Glossary

Please note that the definitions in this glossary are short and simple, intended to convey the core idea but not the full subtleties of a term. For more detail, please follow the references into the main text.

asynchronous
: Not waiting for something to complete (e.g., sending data over the network to another node), and not making any assumptions about how long it is going to take. See "Synchronous Versus Asynchronous Replication" on page 153, "Synchronous Versus Asynchronous Networks" on page 284, and "System Model and Reality" on page 306.

atomic

1. In the context of concurrent operations: describing an operation that appears to take effect at a single point in time, so another concurrent process can never encounter the operation in a "half-finished" state. See also *isolation*.
2. In the context of transactions: grouping together a set of writes that must either all be committed or all be rolled back, even if faults occur. See "Atomicity" on page 223 and "Atomic Commit and Two-Phase Commit (2PC)" on page 354.

backpressure
: Forcing the sender of some data to slow down because the recipient cannot keep up with it. Also known as *flow control*. See "Messaging Systems" on page 441.

batch process
: A computation that takes some fixed (and usually large) set of data as input and produces some other data as output, without modifying the input. See Chapter 10.

bounded
: Having some known upper limit or size. Used for example in the context of network delay (see "Timeouts and Unbounded Delays" on page 281) and datasets (see the introduction to Chapter 11).

Byzantine fault
: A node that behaves incorrectly in some arbitrary way, for example by sending contradictory or malicious messages to other nodes. See "Byzantine Faults" on page 304.

cache
: A component that remembers recently used data in order to speed up future reads of the same data. It is generally not complete: thus, if some data is missing from the cache, it has to be fetched from some underlying, slower data storage

system that has a complete copy of the data.

CAP theorem
A widely misunderstood theoretical result that is not useful in practice. See "The CAP theorem" on page 336.

causality
The dependency between events that arises when one thing "happens before" another thing in a system. For example, a later event that is in response to an earlier event, or builds upon an earlier event, or should be understood in the light of an earlier event. See "The "happens-before" relationship and concurrency" on page 186 and "Ordering and Causality" on page 339.

consensus
A fundamental problem in distributed computing, concerning getting several nodes to agree on something (for example, which node should be the leader for a database cluster). The problem is much harder than it seems at first glance. See "Fault-Tolerant Consensus" on page 364.

data warehouse
A database in which data from several different OLTP systems has been combined and prepared to be used for analytics purposes. See "Data Warehousing" on page 91.

declarative
Describing the properties that something should have, but not the exact steps for how to achieve it. In the context of queries, a query optimizer takes a declarative query and decides how it should best be executed. See "Query Languages for Data" on page 42.

denormalize
To introduce some amount of redundancy or duplication in a *normalized* dataset, typically in the form of a *cache* or *index*, in order to speed up reads. A denormalized value is a kind of precomputed query result, similar to a materialized view. See "Single-Object and Multi-Object Operations" on page 228 and "Deriving several views from the same event log" on page 461.

derived data
A dataset that is created from some other data through a repeatable process, which you could run again if necessary. Usually, derived data is needed to speed up a particular kind of read access to the data. Indexes, caches, and materialized views are examples of derived data. See the introduction to Part III.

deterministic
Describing a function that always produces the same output if you give it the same input. This means it cannot depend on random numbers, the time of day, network communication, or other unpredictable things.

distributed
Running on several nodes connected by a network. Characterized by *partial failures*: some part of the system may be broken while other parts are still working, and it is often impossible for the software to know what exactly is broken. See "Faults and Partial Failures" on page 274.

durable
Storing data in a way such that you believe it will not be lost, even if various faults occur. See "Durability" on page 226.

ETL
Extract–Transform–Load. The process of extracting data from a source database, transforming it into a form that is more suitable for analytic queries, and loading it into a data warehouse or batch processing system. See "Data Warehousing" on page 91.

failover
In systems that have a single leader, failover is the process of moving the leadership role from one node to another. See "Handling Node Outages" on page 156.

fault-tolerant

Able to recover automatically if something goes wrong (e.g., if a machine crashes or a network link fails). See "Reliability" on page 6.

flow control

See *backpressure*.

follower

A replica that does not directly accept any writes from clients, but only processes data changes that it receives from a leader. Also known as a *secondary*, *slave*, *read replica*, or *hot standby*. See "Leaders and Followers" on page 152.

full-text search

Searching text by arbitrary keywords, often with additional features such as matching similarly spelled words or synonyms. A full-text index is a kind of *secondary index* that supports such queries. See "Full-text search and fuzzy indexes" on page 88.

graph

A data structure consisting of *vertices* (things that you can refer to, also known as *nodes* or *entities*) and *edges* (connections from one vertex to another, also known as *relationships* or *arcs*). See "Graph-Like Data Models" on page 49.

hash

A function that turns an input into a random-looking number. The same input always returns the same number as output. Two different inputs are very likely to have two different numbers as output, although it is possible that two different inputs produce the same output (this is called a *collision*). See "Partitioning by Hash of Key" on page 203.

idempotent

Describing an operation that can be safely retried; if it is executed more than once, it has the same effect as if it was only executed once. See "Idempotence" on page 478.

index

A data structure that lets you efficiently search for all records that have a particular value in a particular field. See "Data Structures That Power Your Database" on page 70.

isolation

In the context of transactions, describing the degree to which concurrently executing transactions can interfere with each other. *Serializable* isolation provides the strongest guarantees, but weaker isolation levels are also used. See "Isolation" on page 225.

join

To bring together records that have something in common. Most commonly used in the case where one record has a reference to another (a foreign key, a document reference, an edge in a graph) and a query needs to get the record that the reference points to. See "Many-to-One and Many-to-Many Relationships" on page 33 and "Reduce-Side Joins and Grouping" on page 403.

leader

When data or a service is replicated across several nodes, the leader is the designated replica that is allowed to make changes. A leader may be elected through some protocol, or manually chosen by an administrator. Also known as the *primary* or *master*. See "Leaders and Followers" on page 152.

linearizable

Behaving as if there was only a single copy of data in the system, which is updated by atomic operations. See "Linearizability" on page 324.

