Optimized Storage for Seamless Multi-cloud Systems

UNIVERSITY OF TORONTO

Yunhao Mao, Yuqiu Zhang, Hans-Arno Jacobsen

MIDDLEWARE SYSTEMS RESEARCH GROUP
MSRG.ORG

Introduction

- **Sky Computing** is a novel **multi-cloud** paradigm
  - Unifies different clouds (public and private) via an inter-cloud abstraction layer
- State-of-the-art research includes multi-cloud machine learning and serverless computing, etc.
- Missing: cloud storage
- We propose **SkyBridge**: a multi-cloud *storage* management system
  - Acts as a ***middleware***, automates data placement and access across multiple cloud providers and cloud storage services

Goals

- **Uniform front-end**
  - Applications and users only interact with SkyBridge
- **Optimized cloud storage access and resource allocation**
  - Automatically selects **execution strategies** for requests based on multiple factors (e.g. ***cost*** and ***performance***)
- **Intelligent selection of storage types**
- **Extensibility**
  - Accounts for new cloud providers and storage services
- **Reliability**
  - Tolerates failures of some underlying storage services
- **Low performance overhead and high scalability**

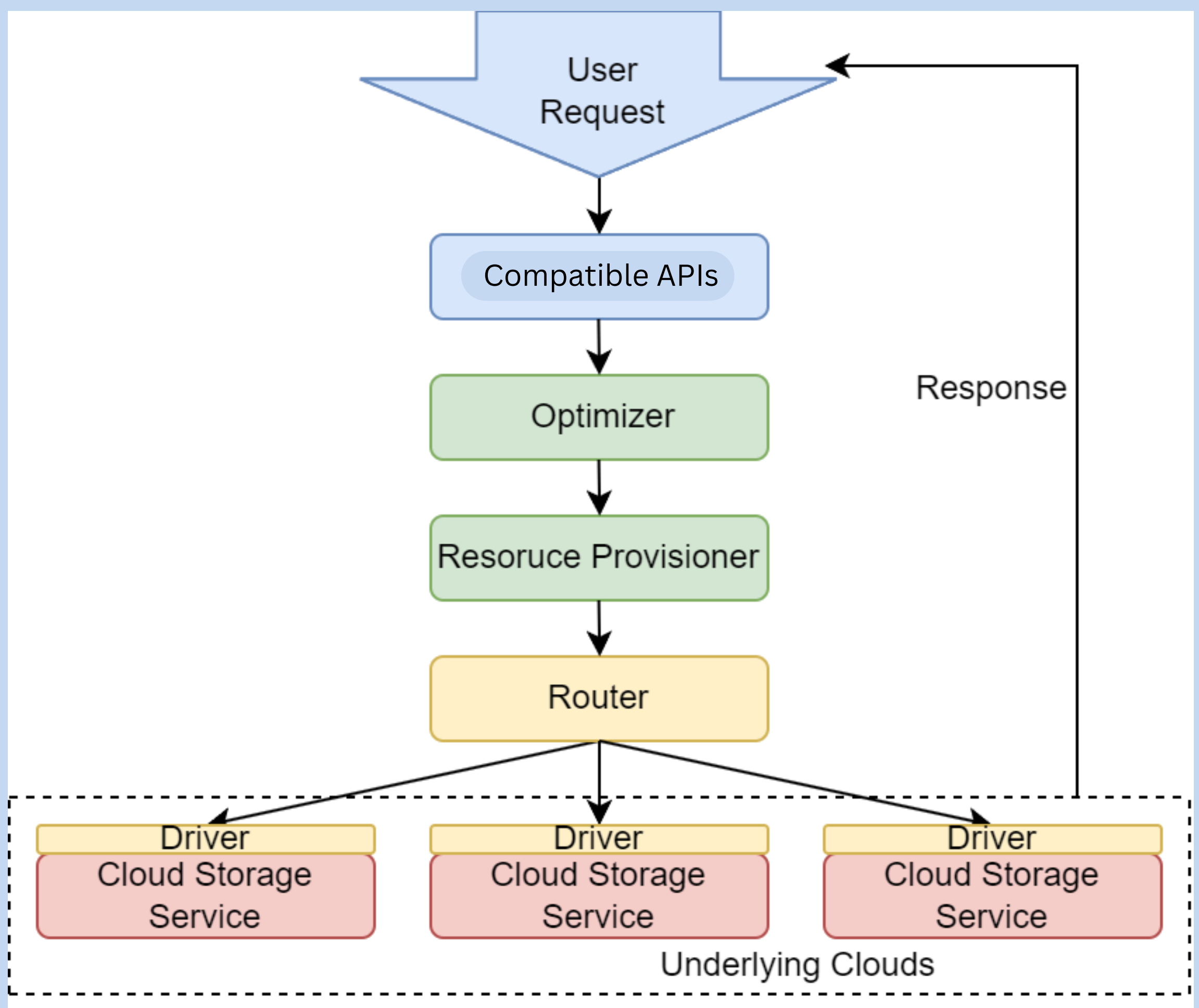
Summary

- **Multi-cloud storage management system**
- **Eliminates the need to manually handle multiple clouds**
- **Automatic optimization**
- **Improves performance, lowers cost**

