# CS 431 Course Notes: Data Intensive Distributed Analytics

Fall 2025 - Dan Holtby

Talha Yildirim,
tyildir [ at ] uwaterloo [ dot ] ca

# Contents

# 1 Introduction

## 1.1 Big Ideas

**Proposition 1.1.1 (Scale 'out', not 'up')**

For data-intensive workloads, a large number of commodity low-end servers (i.e. scaling “out”) is preferred over a small number of high-end servers (i.e. scaling “up”)

Cost Considerations:

- **Scaling up (Symmetric Multiprocessing SMP machines):**
  - Not cost effective because costs do not scale linearly
  - A machine with twice the processors often costs much more than twice as much
- **Scaling Out (Commodity Servers):**
  - Benefits from overlap with high-volume desktop market
  - Prices are kept low by competition, interchangebale components, and economies of scale

Performance Trade Offs:

- **High-end SMP server:**
  - Faster intra-node communication (within machine) compared to cluster communication
- **Clusters:**
  - Network communication is unavoidable since no single machine can handle modern workloads alone
- **Comparison Results:**
  - Cluster of low-end servers perform close to cluster of high-end servers
  - Network communication acts as a speed bottleneck
  - Parallelization hides latency

Operational Costs:

- **Electricity Dominates Operational Costs:**
  - Energy efficiency is therefore a key issue in warehouse-scale computing
  - Non-linearity between load and power draw
    - Example: A server at 10% utilization may use more than half the power of a server at 100% utilization

**Proposition 1.1.2 (Assume Failures are Common)**

At warehouse scale, hardware failures are the norm, so distributed computing systems like MapReduce must be designed from the ground up to tolerate, recover from, and adapt to frequent failures automatically.

Requirements for Fault-Tolerant Services

- Failures should not cause inconsistencies or indeterminate results
- Other cluster nodes should seamlessly take over workloads from failed nodes
- Performance should degrade gradually, not collapse, as failures accumulate
- Repaired servers should be able to rejoin automatically without manual reconfiguration.

**Proposition 1.1.3 (Move Processing to the Data)**

Unlike high performance computers (HPC), which separates compute and storage, MapReduce co-locates them to exploit data locality, reducing costly data movement across the network.

**Proposition 1.1.4 (Process Data Sequentially and Avoid Random Access)**

Because random disk access is extremely slow compared to sequential access, MapReduce organizes workloads into sequential streaming operations, leveraging cluster-wide disk bandwidth and optimizing for throughput over latency.

**Proposition 1.1.5 (Hide System-Level Details from the Application Developer)**

Distributed programming is notoriously complex due to concurrency issues, but MapReduce simplifies it by abstracting away system-level details, letting developers concentrate on high-level computation logic.

**Proposition 1.1.6 (Seamless Scalability)**

While perfect scalability is impossible due to communication costs, MapReduce brings algorithms much closer to the ideal by abstracting away execution details, enabling near-linear scaling across large clusters.