locality

A performance optimization: putting several pieces of data in the same place if they are frequently needed at the same time. See "Data locality for queries" on page 41.

lock
A mechanism to ensure that only one thread, node, or transaction can access something, and anyone else who wants to access the same thing must wait until the lock is released. See "Two-Phase Locking (2PL)" on page 257 and "The leader and the lock" on page 301.

log
An append-only file for storing data. A *write-ahead log* is used to make a storage engine resilient against crashes (see "Making B-trees reliable" on page 82), a *log-structured* storage engine uses logs as its primary storage format (see "SSTables and LSM-Trees" on page 76), a *replication log* is used to copy writes from a leader to followers (see "Leaders and Followers" on page 152), and an *event log* can represent a data stream (see "Partitioned Logs" on page 446).

materialize
To perform a computation eagerly and write out its result, as opposed to calculating it on demand when requested. See "Aggregation: Data Cubes and Materialized Views" on page 101 and "Materialization of Intermediate State" on page 419.

node
An instance of some software running on a computer, which communicates with other nodes via a network in order to accomplish some task.

normalized
Structured in such a way that there is no redundancy or duplication. In a normalized database, when some piece of data changes, you only need to change it in one place, not many copies in many different places. See "Many-to-One and Many-to-Many Relationships" on page 33.

OLAP
Online analytic processing. Access pattern characterized by aggregating (e.g., count, sum, average) over a large number of records. See "Transaction Processing or Analytics?" on page 90.

OLTP
Online transaction processing. Access pattern characterized by fast queries that read or write a small number of records, usually indexed by key. See "Transaction Processing or Analytics?" on page 90.

partitioning
Splitting up a large dataset or computation that is too big for a single machine into smaller parts and spreading them across several machines. Also known as *sharding*. See Chapter 6.

percentile
A way of measuring the distribution of values by counting how many values are above or below some threshold. For example, the 95th percentile response time during some period is the time *t* such that 95% of requests in that period complete in less than *t*, and 5% take longer than *t*. See "Describing Performance" on page 13.

primary key
A value (typically a number or a string) that uniquely identifies a record. In many applications, primary keys are generated by the system when a record is created (e.g., sequentially or randomly); they are not usually set by users. See also *secondary index*.

quorum
The minimum number of nodes that need to vote on an operation before it can be considered successful. See "Quorums for reading and writing" on page 179.

rebalance
To move data or services from one node to another in order to spread the load fairly. See "Rebalancing Partitions" on page 209.

replication
Keeping a copy of the same data on several nodes (*replicas*) so that it remains

accessible if a node becomes unreachable. See Chapter 5.

schema
A description of the structure of some data, including its fields and datatypes. Whether some data conforms to a schema can be checked at various points in the data's lifetime (see "Schema flexibility in the document model" on page 39), and a schema can change over time (see Chapter 4).

secondary index
An additional data structure that is maintained alongside the primary data storage and which allows you to efficiently search for records that match a certain kind of condition. See "Other Indexing Structures" on page 85 and "Partitioning and Secondary Indexes" on page 206.

serializable
A guarantee that if several transactions execute concurrently, they behave the same as if they had executed one at a time, in some serial order. See "Serializability" on page 251.

shared-nothing
An architecture in which independent nodes—each with their own CPUs, memory, and disks—are connected via a conventional network, in contrast to shared-memory or shared-disk architectures. See the introduction to Part II.

skew

1. Imbalanced load across partitions, such that some partitions have lots of requests or data, and others have much less. Also known as *hot spots*. See "Skewed Workloads and Relieving Hot Spots" on page 205 and "Handling skew" on page 407.
2. A timing anomaly that causes events to appear in an unexpected, nonsequential order. See the discussions of *read skew* in "Snapshot Isolation and Repeatable Read" on page 237, *write skew* in "Write Skew and Phantoms" on page 246, and *clock skew* in "Timestamps for ordering events" on page 291.

split brain
A scenario in which two nodes simultaneously believe themselves to be the leader, and which may cause system guarantees to be violated. See "Handling Node Outages" on page 156 and "The Truth Is Defined by the Majority" on page 300.

stored procedure
A way of encoding the logic of a transaction such that it can be entirely executed on a database server, without communicating back and forth with a client during the transaction. See "Actual Serial Execution" on page 252.

stream process
A continually running computation that consumes a never-ending stream of events as input, and derives some output from it. See Chapter 11.

synchronous
The opposite of *asynchronous*.

system of record
A system that holds the primary, authoritative version of some data, also known as the *source of truth*. Changes are first written here, and other datasets may be derived from the system of record. See the introduction to Part III.

timeout
One of the simplest ways of detecting a fault, namely by observing the lack of a response within some amount of time. However, it is impossible to know whether a timeout is due to a problem with the remote node, or an issue in the network. See "Timeouts and Unbounded Delays" on page 281.

total order
A way of comparing things (e.g., timestamps) that allows you to always say which one of two things is greater and which one is lesser. An ordering in which

some things are incomparable (you cannot say which is greater or smaller) is called a *partial order*. See "The causal order is not a total order" on page 341.

transaction
Grouping together several reads and writes into a logical unit, in order to simplify error handling and concurrency issues. See Chapter 7.

two-phase commit (2PC)
An algorithm to ensure that several database nodes either all commit or all abort a transaction. See "Atomic Commit and Two-Phase Commit (2PC)" on page 354.

two-phase locking (2PL)
An algorithm for achieving serializable isolation that works by a transaction acquiring a lock on all data it reads or writes, and holding the lock until the end of the transaction. See "Two-Phase Locking (2PL)" on page 257.

unbounded
Not having any known upper limit or size. The opposite of *bounded*.

