Bolt

FaaS with Disaggregated Shared Memory

Adarsh Patil
Vijay Nagarajan, Nikos Nikoleris

THE UNIVERSITY *of* EDINBURGH
informatics

arm

UK Systems Research Workshop 2021

What is Function-as-a-Service (FaaS)

- Latest category in Cloud Computing Services

"Serverless" *hides servers* : Provider's platform and infrastructure

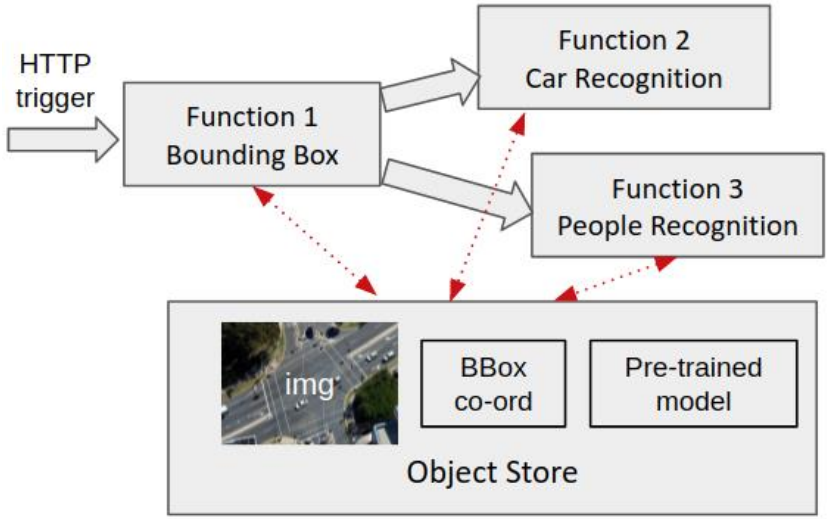
"Modular" *non-monolithic* : Decomposed into several standalone functions

- ⊕ Ease of development
- ⊕ Ease of deployment
- ⊕ Built-in scalability
- ⊕ Cost efficiency

- Fastest growing computing paradigm
  Providers: Amazon, Google, Microsoft, IBM, Alibaba, Oracle....
  Clients: Netflix, Guardian, T-Mobile, PayPal, P&G....

FaaS Applications

- State machine workflow of independent stand-alone functions (happens-before relation)
- Workflow of functions is scheduled by a runtime
- Dynamically instantiated and executed on-demand
- Event-driven execution – triggers, API, crons
- Stateless functions use remote object stores for input/output and ephemeral data stored
  Object-granularity, Strong Consistency Model
  GET reads the value of the last PUT
  last acknowledged PUT visible
- Re-execution of idempotent function, if compute node fails



An Example FaaS workflow
(AMBER Alert Pipeline)

Motivation: Problems with FaaS object stores

Data movement!

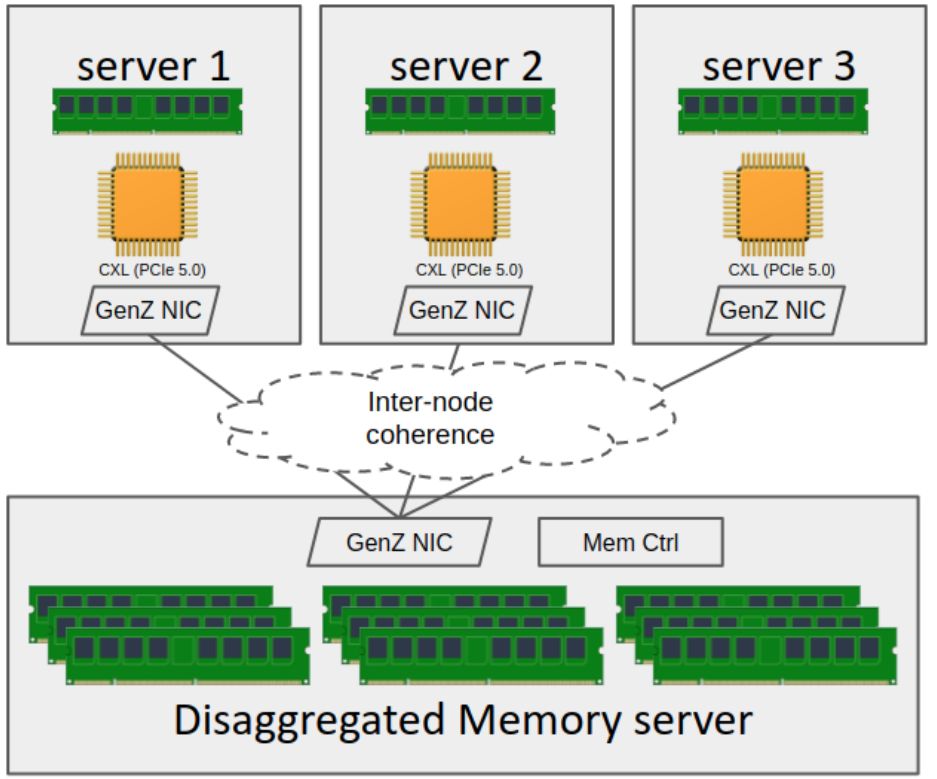
- ⊖ Object reads and writes are software managed
- ⊖ Explicit data duplication from object stores into compute nodes
- ⊖ High access latencies to object store servers over congested general-purpose eth networks

