Cloud-Native File Systems

Remzi H. Arpaci-Dusseau
Andrea C. Arpaci-Dusseau
University of Wisconsin-Madison

Venkat Venkataramani
Rockset, Inc.

How And What We Build Is Always Changing

Earliest days

- Assembly programming on single machines

Big single-machine advances

- Unix: A standard (and good) OS!
- C: A systems language!

Same thing, one level up: Distributed systems

- Collect group of standard machines, build something interesting on top of them

Commonality:
New System on Fixed Substrate

Whether a single machine/distributed, we tend to build **new systems** on a **fixed set of resources** with **fixed (sunk) cost**

- Machine: X CPUs, Y GB memory, Z TB storage
- Buy many such machines
- Build new system of interest on those machines

But the world is changing…

Welcome To Cloud

Cloud is a reality

- Can **rent** cycles or bytes as needed
- Per-unit **cost** is defined and known
- Not just raw resources: **services** too

Many new systems are being realized only in cloud

- Excellent example: **Snowflake** elastic warehouse [sigmod '16]

Thus, Questions

Cloud-native thinking:
How should we build systems given the cloud?

- What new opportunities are available?
- What new systems can we realize?
- What can we stop worrying about?

In This Talk

Cloud-native principles

- Guidelines for how to think about building systems in the era of the cloud

Cloud-native file system

- Case study: How to transform a local file system into a cloud-native one

Principles

Storage principles

CPU principles

Overarching principle

(just highlights; more in paper)

Storage Reliability

Storage reliability principle:
Highly replicated, reliable, and available storage can (should?) be used (The “S3” principle)

- 11 “9s” of durability!

Implication: Build on top of this, don’t build YARSS (Yet Another Replicated Storage System)

- Example (kind of): BigTable on GFS

Storage Cost and Capacity

Storage cost principle:
Storage space is generally inexpensive

- At cheapest, $4 / month / TB

Storage capacity principle:
A lot of storage space available

- “The total volume of data and number of objects you can store are unlimited” (Amazon)

Implication: Use space as needed to improve system

- Example: Indices for added lookup performance

Storage Hierarchy

Storage hierarchy principle: Storage is available in many forms, with noticeable differences in performance and cost across each level

- Example: Amazon Glacier vs S3

Implication: Must manage data across levels

- Can improve performance, reduce costs

CPU Parallelism

CPU parallelism principle (or A x B = B x A):
It should cost roughly the same to execute on A CPUs for B seconds as it does to execute on B CPUs for A seconds

- Granularity of accounting might limit you…

Implication: Do everything you can in parallel

CPU Capacity

CPU capacity principle:
Large numbers of CPUs are available

- As with storage, essentially “unlimited”

Implication: Use as many CPUs as you need

- Scale up to solve tasks quickly

CPU Scale-Up/Down

CPU scale-up/scale-down principle:
One should only use as many CPUs as needed for a task, and not more

- While cheap, CPUs are not free either

Implication: Must monitor usage, turn off CPUS when unused

CPU Remote Work

CPU remote-work principle:
When possible, use remote CPU resources to do needed work

- Shared data store makes this easier

Implication: Can separate foreground/background

- Improve predictability of former, use parallelism for latter

CPU Hierarchy

CPU hierarchy principle: CPU is available in different forms, with differences in performance, cost, and reliability across each level

- Normal vs. spot instance for example

Implication: CPU types must be managed

- Pick CPU right for given task

Overarching Principle

Overall performance/cost principle:
Every decision in cloud-native systems is ultimately driven by a cost/performance trade-off

- Can't make decisions without cost/perf knowledge
- Extremes are interesting:
  highest performance, or lowest cost
- But middle ground is important too:
  "reasonable" cost/performance

Implication: Cost must be fundamental part of systems (and even applications above)

Implications

Replicated storage: Don’t reinvent the wheel

Extra space is cheap: Use for performance?

Massive parallelism: Use for background tasks

Hierarchy: Continuous data migration to lower cost while keeping performance high?

