. This is essentially to the RPC latency.
- Only context switch #2 and #4 are critical to the RPC call itself.
- We can reduce these switches to two by overlapping context switch #1 with #2, and #3 with #4.
- We can also eliminate context switch #1 if the we know that the RPC call will not take much time. We can spin waiting for the RPC results.

Protocol Processing:

- Given that we have a reliable LAN, the following approaches can be used to reduce latency in transport:
  - No low level acknowledges.
  - Using hardware, instead of software, checksum for packet integrity.
  - No client side buffering. Since the client is blocked, it can resend the call if the message is lost.
  - Overlap server side buffering with result transmission.