Can we do better?

- ? Shared memory semantics for object store – load/store
- ? Implicit object movement – on-access
- ? Handle all data movement in hardware – performance
- ? Next-gen datacenter network technologies – efficiency
- ? Incur minimal changes to the entire software stack – productivity

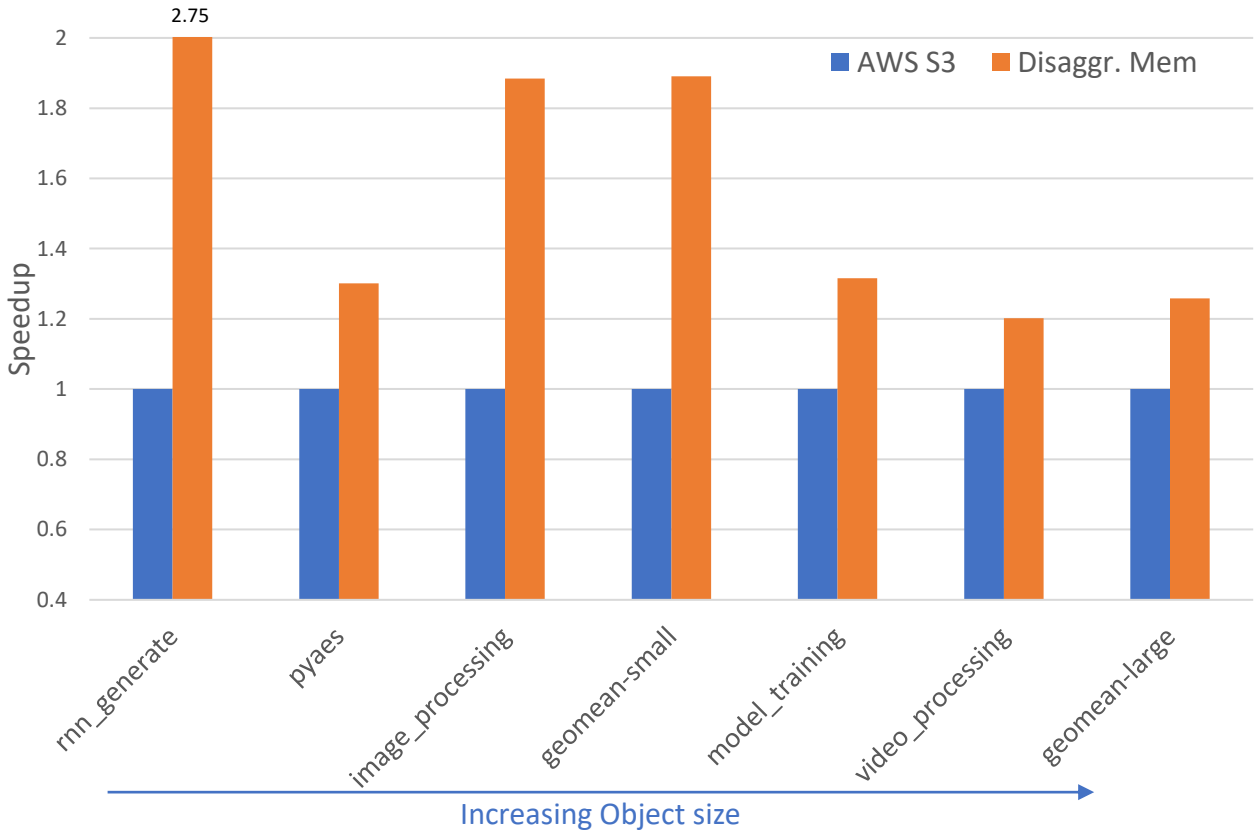
Motivation: The opportunity

- The "*disaggregated memory*" node
  Rack-scale shared memory node to house the objects
- Shared memory semantics
  Shared memory interface to communicate/pass objects
- A load/store interface
  Hardware-semantic memory fabric interconnect (GenZ, OpenCAPI)
- Implicit data movement
  Application transparent, managed by hardware
- Hierarchical management
  Separation of concerns between intra and inter node protocols



Proposed Disaggregated Memory setup enabled by CXL + GenZ

Motivation: Is there performance?