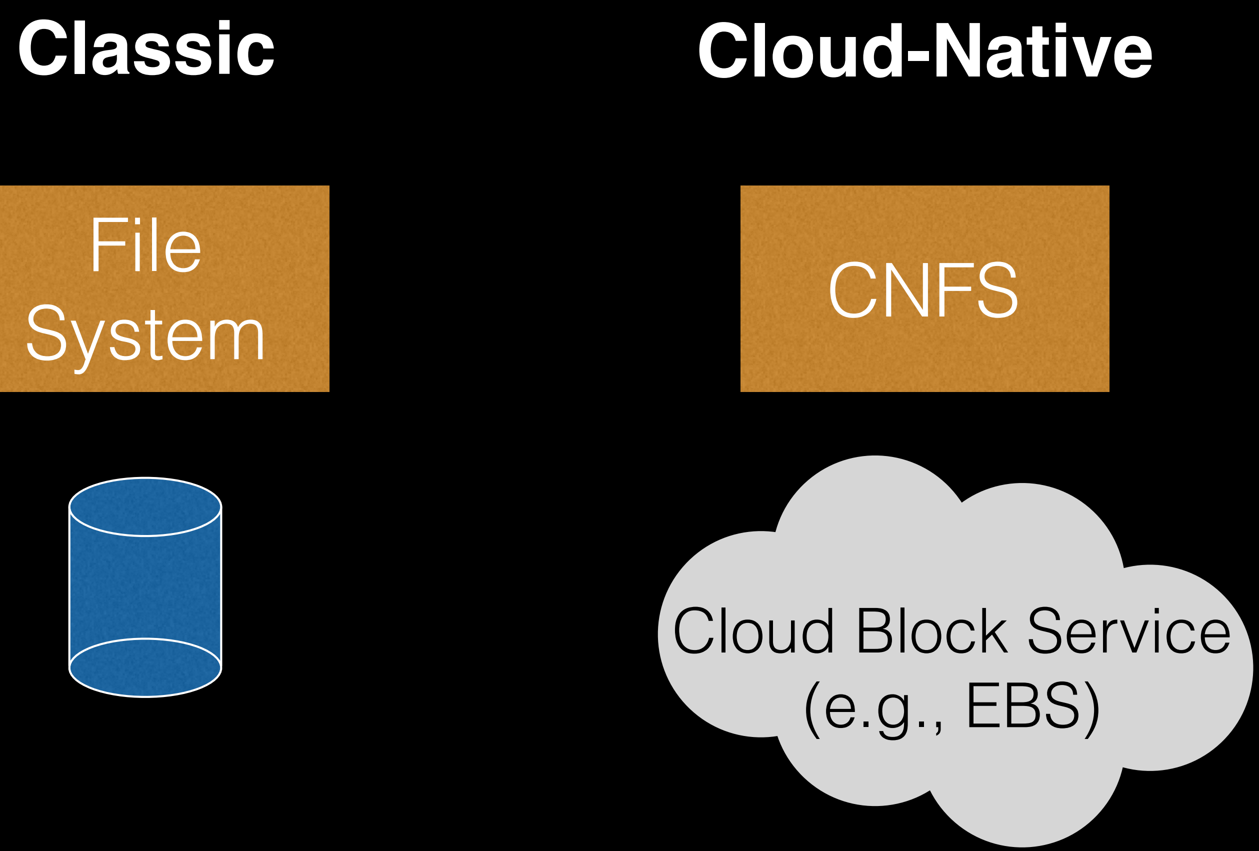
Cost: Have to know how much is OK to spend

Overall: Proper utilization of the cloud requires rethinking of how we build the systems above them