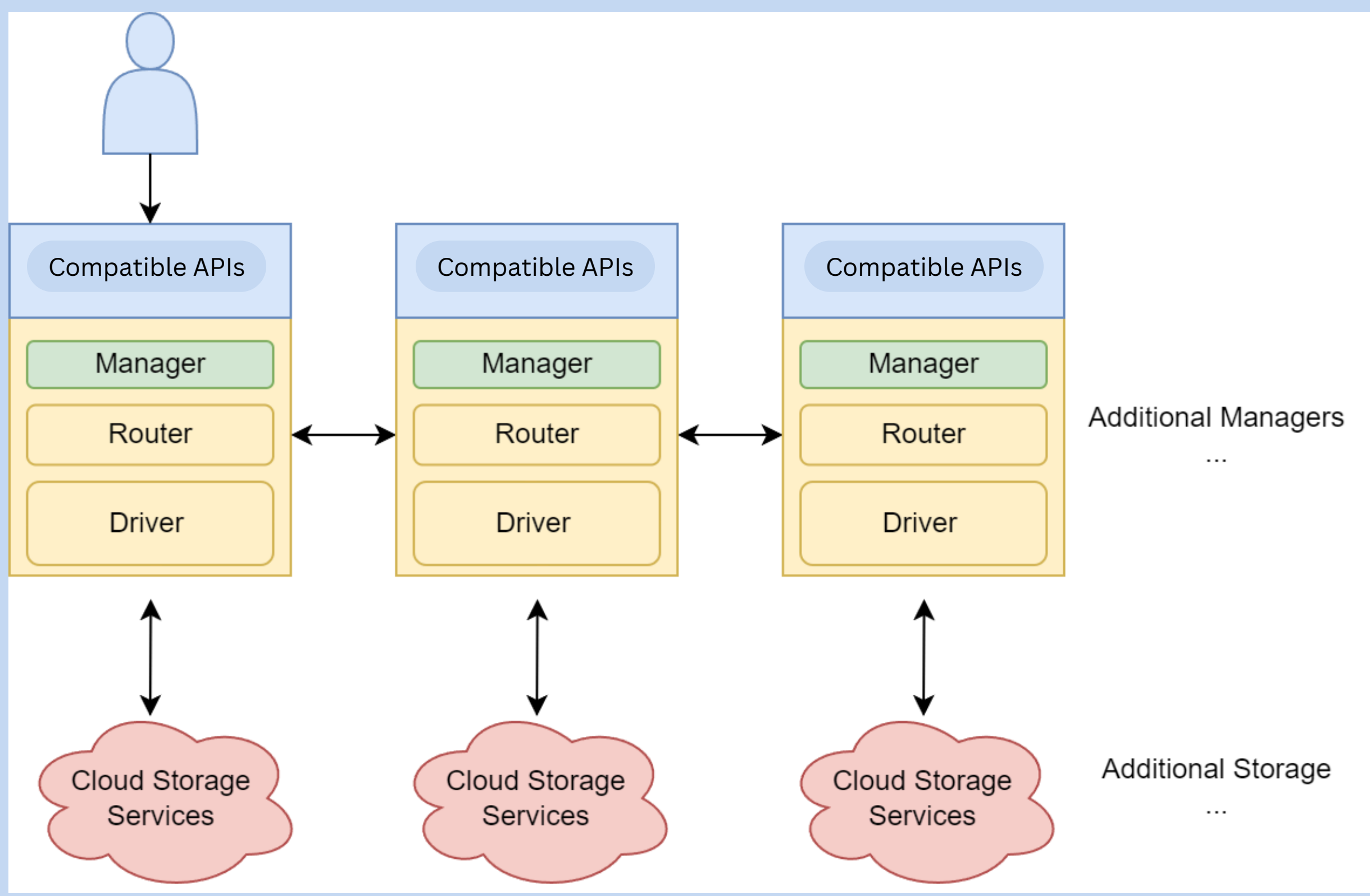
Research Questions

- **Characteristics of different cloud providers and their storage services**
  - How do they perform?
  - How much do they cost?
  - What unique features do they offer?
  - What is the best way to categorize them?
- **Use Cases**
  - What are relevant and useful use case?
  - How to collect **real life** multi-cloud storage usage patterns?
- **Quantifying, developing, and designing multi-cloud execution strategies**
  - What and how to optimize?

