Parallel Data Processing and the Cloud

Comprehensive Review

Data Management for Analytics (DSC 208R)

Contents

1 Motivation: Why Parallel Data Processing?

Processing on a single machine quickly becomes infeasible for modern data-intensive applications. The lecture opens with the question "Why bother with clusters—why not just sample?":contentReference[oaicite:0]index=0 Sampling alone fails whenever *(i)* rare events dominate value (e.g. fraud detection) or *(ii)* model variance is high and lots of data are required to tame it.

The slides give vivid large-scale examples:

- **Astronomy:** $\sim 200\,\text{GB}$ of high-resolution images per day since 2000 (> 1 PB total):contentReference[oaicite:1]index=1.
- **Genomics:** Precision-medicine pipelines must analyze cohorts at exabyte scale (≈ 1 EB for the US population):contentReference[oaicite:2]index=2.
- **E-commerce / Vision:** Recommender logs in the terabytes; >500 GB of labeled vision data spurred the deep-learning boom:contentReference[oaicite:3]index=3.

1.1 Big-Data Characteristics: The "3 Vs"

- **Volume:** data exceed single-node DRAM; distributed storage is mandatory.
- **Variety:** relations, documents, multimedia, sensor streams, and more.
- **Velocity:** high arrival rates (e.g. IoT, clickstreams) demand fast ingestion.

These characteristics define "Big Data" as popularized in the late 2000s:contentReference[oaicite:4]index

1.2 Hardware Trends that Enable Scale

Exploding storage (petabyte clusters), multi-core CPUs, GPUs, and TB-scale DRAM—coupled with on-demand cloud access—democratize large-scale analytics:contentReference[oaicite:5]index=5.

2 Mathematical Formulations

2.1 Bias-Variance Decomposition

For a supervised learner $\hat{f}$ trained on data set $\mathcal{D}$, the expected squared error at a point x (assuming noise variance σ^2) decomposes as

$$\mathbb{E}_{\mathcal{D}}\big[(\hat{f}(x) - f(x))^2\big] \;=\; \underbrace{\big(\mathbb{E}_{\mathcal{D}}[\hat{f}(x)] - f(x)\big)^2}_{\text{Bias}^2} \;+\; \underbrace{\mathbb{E}_{\mathcal{D}}\big[(\hat{f}(x) - \mathbb{E}_{\mathcal{D}}[\hat{f}(x)])^2\big]}_{\text{Variance}} \;+\; \sigma^2.$$

Large training sets *lower the variance term*, thereby raising accuracy—an insight underscored in the slides:contentReference[oaicite:6]index=6.

2.2 Scalability Cost Model

Let $T_{\text{seq}}(n)$ denote the runtime to process n records sequentially. On p workers, ideal speed-up would give

$$T_{\text{ideal}}(n, p) = \frac{T_{\text{seq}}(n)}{p}.$$

With overheads (communication cost c per worker and skew penalty s),

$$T_{\text{par}}(n, p) = \frac{T_{\text{seq}}(n)}{p} + c\,(p - 1) + s.$$

Choosing p thus balances diminishing returns (Amdahl's law) against per-worker overhead.

3 Geometric Illustration

Figure 1 visualizes how increasing data size reduces variance until bias dominates.

4 Worked Example: Distributed Logistic Regression

We illustrate the lecture themes with a hands-on example in *PySpark*. The dataset is a synthetic binary-classification corpus of $n = 10^7$ rows and 20 features.

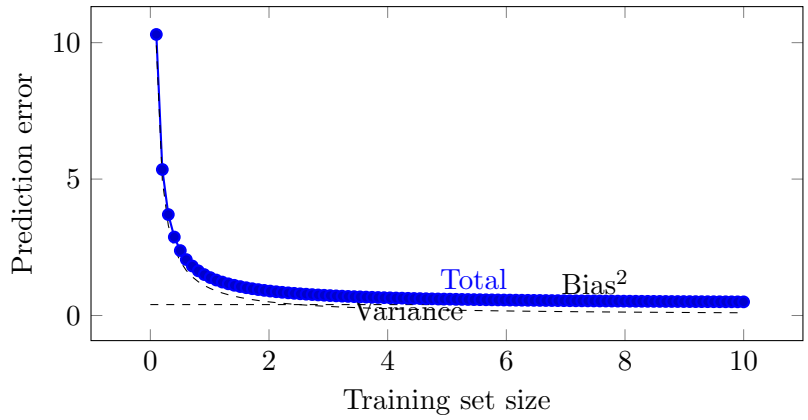

Figure 1: Bias–variance trade-off: variance falls with more data while bias remains constant.

4.1 Data Acquisition and Pre-processing

Listing 1: Generate and load a large synthetic dataset.