Index

A

aborts (transactions), 222, 224
  in two-phase commit, 356
  performance of optimistic concurrency control, 266
  retrying aborted transactions, 231
abstraction, 21, 27, 222, 266, 321
access path (in network model), 37, 60
accidental complexity, removing, 21
accountability, 535
ACID properties (transactions), 90, 223
  atomicity, 223, 228
  consistency, 224, 529
  durability, 226
  isolation, 225, 228
acknowledgements (messaging), 445
active/active replication (see multi-leader replication)
active/passive replication (see leader-based replication)
ActiveMQ (messaging), 137, 444
  distributed transaction support, 361
ActiveRecord (object-relational mapper), 30, 232
actor model, 138
  (see also message-passing)
  comparison to Pregel model, 425
  comparison to stream processing, 468
Advanced Message Queuing Protocol (see AMQP)
aerospace systems, 6, 10, 305, 372
aggregation
  data cubes and materialized views, 101
  in batch processes, 406
  in stream processes, 466
aggregation pipeline query language, 48
Agile, 22
  minimizing irreversibility, 414, 497
  moving faster with confidence, 532
  Unix philosophy, 394
agreement, 365
  (see also consensus)
Airflow (workflow scheduler), 402
Ajax, 131
Akka (actor framework), 139
algorithms
  algorithm correctness, 308
  B-trees, 79-83
  for distributed systems, 306
  hash indexes, 72-75
  mergesort, 76, 402, 405
  red-black trees, 78
  SSTables and LSM-trees, 76-79
all-to-all replication topologies, 175
AllegroGraph (database), 50
ALTER TABLE statement (SQL), 40, 111
Amazon
  Dynamo (database), 177
Amazon Web Services (AWS), 8
  Kinesis Streams (messaging), 448
  network reliability, 279
  postmortems, 9
  RedShift (database), 93
  S3 (object storage), 398
    checking data integrity, 530
amplification
  of bias, 534
  of failures, 364, 495

of tail latency, 16, 207
write amplification, 84
AMQP (Advanced Message Queuing Protocol), 444
(see also messaging systems)
comparison to log-based messaging, 448, 451
message ordering, 446
analytics, 90
comparison to transaction processing, 91
data warehousing (see data warehousing)
parallel query execution in MPP databases, 415
predictive (see predictive analytics)
relation to batch processing, 411
schemas for, 93-95
snapshot isolation for queries, 238
stream analytics, 466
using MapReduce, analysis of user activity events (example), 404
anti-caching (in-memory databases), 89
anti-entropy, 178
Apache ActiveMQ (see ActiveMQ)
Apache Avro (see Avro)
Apache Beam (see Beam)
Apache BookKeeper (see BookKeeper)
Apache Cassandra (see Cassandra)
Apache CouchDB (see CouchDB)
Apache Curator (see Curator)
Apache Drill (see Drill)
Apache Flink (see Flink)
Apache Giraph (see Giraph)
Apache Hadoop (see Hadoop)
Apache HAWQ (see HAWQ)
Apache HBase (see HBase)
Apache Helix (see Helix)
Apache Hive (see Hive)
Apache Impala (see Impala)
Apache Jena (see Jena)
Apache Kafka (see Kafka)
Apache Lucene (see Lucene)
Apache MADlib (see MADlib)
Apache Mahout (see Mahout)
Apache Oozie (see Oozie)
Apache Parquet (see Parquet)
Apache Qpid (see Qpid)
Apache Samza (see Samza)
Apache Solr (see Solr)
Apache Spark (see Spark)
Apache Storm (see Storm)
Apache Tajo (see Tajo)
Apache Tez (see Tez)
Apache Thrift (see Thrift)
Apache ZooKeeper (see ZooKeeper)
Apama (stream analytics), 466
append-only B-trees, 82, 242
append-only files (see logs)
Application Programming Interfaces (APIs), 5, 27
for batch processing, 403
for change streams, 456
for distributed transactions, 361
for graph processing, 425
for services, 131-136
(see also services)
evolvability, 136
RESTful, 133
SOAP, 133
application state (see state)
approximate search (see similarity search)
archival storage, data from databases, 131
arcs (see edges)
arithmetic mean, 14
ASCII text, 119, 395
ASN.1 (schema language), 127
asynchronous networks, 278, 553
comparison to synchronous networks, 284
formal model, 307
asynchronous replication, 154, 553
conflict detection, 172
data loss on failover, 157
reads from asynchronous follower, 162
Asynchronous Transfer Mode (ATM), 285
atomic broadcast (see total order broadcast)
atomic clocks (caesium clocks), 294, 295
(see also clocks)
atomicity (concurrency), 553
atomic increment-and-get, 351
compare-and-set, 245, 327
(see also compare-and-set operations)
replicated operations, 246
write operations, 243
atomicity (transactions), 223, 228, 553
atomic commit, 353
avoiding, 523, 528
blocking and nonblocking, 359
in stream processing, 360, 477
maintaining derived data, 453

for multi-object transactions, 229
for single-object writes, 230
auditability, 528-533
designing for, 531
self-auditing systems, 530
through immutability, 460
tools for auditable data systems, 532
availability, 8
(see also fault tolerance)
in CAP theorem, 337
in service level agreements (SLAs), 15
Avro (data format), 122-127
code generation, 127
dynamically generated schemas, 126
object container files, 125, 131, 414
reader determining writer's schema, 125
schema evolution, 123
use in Hadoop, 414
awk (Unix tool), 391
AWS (see Amazon Web Services)
Azure (see Microsoft)

