An Operating System for Disaggregation with Coherence

Erika Hunhoff (Student)
University of Colorado Boulder

Gerd Zellweger
Feldera (work performed at VMware Research)

Eric Keller
University of Colorado Boulder

Introduction

Disaggregation:

- Assign resources as needed
- Useful for utilization, scaling, failure domains, etc.
- RDMA, CXL 1.0, etc. for memory disaggregation

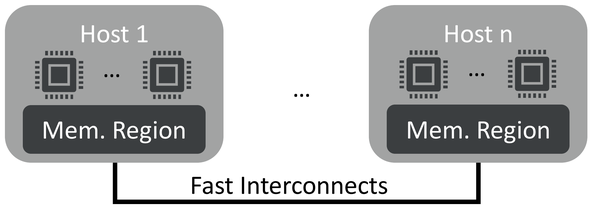

Figure 1. A rack with hardware accelerated memory coherence.

With disaggregation, resource management tasks are not constrained by a single host. Should the OS also go beyond a single host?

Goals

◇ Hardware Advances

- Use coherent shared memory to support the OS, not just applications

◇ Scalability

- Allow a process to scale beyond the bounds of a single host for both compute and memory

◇ Scheduling & Locality

- Support on-demand allocation for both compute and memory
- Co-locate process resources when possible

Distributed NrOS (DiNOS) Architecture

The **data plane** consists of one or more **dkernels** (data kernels) based on NrOS [1].

1. Log replicas (similar to NR [2]) are used to keep process state consistent.

2. Dkernel resources (cores, slices of memory) are assigned to processes by the control plane.

3. Control operations are forwarded via RPCs using shared memory queues.

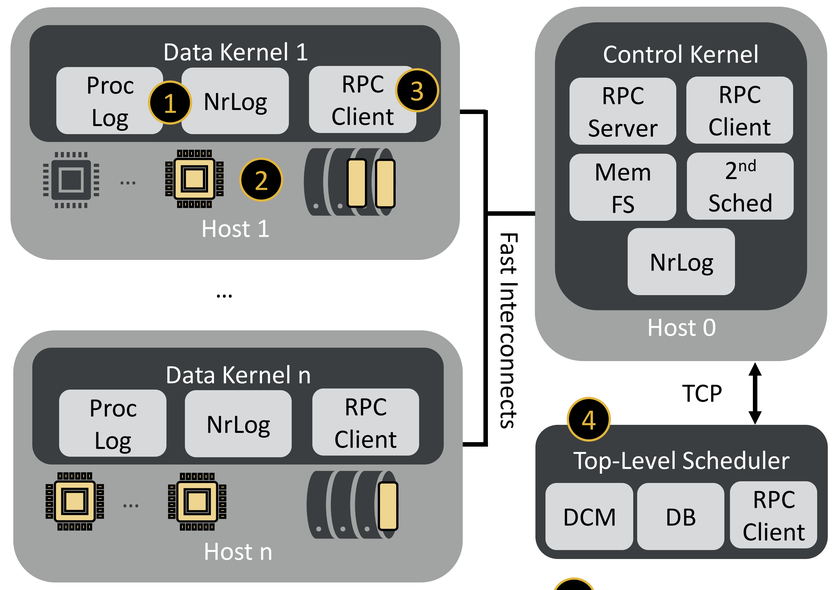

Figure 2. Architecture of distributed OS for disaggregated coherent memory.

The **control plane** consists of one **ckernel** (control kernel) based on NrOS [1] and a **scheduler** built using DCM (Declarative Cluster Management) [3].

4. The scheduler uses DCM to encode the scheduling problem as an optimization problem.

5. A process (gold) may be assigned to use resources from different dkernels.

Preliminary Microbenchmark: `map()`

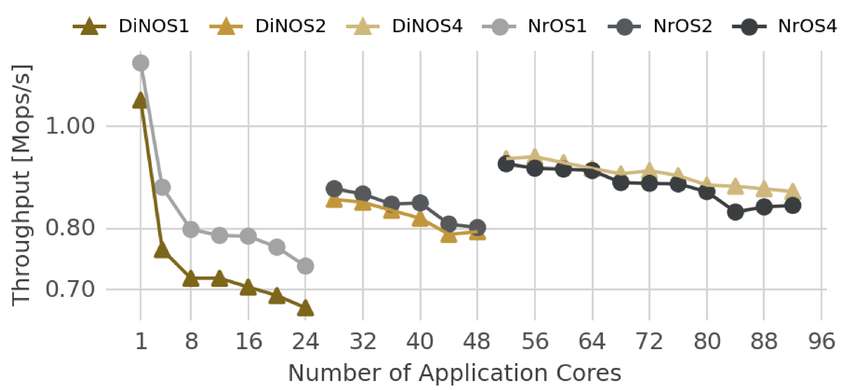

Figure 3. Throughput of `map()` in million operations per second for 1, 2, and 4 dkernels for DiNOS (distributed NrOS) compared to NrOS with 1, 2, and 4 log replicas as the number of data plane cores (x-axis) increases.

Design and Implementation

◇ Prioritize Usability

- Run largely unmodified Unix binaries
- Does not require bespoke framework
- Transparent scaling (application hints?)

◇ Implementation

- Written in Rust, extends NrOS [1]
- Emulated: QEMU & QEMU shared memory

References

[1] A. Bhardwaj et al. NrOS: Effective replication and sharing in an operating system. In OSDI, 2021.
[2] I. Calciu et al. Black-box concurrent data structures for NUMA architectures. SIGPLAN Not., 52(4), 2017.
[3] L. Suresh et al. Building scalable and flexible cluster managers using declarative programming. In OSDI, 2020.

Future Work

- Optimization & benchmarking
- Dynamic log replication