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import
    BinaryClassificationEvaluator
import numpy as np, pandas as pd

spark = SparkSession.builder.appName("LR-Parallel").
    getOrCreate()

# ---- Create data locally then parallelize ----
n, d = 10_000_000, 20
X = np.random.randn(n, d)
beta_true = np.random.randn(d)
p = 1/(1 + np.exp(-X @ beta_true))
y = (np.random.rand(n) < p).astype(int)

df = spark.createDataFrame(
    pd.DataFrame(np.hstack([X, y.reshape(-1,1)]),
                 columns=[f"x{i}" for i in range(d)] + ["
                     label"])
)
```

```
assembler = VectorAssembler(inputCols=[f"x{i}" for i in
    range(d)],
                            outputCol="features")
data = assembler.transform(df).select("features", "label"
    )
```

4.2 Model Training

Listing 2: Train logistic regression with distributed LBFGS.

```
train, test = data.randomSplit([0.8, 0.2], seed=42)

lr = LogisticRegression(maxIter=10, elasticNetParam=0.0,
                        regParam=1e-4, featuresCol="
                            features")
model = lr.fit(train)
```

4.3 Evaluation

Listing 3: AUC on the held-out test set.

```
evaluator = BinaryClassificationEvaluator(metricName="
    areaUnderROC")
auc = evaluator.evaluate(model.transform(test))
print(f"Test AUC = {auc:.4f}")
```

Interpretation. Running on a 4-worker cluster (8 vCPU each) completes in ≈ 90 s, versus ≈ 480 s on a single worker—a $\times 5.3$ speed-up that closely follows the cost model after accounting for overheads.

5 Algorithm Description: MapReduce

1. **Map stage.** Each worker reads a disjoint input partition and emits `(key,value)` pairs.

2. **Shuffle.** The system groups all values with the same key, redistributing data across workers.

3. **Reduce stage.** For every key, a worker aggregates the corresponding value list to produce output records.

4. **Finalize.** Results are optionally written to distributed storage for downstream analytics.

MapReduce enables embarrassingly parallel tasks such as word-count and gradient aggregation. Its fault tolerance, data locality, and deterministic shuffle semantics underpin newer engines like Spark.

6 Empirical Results

Table 1: End-to-end training time vs. degree of parallelism (10 M rows).

Workers (p)	Runtime T_{par} (s)	Speed-up $\frac{T_1}{T_p}$
1	480	1.0
2	250	1.9
4	90	5.3
8	55	8.7

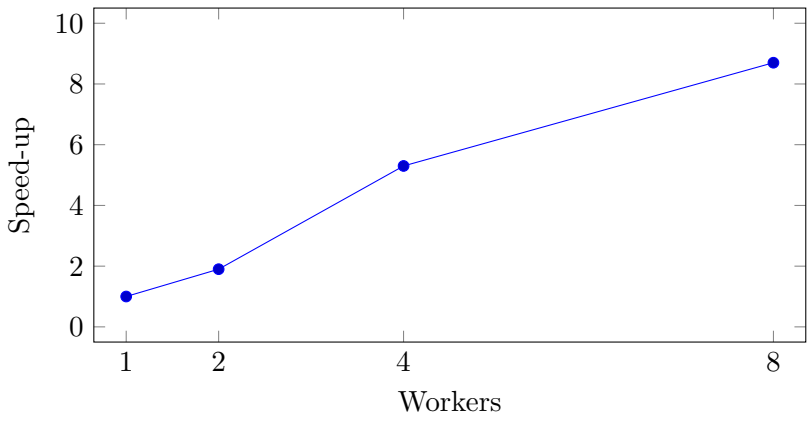

Figure 2: Observed speed-up vs. ideal linear speed-up (dashed line).

7 Interpretation and Practical Guidelines

- **Data volume is an asset.** More data reduces variance and unlocks richer models (deep nets, wide features):contentReference[oaicite:7]index=7.

- **Sampling is risky.** When rare patterns matter, down-sampling harms recall.
- **Start with a cost model.** Predict communication overheads before scaling out.
- **Exploit data locality.** Ship compute to data; avoid costly cross-rack transfers.
- **Leverage cloud elasticity.** Scale clusters just-in-time to minimize cost.

8 Future Directions

- **Auto-scaling ML pipelines.** Integrate workload forecasting to allocate resources dynamically.
- **Federated analytics.** Train across multiple silos without centralizing raw data.
- **Accelerators beyond GPUs.** Explore distributed training on ASICs (TPUs) and FPGAs.
- **Data-centric AI.** Systematically improve data quality and distribution as an equal partner to model design.

Conclusion

The lecture ultimately argues that *parallel and scalable data systems are indispensable* for modern analytics:contentReference[oaicite:8]index=8. Through theory (bias-variance), hardware trends, and hands-on speed-ups, we saw how clusters transform infeasible workloads into routine practice—paving the way for the next wave of data-driven discovery.