B

B-trees (indexes), 79-83
append-only/copy-on-write variants, 82, 242
branching factor, 81
comparison to LSM-trees, 83-85
crash recovery, 82
growing by splitting a page, 81
optimizations, 82
similarity to dynamic partitioning, 212
backpressure, 441, 553
in TCP, 282
backups
database snapshot for replication, 156
integrity of, 530
snapshot isolation for, 238
use for ETL processes, 405
backward compatibility, 112
BASE, contrast to ACID, 223
bash shell (Unix), 70, 395, 503
batch processing, 28, 389-431, 553
combining with stream processing
lambda architecture, 497
unifying technologies, 498
comparison to MPP databases, 414-418
comparison to stream processing, 464
comparison to Unix, 413-414
dataflow engines, 421-423
fault tolerance, 406, 414, 422, 442
for data integration, 494-498
graphs and iterative processing, 424-426
high-level APIs and languages, 403, 426-429
log-based messaging and, 451
maintaining derived state, 495
MapReduce and distributed filesystems, 397-413
(see also MapReduce)
measuring performance, 13, 390
outputs, 411-413
key-value stores, 412
search indexes, 411
using Unix tools (example), 391-394
Bayou (database), 522
Beam (dataflow library), 498
bias, 534
big ball of mud, 20
Bigtable data model, 41, 99
binary data encodings, 115-128
Avro, 122-127
MessagePack, 116-117
Thrift and Protocol Buffers, 117-121
binary encoding
based on schemas, 127
by network drivers, 128
binary strings, lack of support in JSON and XML, 114
BinaryProtocol encoding (Thrift), 118
Bitcask (storage engine), 72
crash recovery, 74
Bitcoin (cryptocurrency), 532
Byzantine fault tolerance, 305
concurrency bugs in exchanges, 233
bitmap indexes, 97
blockchains, 532
Byzantine fault tolerance, 305
blocking atomic commit, 359
Bloom (programming language), 504
Bloom filter (algorithm), 79, 466
BookKeeper (replicated log), 372
Bottled Water (change data capture), 455
bounded datasets, 430, 439, 553
(see also batch processing)
bounded delays, 553
in networks, 285
process pauses, 298
broadcast hash joins, 409

brokerless messaging, 442
Brubeck (metrics aggregator), 442
BTM (transaction coordinator), 356
bulk synchronous parallel (BSP) model, 425
bursty network traffic patterns, 285
business data processing, 28, 90, 390
byte sequence, encoding data in, 112
Byzantine faults, 304-306, 307, 553
    Byzantine fault-tolerant systems, 305, 532
    Byzantine Generals Problem, 304
    consensus algorithms and, 366

C

caches, 89, 553
    and materialized views, 101
    as derived data, 386, 499-504
    database as cache of transaction log, 460
    in CPUs, 99, 338, 428
    invalidation and maintenance, 452, 467
    linearizability, 324
CAP theorem, 336-338, 554
Cascading (batch processing), 419, 427
    hash joins, 409
    workflows, 403
cascading failures, 9, 214, 281
Cascalog (batch processing), 60
Cassandra (database)
    column-family data model, 41, 99
    compaction strategy, 79
    compound primary key, 204
    gossip protocol, 216
    hash partitioning, 203-205
    last-write-wins conflict resolution, 186, 292
    leaderless replication, 177
    linearizability, lack of, 335
    log-structured storage, 78
    multi-datacenter support, 184
    partitioning scheme, 213
    secondary indexes, 207
    sloppy quorums, 184
cat (Unix tool), 391
causal context, 191
    (see also causal dependencies)
causal dependencies, 186-191
    capturing, 191, 342, 494, 514
        by total ordering, 493
    causal ordering, 339
    in transactions, 262
    sending message to friends (example), 494
causality, 554
    causal ordering, 339-343
        linearizability and, 342
        total order consistent with, 344, 345
    consistency with, 344-347
    consistent snapshots, 340
    happens-before relationship, 186
    in serializable transactions, 262-265
    mismatch with clocks, 292
    ordering events to capture, 493
    violations of, 165, 176, 292, 340
    with synchronized clocks, 294
CEP (see complex event processing)
certificate transparency, 532
chain replication, 155
    linearizable reads, 351
change data capture, 160, 454
    API support for change streams, 456
    comparison to event sourcing, 457
    implementing, 454
    initial snapshot, 455
    log compaction, 456
changelogs, 460
    change data capture, 454
    for operator state, 479
    generating with triggers, 455
    in stream joins, 474
    log compaction, 456
    maintaining derived state, 452
Chaos Monkey, 7, 280
checkpointing
    in batch processors, 422, 426
    in high-performance computing, 275
    in stream processors, 477, 523
chronicle data model, 458
circuit-switched networks, 284
circular buffers, 450
circular replication topologies, 175
clickstream data, analysis of, 404
clients
    calling services, 131
    pushing state changes to, 512
    request routing, 214
    stateful and offline-capable, 170, 511
clocks, 287-299
    atomic (caesium) clocks, 294, 295
    confidence interval, 293-295
    for global snapshots, 294
    logical (see logical clocks)

skew, 291-294, 334
slewing, 289
synchronization and accuracy, 289-291
synchronization using GPS, 287, 290, 294, 295
time-of-day versus monotonic clocks, 288
timestamping events, 471
cloud computing, 146, 275
need for service discovery, 372
network glitches, 279
shared resources, 284
single-machine reliability, 8
Cloudera Impala (see Impala)
clustered indexes, 86
CODASYL model, 36
(see also network model)
code generation
with Avro, 127
with Thrift and Protocol Buffers, 118
with WSDL, 133
collaborative editing
multi-leader replication and, 170
column families (Bigtable), 41, 99
column-oriented storage, 95-101
column compression, 97
distinction between column families and, 99
in batch processors, 428
Parquet, 96, 131, 414
sort order in, 99-100
vectorized processing, 99, 428
writing to, 101
comma-separated values (see CSV)
command query responsibility segregation (CQRS), 462
commands (event sourcing), 459
commits (transactions), 222
atomic commit, 354-355
(see also atomicity; transactions)
read committed isolation, 234
three-phase commit (3PC), 359
two-phase commit (2PC), 355-359
commutative operations, 246
compaction
of changelogs, 456
(see also log compaction)
for stream operator state, 479
of log-structured storage, 73
issues with, 84
size-tiered and leveled approaches, 79
CompactProtocol encoding (Thrift), 119
compare-and-set operations, 245, 327
implementing locks, 370
implementing uniqueness constraints, 331
implementing with total order broadcast, 350
relation to consensus, 335, 350, 352, 374
relation to transactions, 230
compatibility, 112, 128
calling services, 136
properties of encoding formats, 139
using databases, 129-131
using message-passing, 138
compensating transactions, 355, 461, 526
complex event processing (CEP), 465
complexity
distilling in theoretical models, 310
hiding using abstraction, 27
of software systems, managing, 20
composing data systems (see unbundling databases)
compute-intensive applications, 3, 275
concatenated indexes, 87
in Cassandra, 204
Concord (stream processor), 466
concurrency
actor programming model, 138, 468
(see also message-passing)
bugs from weak transaction isolation, 233
conflict resolution, 171, 174
detecting concurrent writes, 184-191
dual writes, problems with, 453
happens-before relationship, 186
in replicated systems, 161-191, 324-338
lost updates, 243
multi-version concurrency control (MVCC), 239
optimistic concurrency control, 261
ordering of operations, 326, 341
reducing, through event logs, 351, 462, 507
time and relativity, 187
transaction isolation, 225
write skew (transaction isolation), 246-251
conflict-free replicated datatypes (CRDTs), 174
conflicts
conflict detection, 172
causal dependencies, 186, 342
in consensus algorithms, 368
in leaderless replication, 184

