Datacenter as a Computer

Oversubscription: full usage of the downlink bandwidth is greater than the uplink bandwidth
Trends: CPU speed per core is flat; Memory bandwidth growing slower than capacity; SSD replacing HDDs; Ethernet bandwidth growing
Scale out: if data can be sharded then the workload can be divided into chunks; redundancy; high availability
Scale up: easy to use and program; when data cannot be sharded and distributed

Google File System

Workload: many large, sequential writes; bw>latency
Design: single master for metadata; chunk servers for data; no caches; no POSIX API (co-design with app)
Append: GFS choose the offset; atomic; at least once
Data FT: 3-way chain replication; cross racks; data integrity using checksum blocks
Master FT: shadow master; log checkpoint
CON: small random writes not well supported
NFS: file handle; STAT; stateless; idempotent
AFS: whole file caching; callbacks from server

MapReduce

Model: Map function: (K_{in}, V_{in}) -> list(K_{inter}, V_{inter}); reduce function: $(K_{inter}$, list$(V_{inter}))$ -> list(K_{out}, V_{out})
Assumption: Commodity networking, less bisection bandwidth; failures are common; cheap local storage; replicated FS
Worker FT: Completed map tasks are re-executed on a failure. Completed reduce tasks do not need to be re-executed. When a map task failed, all reducers are notified.
Master FT: there is only a single master, its failure is unlikely; current implementation aborts.
Refinements: combiner functions; counters; skip bad records; use disk for fault tolerance
Straggler: backup execution

MPI: Message Passing Interface; no failure recovery; hard to program;

Spark

Feature: Combine multiple MapReduce steps to 1 pass; reuse intermediate steps; easy to program; optimization across operators; shared variables (using broadcasting)
Fault Tolerance: re-execute lineage graph; manual checkpointing
Narrow dependency: output of an RDD used only for one other RDD. Grouped into stages for optimization.
Wide dependency: failure will lead to the re-execution of the whole RDD since more than one dependency
Action: collect, reduce, take, fold, count, saveAsFile

Mesos

Motivation: Run multiple frameworks on the same cluster; avoid per-framework cluster; data-sharing across framework
Design: guaranteed allocation; revocation;
FT: soft state; no checkpointing; all information can be retrieved from the workers;
Scheduler: send offer; lottery scheduling; decentralized
CON: fragmentation, large chunks of short tasks may starve the longer tasks, which those will never get a share of the resources to launch
YARN: Centralized scheduling. Pro: global optimum, avoid fragmentation, avoid starvation, better packing; Con: Not scalable, no support for future frameworks.

DRF

Dominant resource: resource with the biggest share
Dominant share: fraction of the dominant resource.
Asset Fairness: Equalize each user's sum of resource shares; violates sharing incentive
CEEI: maximizing product of utilities across users; trade resources with other users in a perfectly competitive market; not strategy-proof
Sharing incentive: better off by sharing the cluster
Strategy-proofness: not be able to benefit by lying
Envy-freeness: not prefer the allocation of another
Pareto efficiency: not possible to increase allocation without decreasing
Single resource fairness: For a single resource, the solution should reduce to max-min fairness.
Bottleneck fairness: If there is one resource that is percentwise demanded most of by every user, then the solution should reduce to max-min fairness for it.
Population monotonicity: When leaves, none of the allocations of the remaining users should decrease.
Resource monotonicity: If more resources are added to the system, none of the allocations should decrease.

Bismarck

Assumption: single machine; ML on RDBMS; model fits in memory; shared memory; convex optimization
Incremental gradient descent: only pick one point
Reservoir Sampling: I/O & mem worker, swap buffer.
Parallel: Lock vs AIG vs Lock-free

Parameter Server:

Assumption: sparse training data per worker
Architecture: server & worker group; server manager; task scheduler inside worker group
Representation: (ID, weight) pairs; range push & pull
Consistency: sequential, eventual, bounded delay
Vector clock: ensure well-defined behavior after network partition and failure
Implementation: key caching; value compression
Server replication: server stores a replica of the k counterclockwise neighbor key ranges
Worker failure: restart or ignore

TensorFlow:
Motivation: experiment with new layers avoiding the overhead of implementing layers using less familiar language; experiment with new optimization methods; adaptable if the resource architecture changes
Dataflow graph: static, symbolic; Edge=Tensor; Vertex=Operation
Partial execution: Multiple concurrent executions of overlapping sub-graphs (pre-processing, training, checkpointing, multiple layers)
Distributed execution: operation placed on devices; account for colocation; send-recv to stitch subgraphs
Extensions: auto differentiation; graph; user-level checkpointing; heterogeneous accelerator; async replication, sync replication w/ or w/o backup; backup worker for straggler proactively.
DistBelief: Written in C++; hard to experiment; execution pattern fixed; no support for new optimization methods; hard to write new layer

Ray: Reinforcement learning
Requirement: Simulation (tasks varying length), Training (dynamic execution), Serving (low latency)
Actor vs task: stateful classe vs. stateless function
Edge: control edge, data edge, stateful data edge;
Object store: communicate across nodes
Global control store: object table (list of object) + task table (lineage) + function table (running functions); **Scheduler**: local & global, talks to global control store **FT**: tasks (lineage from GCS), actors (checkpointing), GCS (sharded, replicated), scheduler (stateless); lineage

Clipper:
Challenge: the increase of different machine learning frameworks is optimized for development instead of deployment; Clipper introduces a model abstraction layer and common prediction interface that isolates applications
Interactive latency: using adaptive batching (Addictive increment & multiplicative decrement); Cache: LRU, improve performance for frequent query
Exp3: uses feedback to influence next selection
Model selection: improved prediction accuracy
Containers: isolation and auto-scaling up and down
Straggler: deadline, better approximate than late.

Gandiva
Workload: shared ML cluster, low hardware efficiency, feedback-driven exploration, early stopping, truncate execution
ML job: sensitive to locality, intra job predictability
Mechanism: suspend-resume jobs, migration for better locality, profiling resource usage, grow-shrink.
Reactive mode: scheduler reacts to events such as job arrivals, departures, machine failures
Introspective mode: a continuous process where the scheduler aims to improve cluster utilization and job completion time using packing, migration, and grow-shrink
Con: Need to modify PyTorch/TensorFlow. Does not have cluster-level fairness that ensures that each cluster is utilized fairly according to the workload

SparkSQL
RDD: relatively unordered, semi-structured
DataFrame: understand the structure of data.
Feature: Relational, not procedural. Operators, not expressions. Support debugging and interoperability, caching, UDF.
Catalyst: optimization queries
Logical optimization: constant folding, predicate pushdown, projection pruning, null propagation, Boolean expression simplification
Physical planning: join selection: hash/broadcast
Columnar storage: Compared with Spark's native cache, which simply stores data as JVM objects, the columnar cache can reduce memory footprint by an order of magnitude because it applies columnar compression schemes such as dictionary encoding and run-length encoding.

Geode:
Motivation: queries across dc; network awareness
Cannot be controlled: data birth, sovereignty. Support analytics queries; Minimize wide-area network usage
Assumption: plentiful resources in DC; fixed queries
Metric: Bandwidth cost not latency
Approach: join order selection, task assignment, manage data replication, sub query delta (cache intermediate results in sub-queries), make suggestion to administrators
Calcite++: estimate distributed join cost, input SQL parse tree, output optimized parse tree
Pseudo distributed execution: run on a single machine, has similar effect of multiple DC, precise estimation with some overhead.
Site selection & data replication: ILP problem, greedy approximation.