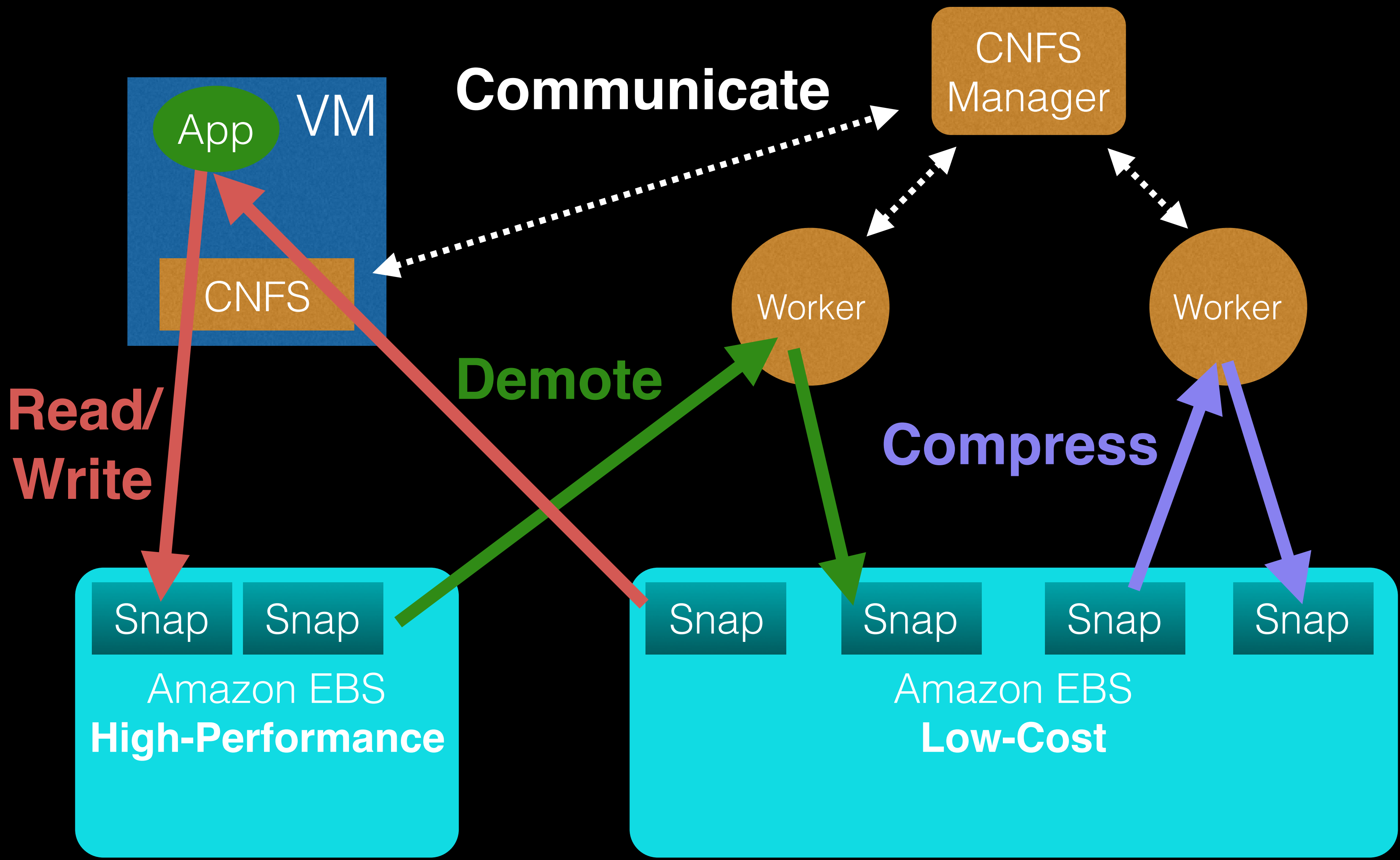
Case Study: CNFS

Case Study: CNFS

Case Study: **Cloud-Native File System (CNFS)**



CNFS Architecture



CNFS: Key Points

Copy-on-write (**COW**): Natural fit for cloud

- Enables background work on immutable storage

Storage work naturally offloaded from front end

- Enables predictable low-latency for foreground
- Adds massive parallelism for background

Can optimize for cost or performance or mix

- Need hints from above on what is important
- New APIs too

But, still needs help from cloud providers

- Example: Can't access EBS volumes from many clients (now)

Conclusions

Cloud Native

- New way to build systems upon substrate provided by Cloud

Principles: New guidelines for design

- **Higher-level services:** Don't reinvent the wheel
- **Flexible resources:** Can use a lot or a little
- **Different types of resources:** Costly/Fast vs. Cheap/Slow
- **Cost awareness:** Nothing is free

Case study: **CNFS**

- A local COW file system built to run on EBS (not a disk)
- Early prototype: Modified ext4 can migrate files across cloud volumes (but much still to be done)

Cloud-native thinking: How does it change your next system?

End