in log-based systems, 351, 521
in nonlinearizable systems, 343
in serializable snapshot isolation (SSI), 264
in two-phase commit, 357, 364
conflict resolution
automatic conflict resolution, 174
by aborting transactions, 261
by apologizing, 527
convergence, 172-174
in leaderless systems, 190
last write wins (LWW), 186, 292
using atomic operations, 246
using custom logic, 173
determining what is a conflict, 174, 522
in multi-leader replication, 171-175
avoiding conflicts, 172
lost updates, 242-246
materializing, 251
relation to operation ordering, 339
write skew (transaction isolation), 246-251
congestion (networks)
avoidance, 282
limiting accuracy of clocks, 293
queueing delays, 282
consensus, 321, 364-375, 554
algorithms, 366-368
preventing split brain, 367
safety and liveness properties, 365
using linearizable operations, 351
cost of, 369
distributed transactions, 352-375
in practice, 360-364
two-phase commit, 354-359
XA transactions, 361-364
impossibility of, 353
membership and coordination services, 370-373
relation to compare-and-set, 335, 350, 352, 374
relation to replication, 155, 349
relation to uniqueness constraints, 521
consistency, 224, 524
across different databases, 157, 452, 462, 492
causal, 339-348, 493
consistent prefix reads, 165-167
consistent snapshots, 156, 237-242, 294, 455, 500
(see also snapshots)
crash recovery, 82
enforcing constraints (see constraints)
eventual, 162, 322
(see also eventual consistency)
in ACID transactions, 224, 529
in CAP theorem, 337
linearizability, 324-338
meanings of, 224
monotonic reads, 164-165
of secondary indexes, 231, 241, 354, 491, 500
ordering guarantees, 339-352
read-after-write, 162-164
sequential, 351
strong (see linearizability)
timeliness and integrity, 524
using quorums, 181, 334
consistent hashing, 204
consistent prefix reads, 165
constraints (databases), 225, 248
asynchronously checked, 526
coordination avoidance, 527
ensuring idempotence, 519
in log-based systems, 521-524
across multiple partitions, 522
in two-phase commit, 355, 357
relation to consensus, 374, 521
relation to event ordering, 347
requiring linearizability, 330
Consul (service discovery), 372
consumers (message streams), 137, 440
backpressure, 441
consumer offsets in logs, 449
failures, 445, 449
fan-out, 11, 445, 448
load balancing, 444, 448
not keeping up with producers, 441, 450, 502
context switches, 14, 297
convergence (conflict resolution), 172-174, 322
coordination
avoidance, 527
cross-datacenter, 168, 493
cross-partition ordering, 256, 294, 348, 523
services, 330, 370-373
coordinator (in 2PC), 356
failure, 358
in XA transactions, 361-364
recovery, 363

copy-on-write (B-trees), 82, 242
CORBA (Common Object Request Broker Architecture), 134
correctness, 6
  auditability, 528-533
  Byzantine fault tolerance, 305, 532
  dealing with partial failures, 274
  in log-based systems, 521-524
  of algorithm within system model, 308
  of compensating transactions, 355
  of consensus, 368
  of derived data, 497, 531
  of immutable data, 461
  of personal data, 535, 540
  of time, 176, 289-295
  of transactions, 225, 515, 529
  timeliness and integrity, 524-528
corruption of data
  detecting, 519, 530-533
  due to pathological memory access, 529
  due to radiation, 305
  due to split brain, 158, 302
  due to weak transaction isolation, 233
  formalization in consensus, 366
  integrity as absence of, 524
  network packets, 306
  on disks, 227
  preventing using write-ahead logs, 82
  recovering from, 414, 460
Couchbase (database)
  durability, 89
  hash partitioning, 203-204, 211
  rebalancing, 213
  request routing, 216
CouchDB (database)
  B-tree storage, 242
  change feed, 456
  document data model, 31
  join support, 34
  MapReduce support, 46, 400
  replication, 170, 173
covering indexes, 86
CPUs
  cache coherence and memory barriers, 338
  caching and pipelining, 99, 428
  increasing parallelism, 43
CRDTs (see conflict-free replicated datatypes)
CREATE INDEX statement (SQL), 85, 500
credit rating agencies, 535
Crunch (batch processing), 419, 427
  hash joins, 409
  sharded joins, 408
  workflows, 403
cryptography
  defense against attackers, 306
  end-to-end encryption and authentication, 519, 543
  proving integrity of data, 532
CSS (Cascading Style Sheets), 44
CSV (comma-separated values), 70, 114, 396
Curator (ZooKeeper recipes), 330, 371
curl (Unix tool), 135, 397
cursor stability, 243
Cypher (query language), 52
  comparison to SPARQL, 59

