CSE221 Lecture 17

Aronya Baksy

December 2024

1 Datacenter Networking

- High bandwidth (100-400Gb/s), high speed storage tech (flash storage)
- Any networking stack requires min overhead in software (in a 100 Gb/s network, a 1500b packet can arrive every 120ns, compare this to a syscall time or a ctx. switch time)
- **Kernel Bypass**: NICs write packets directly to reserved buffers in main memory (using DMA engines).
- **Data Path**: path taken by packets when they're being sent/received, **Control Path**: code path used for connection mgmt/mem alloc/core alloc/other book-keeping tasks
- Main concern with kernel bypass stacks: isolation between applications, this is solved using *virtual send/recv queues* per application or **IOMMU** which allows apps to use virtual addresses and translates to phy. addr. for NIC.

2 IX: A Protected Dataplane OS for High Throughput and Low Latency

2.1 Goals

- Implement a network stack that trades off between high throughput, low latency, strong protection (only in kernel, since user level apps can crash network stacks in user space) and resource efficiency.
- Demonstrate increased performance over standard Linux n/w stack

2.2 DataCenter Challenges

- Low tail latency
- High packet rates, large number of concurrent connections and high connection churn
- Isolation between applications
- Resource efficiency

2.3 Alternate Approaches

- **User-Level Stack**: tough to implement security, uses more hw threads irrespective of actual load, high latency due to batching of network requests between applications
- RDMA as an alt. to TCP can reduce latency, but requires specialized adapters be present at both ends of the connection.

2.4 IX Apparoach: Control Plane and Data Plane

- Separate the control plane, which is responsible for system configuration and coarse-grained resource provisioning between applications, from the dataplanes, which run the networking stack and application logic.
- designed around a native, zero-copy API that supports processing of bounded batches of packets to completion

- each dataplane runs a single application in a single address space linked with a library OS (similar to Exokernel, but uses hw virtualization for security, have direct access to NIC queues through memory mapped I/O)
- IX uses **run to completion during all load conditions**, use polling and avoid interrupt overhead by dedicating cores to the dataplane
- Use **adaptive batching**: batch only during congestion, upper bound on number of batched packets
- Amortize syscall overhead since you can use one syscall for a batch of packets
- dataplane can also monitor queue depths at NIC edge and request additional resources to prevent packet buildup
- **Zero copy API**: dtaplane kernel and application communicate at coordinated transition points via messages stored in memory, (downside: apps need to release buffer space)
- **Synchronization-free** since all h/w resources (NICs) are partitioned between cores, all shared data structures use RCU locks,
- Use receiver-side scaling to split incoming packets between cores (hash of packet headers), ensure that one core processes all packets for a single connection

2.5 Evaluation

- Outperform Linux in microbenchmarks in terms of throughput and latency
- Scales up to 100ks of active connections
- Real world application test: memcached dist. key value store, 2x tail latency reduction, 3.6x higher throughput

3 Snap: a Microkernel Approach to Host Networking

3.1 Goals (apart from IX)

- Developer velocity, fast deployment and upgrade
- Modularity

3.2 Architecture

- Threading using kernel-level threads, each thread is composed into an **engine**
- Threads communicate using IPC, avoids context switching and makes better use of multicore architectures
- CPU scheduling: dedicated cores, spreading engines, compacting engines
- Last two schedulers use *MicroQuanta* scheduling class which modifies Linux kernel to give priority to SNAP threads

3.3 Upgrade Process

- Start new SNAP instance
- Brown out phase: write state to memory, new SNAP instance reads the state
- Black out phase: stop old engines, transfer latest state to new engine, restart new engines and finally kill old instance
- Entire process takes around 250ms, which is acceptable overhead for an upgrade