SkyBridge Design



Request Worklflow

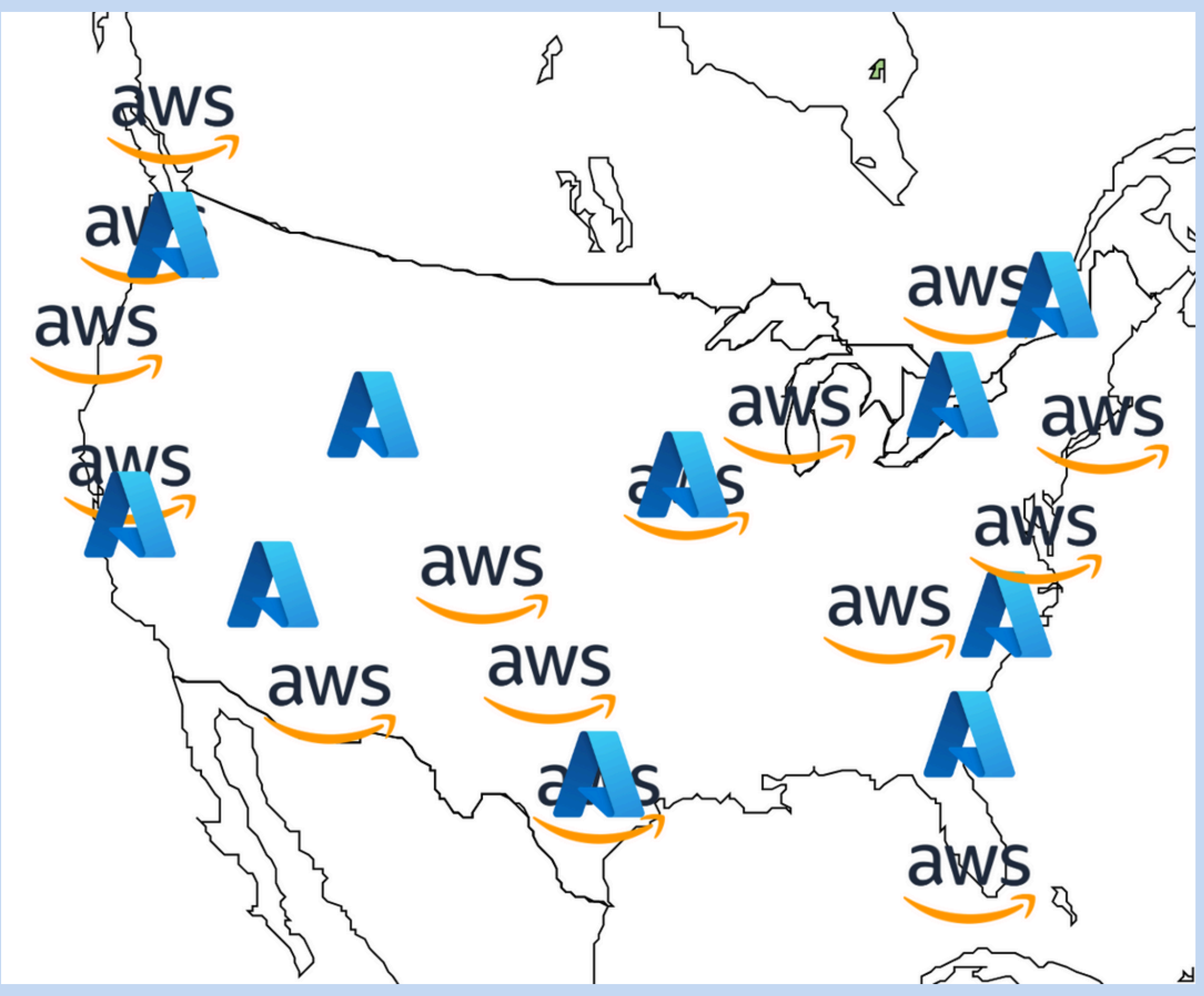


Architecture

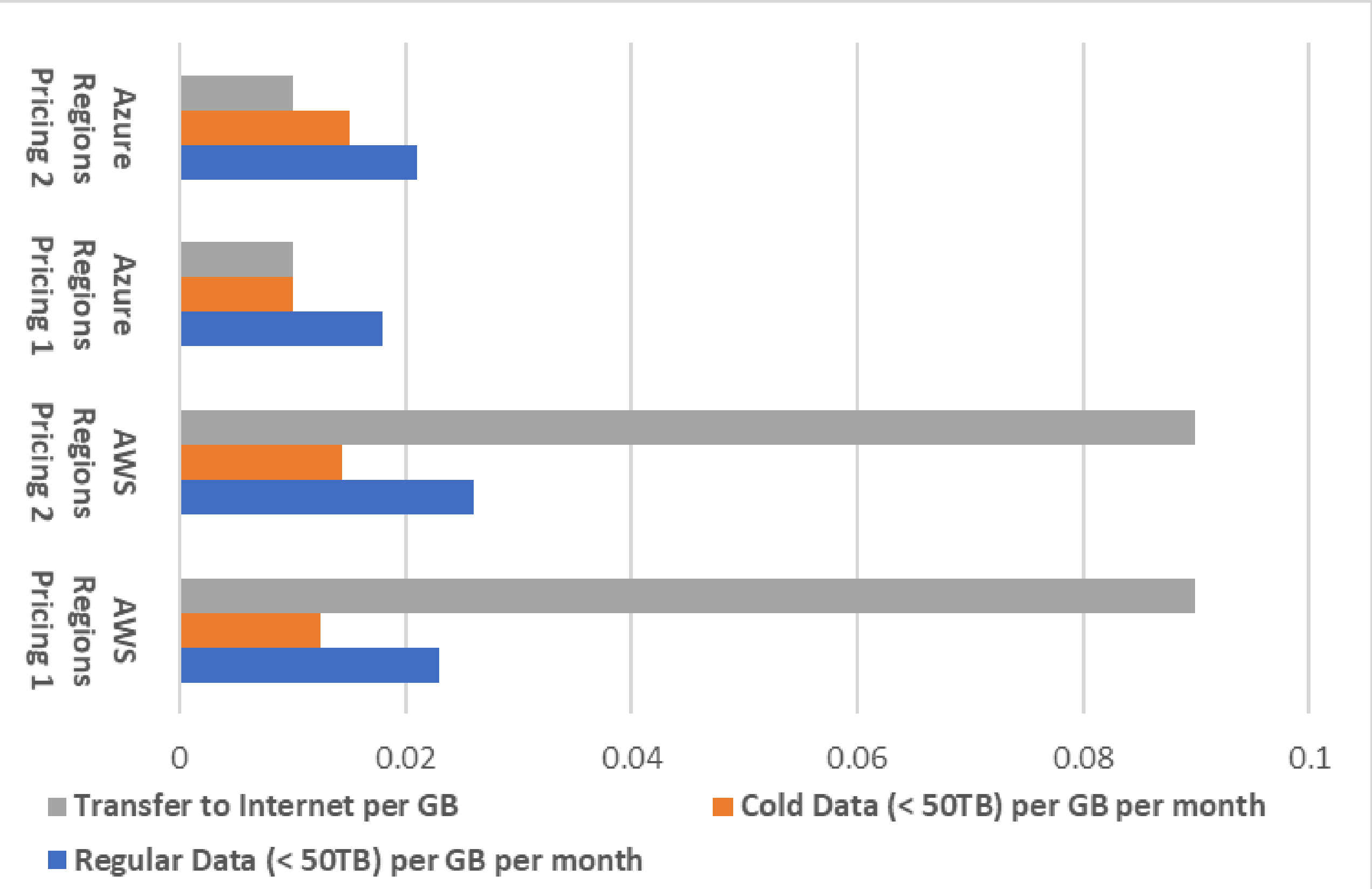
- **Compatible API**: compatible APIs such as object store, e.g. S3 API, key-value, SQL, etc.
- **Optimizer**: determines the best execution strategy to handle user requests
- **Provisioner**: determines the allocation of resources
- **Router**: a distributed manager that sends requests to target services
- **Driver**: translates requests from uniform APIs to cloud service-specific APIs

- ***Fully distributed***
- A cluster of **managers** (each contains optimizer and provisioner) that run in proximity to storage services
  - Example: a manager instance runs on an EC2 VM to manage co-located S3 and DynamoDB
- Requests are sent to ***any*** of the managers
- Managers independently select a strategy but coordinate with each other to route a request to the corresponding storage services

Execution Strategy Selection



US & Canada
AWS vs. Azure Datacenter Locations



AWS S3 vs. Azure Blob Pricing

- Execution strategies are decided based on ***user criteria, data access patterns, and cloud characteristics***
- **User criteria**
  - Replication or geo-distribution requirements
  - Hosting locations of the application
  - Cost vs. performance (latency, throughput)
- **Data access patterns**
  - Amount of data
  - Hot/cold data distribution
  - Random or with a regular access pattern
- **Cloud characteristic example**: AWS (S3) vs. Azure (Blob) Object Storage
  - Locations: data center locations are not evenly distributed, ***affecting latency***
  - Pricing: Azure is cheaper in general; data transfer out is costly, especially for AWS
- **Example strategy for an application hosted on AWS Vancouver:** uses AWS S3 to store hot data; selectively moves cold data to Azure Seattle and Azure Virginia as remote backup