D

data corruption (see corruption of data)
data cubes, 102
data formats (see encoding)
data integration, 490-498, 543
  batch and stream processing, 494-498
    lambda architecture, 497
    maintaining derived state, 495
    reprocessing data, 496
    unifying, 498
  by unbundling databases, 499-515
    comparison to federated databases, 501
  combining tools by deriving data, 490-494
    derived data versus distributed transactions, 492
    limits of total ordering, 493
    ordering events to capture causality, 493
    reasoning about dataflows, 491
  need for, 385
data lakes, 415
data locality (see locality)
data models, 27-64
  graph-like models, 49-63
    Datalog language, 60-63
    property graphs, 50
    RDF and triple-stores, 55-59
  query languages, 42-48
  relational model versus document model, 28-42
data protection regulations, 542
data systems, 3
  about, 4

concerns when designing, 5
future of, 489-544
correctness, constraints, and integrity, 515-533
data integration, 490-498
unbundling databases, 499-515
heterogeneous, keeping in sync, 452
maintainability, 18-22
possible faults in, 221
reliability, 6-10
hardware faults, 7
human errors, 9
importance of, 10
software errors, 8
scalability, 10-18
unreliable clocks, 287-299
data warehousing, 91-95, 554
comparison to data lakes, 415
ETL (extract-transform-load), 92, 416, 452
keeping data systems in sync, 452
schema design, 93
slowly changing dimension (SCD), 476
data-intensive applications, 3
database triggers (see triggers)
database-internal distributed transactions, 360, 364, 477
databases
archival storage, 131
comparison of message brokers to, 443
dataflow through, 129
end-to-end argument for, 519-520
checking integrity, 531
inside-out, 504
(see also unbundling databases)
output from batch workflows, 412
relation to event streams, 451-464
(see also changelogs)
API support for change streams, 456, 506
change data capture, 454-457
event sourcing, 457-459
keeping systems in sync, 452-453
philosophy of immutable events, 459-464
unbundling, 499-515
composing data storage technologies, 499-504
designing applications around dataflow, 504-509
observing derived state, 509-515
datacenters
geographically distributed, 145, 164, 278, 493
multi-tenancy and shared resources, 284
network architecture, 276
network faults, 279
replication across multiple, 169
leaderless replication, 184
multi-leader replication, 168, 335
dataflow, 128-139, 504-509
correctness of dataflow systems, 525
differential, 504
message-passing, 136-139
reasoning about, 491
through databases, 129
through services, 131-136
dataflow engines, 421-423
comparison to stream processing, 464
directed acyclic graphs (DAG), 424
partitioning, approach to, 429
support for declarative queries, 427
Datalog (query language), 60-63
datatypes
binary strings in XML and JSON, 114
conflict-free, 174
in Avro encodings, 122
in Thrift and Protocol Buffers, 121
numbers in XML and JSON, 114
Datomic (database)
B-tree storage, 242
data model, 50, 57
Datalog query language, 60
excision (deleting data), 463
languages for transactions, 255
serial execution of transactions, 253
deadlocks
detection, in two-phase commit (2PC), 364
in two-phase locking (2PL), 258
Debezium (change data capture), 455
declarative languages, 42, 554
Bloom, 504
CSS and XSL, 44
Cypher, 52
Datalog, 60
for batch processing, 427
recursive SQL queries, 53
relational algebra and SQL, 42
SPARQL, 59

delays
  bounded network delays, 285
  bounded process pauses, 298
  unbounded network delays, 282
  unbounded process pauses, 296
deleting data, 463
denormalization (data representation), 34, 554
  costs, 39
  in derived data systems, 386
  materialized views, 101
  updating derived data, 228, 231, 490
  versus normalization, 462
derived data, 386, 439, 554
  from change data capture, 454
  in event sourcing, 458-458
  maintaining derived state through logs, 452-457, 459-463
  observing, by subscribing to streams, 512
  outputs of batch and stream processing, 495
  through application code, 505
  versus distributed transactions, 492
deterministic operations, 255, 274, 554
  accidental nondeterminism, 423
  and fault tolerance, 423, 426
  and idempotence, 478, 492
  computing derived data, 495, 526, 531
  in state machine replication, 349, 452, 458
  joins, 476
DevOps, 394
differential dataflow, 504
dimension tables, 94
dimensional modeling (see star schemas)
directed acyclic graphs (DAGs), 424
dirty reads (transaction isolation), 234
dirty writes (transaction isolation), 235
discrimination, 534
disks (see hard disks)
distributed actor frameworks, 138
distributed filesystems, 398-399
  decoupling from query engines, 417
  indiscriminately dumping data into, 415
  use by MapReduce, 402
distributed systems, 273-312, 554
  Byzantine faults, 304-306
  cloud versus supercomputing, 275
  detecting network faults, 280
  faults and partial failures, 274-277
  formalization of consensus, 365
  impossibility results, 338, 353
  issues with failover, 157
  limitations of distributed transactions, 363
  multi-datacenter, 169, 335
  network problems, 277-286
  quorums, relying on, 301
  reasons for using, 145, 151
  synchronized clocks, relying on, 291-295
  system models, 306-310
  use of clocks and time, 287
distributed transactions (see transactions)
Django (web framework), 232
DNS (Domain Name System), 216, 372
Docker (container manager), 506
document data model, 30-42
  comparison to relational model, 38-42
  document references, 38, 403
  document-oriented databases, 31
  many-to-many relationships and joins, 36
  multi-object transactions, need for, 231
  versus relational model
    convergence of models, 41
    data locality, 41
document-partitioned indexes, 206, 217, 411
domain-driven design (DDD), 457
DRBD (Distributed Replicated Block Device), 153
drift (clocks), 289
Drill (query engine), 93
Druid (database), 461
Dryad (dataflow engine), 421
dual writes, problems with, 452, 507
duplicates, suppression of, 517
  (see also idempotence)
  using a unique ID, 518, 522
durability (transactions), 226, 554
duration (time), 287
  measurement with monotonic clocks, 288
dynamic partitioning, 212
dynamically typed languages
  analogy to schema-on-read, 40
  code generation and, 127
Dynamo-style databases (see leaderless replication)

E

edges (in graphs), 49, 403
  property graph model, 50
edit distance (full-text search), 88
effectively-once semantics, 476, 516