Single function execution: Disaggregated Memory vs AWS S3

- Simulation: SynchroTrace-gem5
- Caches: L1 I/D 64KB, 8-way; L2 16MB 16-way
- Memory: single channel, DDR4, 16GB
- Disaggregated Mem: 500ns one-way*, DDR4
- Benchmarks – CPU, mem intensive FunctionBench+
- Object sizes – small <50KB, large <2MB

89% and 25% geomean execution time speedup for small and large object functions respectively

*DeACT [HPCA '21] +FunctionBench[CLOUD '19]

Objective: Improving Performance of FaaS applications

How?

- Hardware caching
  Cache recently used data from memory node in SRAM/DRAM caches on compute node
- Hardware coherence
  (a) Inter-node coherence to enforce the object store memory consistency
  (b) GenZ NIC participates in the coherence (not behind the IOMMU)
  e.g., CXL.cache
- Hardware fault-tolerance
  Multiple independent, loosely-coupled nodes mandates fault-tolerance

Can and should objects be cached? Yes!

Field Data Study of Function-Object Accesses *

Object sizes

- Median: 28B, 3rd Quartile: 2.2KB
- Long tail of object sizes (max 1.2 GB)
- Objects read are larger than the ones written

Objects fit in onchip caches, but large objects do exist

Object access and reuse characteristics

- Majority of apps access single, different object per invocation
- 42% of the apps access the same object in more than one invocation
- Only 11% of apps access more than 1 object per invocation

Objects reused throughout the application workflow

Objects temporal access pattern

- Accesses to a large percent of objects are very bursty (Poisson distrib)
- 15% of applications account for 99% of the invocation
- 30% of functions access the same data across all invocations

Objects have good temporal locality

*Azure Public Dataset (github)

Challenges: Architecting Coherent Disaggregated Memory for FaaS applications

- Shared Memory interface across nodes
- Scalable, low-latency coherent disaggregated memory
- Failure resilient disaggregated memory

Design Requirements (specifications)

1. Shared Memory interface across nodes
   - Application transparent
   - Map objects into address space of functions (processes) running on same or different compute nodes
2. Scalable, low latency coherent disaggregated memory
   - Cognizant of underlying inter-node interconnect characteristics – flit size, packet ordering, topology
   - Specialized for FaaS functions data sharing characteristics – function scheduling/communication, object read/write, temporal accesses characteristics
3. Failure resilient disaggregated memory
   - Compute node failures
   - Failures during object writes (network failure)
   - Disaggregated memory node failure

Implementation Particulars

1. Shared Memory across nodes — Enablement
   - Encapsulates workflow into single unified orchestrator function [Faastlane, ATC '21]
   - Objects passed between function processes through shared memory inter-process communication (IPC)
   - Extend shmem IPC to multi-node with addr mapping/translation [DeACT, HPCA '21]
2. Scalable, low latency coherent disaggregated memory — Evaluation
   - Coarse-grained coherence with write-through caches
   - Hardware/Software co-operation (Coherence Directory-FaaS scheduler)
3. Failure resilient disaggregated memory — Evaluation
   - Non-blocking coherence protocol
   - Object atomic, durable hardware transactions

Putting it all together: FaaS with Disaggregated Shared Memory

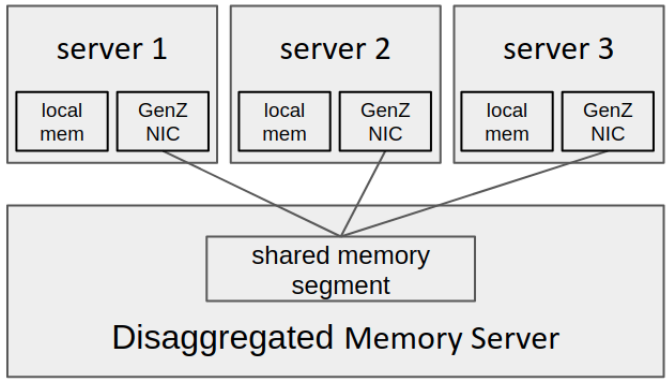
The disaggregated memory-based object store participates in 2 hierarchies of communication

Intra-rack coherence protocol

Scope of this work : “Bolt”

Memory-semantic interconnect GenZ/OpenCAPI

Hardware caching, Optimized coherence, Hardware fault-tolerance



Inter-rack consistency protocol

e.g., Amazon Dynamo++ [SOSP ’07]

General purpose ethernet

Software sharing / replication