Practical Course: Cloud-Based Data Processing

Michalis Georgoulakis

Chair for Database Systems
School of Computation, Information and Technology
Technical University of Munich

23.06.2026

Schedule

	Content	Lab Start	Lab End	Oral Discussions
23.06	Trends and Hardware			Lab 6
30.06	Q&A			
07.07	Q&A			
14.07	Project Presentations		Project	

Lab 6: Cloud Compute

Goal: Compute p50 / p95 of trip_time per pickup zone (NYC HVFHS) as a pipeline of stateless Azure Functions.

- Upload input/*.csv.gz
- Trigger -> 1. REPARTITION -> blob
- Trigger -> 2. AGGREGATE -> blob
- Trigger -> 3. MERGE -> Result

Functions

- repartition (blob-trigger, one per uploaded file): shard rows by zone
- aggregate (HTTP, one per shard): percentile per zone
- merge (HTTP, once): rank the zones

Driver Logic

- driver uploads, waits for stage 1, fans out aggregate, calls merge

Interesting Angles

- # partitions, # shards, function DOP

Lab 6 Considerations

- Blob triggers are async with no “join”
  - Polling before fanning out stage 2

- Percentiles are holistic.
  - To get a zone’s median you need all its values → **shuffle by zone**

- Shard by zone ⇒ each zone is contained in one shard ⇒ its percentile is final after aggregate; merge only ranks. (The core invariant.)

- Functions are stateless & ephemeral

Lab 6 Results

Batches ↑

- upload scales (sort of) linearly;
- repartition wall-clock partly hidden by overlap with upload.

Shards ↑

- aggregate faster (more parallelism, smaller shards)
- repartition cost rises (more, smaller blob writes).

maxDoP

- More ≠ better, due to CPU contention
- 2 was the sweet value for many people

Cold vs Warm

- First run pays worker spin-up; on Azure upload often dominates and masks it.

Lab 6 Findings

Storage is the shuffle substrate

- Lab 5 (substrates) → Lab 6 (serverless compute on it).

“Aggregate grows with shard count” is not necessarily true

- Driver’s serial HTTP can be parallelized → bounded by the slowest shard.

Serverless memory is a hard wall

- maxDoP raises per-instance concurrency and can OOM;
- real parallelism comes from scale-out (# instances)

No free “join”: orchestrating a serverless DAG means client-side polling for stage barriers.

Real e2e bottleneck: **upload + repartition (blob-trigger)**, not aggregate/merge.

Closing the Loop

- **What we built**
  - Most of the course builds around a single idea: storage–compute disaggregation
- **Where the hardware is going**
  - Object storage, faster networks, CXL, ARM, DPUs and accelerators
- **Your next three weeks**
  - You are now a small version of the people solving these problems

Disaggregation

Storage and compute are now separate machines. Almost every design choice follows from that.

1. The local disk is gone
2. Bandwidth scales with request parallelism, not with spindles or local IOPS
3. Compute is more elastic and stateless

Decision that come as a consequence:

- lightweight formats & compression → fewer bytes over the wire
- predicate / projection pushdown → move the filter to the data
- partitioning & shuffles → hide latency behind concurrency

Hardware Trend I: Object Storage Is the New Disk

- **A single GET is slow; thousands of concurrent GETs are fast**
- Throughput is bounded by your request concurrency, not by the store
- Consequence: the client — threading, batching, scheduling — is the database

A 25 Gbps NIC ≈ 3.1 GB/s.
At ~80 MB/s per request,
you need ~40 concurrent GETs
just to fill the pipe.

Hardware Trend II — The Network Caught Up

Remote storage now rivals local.

- **NICs went 10 → 25 → 100 → 200 → 400 Gbps in a few hardware generations**
- **NVMe-over-Fabrics: remote SSDs at near-local latency**
- **Low-latency object tiers**
  - S3 Express One Zone, Azure Premium Blob — single-digit-ms object access
- **Net effect: 'where is the data' matters less; 'how parallel is the access' matters more**

Hardware Trend III — Beyond CPU & DRAM

After storage, the next things to disaggregate are memory and the work itself.

Trend	What changes	Why it matters for you
CXL	Memory pooled & shared across nodes	The next disaggregation; spill without leaving DRAM speed
ARM in cloud	Graviton, Azure Cobalt, Ampere	Better perf/$ and perf/W; benchmark before assuming x86
DPUs / SmartNICs	Network, storage, off the CPU	More CPU left for actual query work
GPU / FPGA	Accelerated scans, joins, decompress	Throughput where the CPU is the limit
Serverless	Query as thousands of functions	Hypervisor

What's Unsolved

Open problems

- **Cost models across tiers: when is it cheaper to recompute than to store?**
- **Predictable tail latency on object storage**
- **CXL-aware query engines - engines still assume DRAM is local and private**
- **Pushing compute into storage**

Questions

Thank you for your attention!