(see also exactly-once semantics)
preservation of integrity, 525
elastic systems, 17
Elasticsearch (search server)
document-partitioned indexes, 207
partition rebalancing, 211
percolator (stream search), 467
usage example, 4
use of Lucene, 79
ElephantDB (database), 413
Elm (programming language), 504, 512
encodings (data formats), 111-128
Avro, 122-127
binary variants of JSON and XML, 115
compatibility, 112
calling services, 136
using databases, 129-131
using message-passing, 138
defined, 113
JSON, XML, and CSV, 114
language-specific formats, 113
merits of schemas, 127
representations of data, 112
Thrift and Protocol Buffers, 117-121
end-to-end argument, 277, 519-520
checking integrity, 531
publish/subscribe streams, 512
enrichment (stream), 473
Enterprise JavaBeans (EJB), 134
entities (see vertices)
epoch (consensus algorithms), 368
epoch (Unix timestamps), 288
equi-joins, 403
erasure coding (error correction), 398
Erlang OTP (actor framework), 139
error handling
for network faults, 280
in transactions, 231
error-correcting codes, 277, 398
Esper (CEP engine), 466
etcd (coordination service), 370-373
linearizable operations, 333
locks and leader election, 330
quorum reads, 351
service discovery, 372
use of Raft algorithm, 349, 353
Ethereum (blockchain), 532
Ethernet (networks), 276, 278, 285
packet checksums, 306, 519
Etherpad (collaborative editor), 170
ethics, 533-543
code of ethics and professional practice, 533
legislation and self-regulation, 542
predictive analytics, 533-536
amplifying bias, 534
feedback loops, 536
privacy and tracking, 536-543
consent and freedom of choice, 538
data as assets and power, 540
meaning of privacy, 539
surveillance, 537
respect, dignity, and agency, 543, 544
unintended consequences, 533, 536
ETL (extract-transform-load), 92, 405, 452, 554
use of Hadoop for, 416
event sourcing, 457-459
commands and events, 459
comparison to change data capture, 457
comparison to lambda architecture, 497
deriving current state from event log, 458
immutability and auditability, 459, 531
large, reliable data systems, 519, 526
Event Store (database), 458
event streams (see streams)
events, 440
deciding on total order of, 493
deriving views from event log, 461
difference to commands, 459
event time versus processing time, 469, 477, 498
immutable, advantages of, 460, 531
ordering to capture causality, 493
reads as, 513
stragglers, 470, 498
timestamp of, in stream processing, 471
EventSource (browser API), 512
eventual consistency, 152, 162, 308, 322
(see also conflicts)
and perpetual inconsistency, 525
evolvability, 21, 111
calling services, 136
graph-structured data, 52
of databases, 40, 129-131, 461, 497
of message-passing, 138
reprocessing data, 496, 498
schema evolution in Avro, 123
schema evolution in Thrift and Protocol Buffers, 120

schema-on-read, 39, 111, 128
exactly-once semantics, 360, 476, 516
parity with batch processors, 498
preservation of integrity, 525
exclusive mode (locks), 258
eXtended Architecture transactions (see XA transactions)
extract-transform-load (see ETL)

F

Facebook
Presto (query engine), 93
React, Flux, and Redux (user interface libraries), 512
social graphs, 49
Wormhole (change data capture), 455
fact tables, 93
failover, 157, 554
(see also leader-based replication)
in leaderless replication, absence of, 178
leader election, 301, 348, 352
potential problems, 157
failures
amplification by distributed transactions, 364, 495
failure detection, 280
automatic rebalancing causing cascading failures, 214
perfect failure detectors, 359
timeouts and unbounded delays, 282, 284
using ZooKeeper, 371
faults versus, 7
partial failures in distributed systems, 275-277, 310
fan-out (messaging systems), 11, 445
fault tolerance, 6-10, 555
abstractions for, 321
formalization in consensus, 365-369
use of replication, 367
human fault tolerance, 414
in batch processing, 406, 414, 422, 425
in log-based systems, 520, 524-526
in stream processing, 476-479
atomic commit, 477
idempotence, 478
maintaining derived state, 495
microbatching and checkpointing, 477
rebuilding state after a failure, 478
of distributed transactions, 362-364
transaction atomicity, 223, 354-361
faults, 6
Byzantine faults, 304-306
failures versus, 7
handled by transactions, 221
handling in supercomputers and cloud computing, 275
hardware, 7
in batch processing versus distributed databases, 417
in distributed systems, 274-277
introducing deliberately, 7, 280
network faults, 279-281
asymmetric faults, 300
detecting, 280
tolerance of, in multi-leader replication, 169
software errors, 8
tolerating (see fault tolerance)
federated databases, 501
fence (CPU instruction), 338
fencing (preventing split brain), 158, 302-304
generating fencing tokens, 349, 370
properties of fencing tokens, 308
stream processors writing to databases, 478, 517
Fibre Channel (networks), 398
field tags (Thrift and Protocol Buffers), 119-121
file descriptors (Unix), 395
financial data, 460
Firebase (database), 456
Flink (processing framework), 421-423
dataflow APIs, 427
fault tolerance, 422, 477, 479
Gelly API (graph processing), 425
integration of batch and stream processing, 495, 498
machine learning, 428
query optimizer, 427
stream processing, 466
flow control, 282, 441, 555
FLP result (on consensus), 353
FlumeJava (dataflow library), 403, 427
followers, 152, 555
(see also leader-based replication)
foreign keys, 38, 403
forward compatibility, 112
forward decay (algorithm), 16

Fossil (version control system), 463
  shunning (deleting data), 463
FoundationDB (database)
  serializable transactions, 261, 265, 364
fractal trees, 83
full table scans, 403
full-text search, 555
  and fuzzy indexes, 88
  building search indexes, 411
  Lucene storage engine, 79
functional reactive programming (FRP), 504
functional requirements, 22
futures (asynchronous operations), 135
fuzzy search (see similarity search)

G
garbage collection
  immutability and, 463
  process pauses for, 14, 296-299, 301
    (see also process pauses)
genome analysis, 63, 429
geographically distributed datacenters, 145, 164, 278, 493
geospatial indexes, 87
Giraph (graph processing), 425
Git (version control system), 174, 342, 463
GitHub, postmortems, 157, 158, 309
global indexes (see term-partitioned indexes)
GlusterFS (distributed filesystem), 398
GNU Coreutils (Linux), 394
GoldenGate (change data capture), 161, 170, 455
  (see also Oracle)
Google
  Bigtable (database)
    data model (see Bigtable data model)
    partitioning scheme, 199, 202
    storage layout, 78
  Chubby (lock service), 370
  Cloud Dataflow (stream processor), 466, 477, 498
    (see also Beam)
  Cloud Pub/Sub (messaging), 444, 448
  Docs (collaborative editor), 170
  Dremel (query engine), 93, 96
  FlumeJava (dataflow library), 403, 427
  GFS (distributed file system), 398
  gRPC (RPC framework), 135
  MapReduce (batch processing), 390
    (see also MapReduce)
    building search indexes, 411
    task preemption, 418
  Pregel (graph processing), 425
  Spanner (see Spanner)
  TrueTime (clock API), 294
gossip protocol, 216
government use of data, 541
GPS (Global Positioning System)
  use for clock synchronization, 287, 290, 294, 295
GraphChi (graph processing), 426
graphs, 555
  as data models, 49-63
    example of graph-structured data, 49
    property graphs, 50
    RDF and triple-stores, 55-59
    versus the network model, 60
  processing and analysis, 424-426
    fault tolerance, 425
    Pregel processing model, 425
  query languages
    Cypher, 52
    Datalog, 60-63
    recursive SQL queries, 53
    SPARQL, 59-59
Gremlin (graph query language), 50
grep (Unix tool), 392
GROUP BY clause (SQL), 406
grouping records in MapReduce, 406
  handling skew, 407

H
Hadoop (data infrastructure)
  comparison to distributed databases, 390
  comparison to MPP databases, 414-418
  comparison to Unix, 413-414, 499
  diverse processing models in ecosystem, 417
  HDFS distributed filesystem (see HDFS)
  higher-level tools, 403
  join algorithms, 403-410
    (see also MapReduce)
  MapReduce (see MapReduce)
  YARN (see YARN)
happens-before relationship, 340
  capturing, 187
  concurrency and, 186
hard disks
  access patterns, 84

detecting corruption, 519, 530
faults in, 7, 227
sequential write throughput, 75, 450
hardware faults, 7
hash indexes, 72-75
broadcast hash joins, 409
partitioned hash joins, 409
hash partitioning, 203-205, 217
consistent hashing, 204
problems with hash mod N, 210
range queries, 204
suitable hash functions, 203
with fixed number of partitions, 210
HAWQ (database), 428
HBase (database)
bug due to lack of fencing, 302
bulk loading, 413
column-family data model, 41, 99
dynamic partitioning, 212
key-range partitioning, 202
log-structured storage, 78
request routing, 216
size-tiered compaction, 79
use of HDFS, 417
use of ZooKeeper, 370
HDFS (Hadoop Distributed File System), 398-399
(see also distributed filesystems)
checking data integrity, 530
decoupling from query engines, 417
indiscriminately dumping data into, 415
metadata about datasets, 410
NameNode, 398
use by Flink, 479
use by HBase, 212
use by MapReduce, 402
HdrHistogram (numerical library), 16
head (Unix tool), 392
head vertex (property graphs), 51
head-of-line blocking, 15
heap files (databases), 86
Helix (cluster manager), 216
heterogeneous distributed transactions, 360, 364
heuristic decisions (in 2PC), 363
Hibernate (object-relational mapper), 30
hierarchical model, 36
high availability (see fault tolerance)
high-frequency trading, 290, 299
high-performance computing (HPC), 275
hinted handoff, 183
histograms, 16
Hive (query engine), 419, 427
for data warehouses, 93
HCatalog and metastore, 410
map-side joins, 409
query optimizer, 427
skewed joins, 408
workflows, 403
Hollerith machines, 390
hopping windows (stream processing), 472
(see also windows)
horizontal scaling (see scaling out)
HornetQ (messaging), 137, 444
distributed transaction support, 361
hot spots, 201
due to celebrities, 205
for time-series data, 203
in batch processing, 407
relieving, 205
hot standbys (see leader-based replication)
HTTP, use in APIs (see services)
human errors, 9, 279, 414
HyperDex (database), 88
HyperLogLog (algorithm), 466

I

I/O operations, waiting for, 297
IBM
DB2 (database)
distributed transaction support, 361
recursive query support, 54
serializable isolation, 242, 257
XML and JSON support, 30, 42
electromechanical card-sorting machines, 390
IMS (database), 36
imperative query APIs, 46
InfoSphere Streams (CEP engine), 466
MQ (messaging), 444
distributed transaction support, 361
System R (database), 222
WebSphere (messaging), 137
idempotence, 134, 478, 555
by giving operations unique IDs, 518, 522
idempotent operations, 517
immutability
advantages of, 460, 531

deriving state from event log, 459-464
for crash recovery, 75
in B-trees, 82, 242
in event sourcing, 457
inputs to Unix commands, 397
limitations of, 463
Impala (query engine)
for data warehouses, 93
hash joins, 409
native code generation, 428
use of HDFS, 417
impedance mismatch, 29
imperative languages, 42
setting element styles (example), 45
in doubt (transaction status), 358
holding locks, 362
orphaned transactions, 363
in-memory databases, 88
durability, 227
serial transaction execution, 253
incidents
cascading failures, 9
crashes due to leap seconds, 290
data corruption and financial losses due to concurrency bugs, 233
data corruption on hard disks, 227
data loss due to last-write-wins, 173, 292
data on disks unreadable, 309
deleted items reappearing, 174
disclosure of sensitive data due to primary key reuse, 157
errors in transaction serializability, 529
gigabit network interface with 1 Kb/s throughput, 311
network faults, 279
network interface dropping only inbound packets, 279
network partitions and whole-datacenter failures, 275
poor handling of network faults, 280
sending message to ex-partner, 494
sharks biting undersea cables, 279
split brain due to 1-minute packet delay, 158, 279
vibrations in server rack, 14
violation of uniqueness constraint, 529
indexes, 71, 555
and snapshot isolation, 241
as derived data, 386, 499-504
B-trees, 79-83
building in batch processes, 411
clustered, 86
comparison of B-trees and LSM-trees, 83-85
concatenated, 87
covering (with included columns), 86
creating, 500
full-text search, 88
geospatial, 87
hash, 72-75
index-range locking, 260
multi-column, 87
partitioning and secondary indexes, 206-209, 217
secondary, 85
(see also secondary indexes)
problems with dual writes, 452, 491
SSTables and LSM-trees, 76-79
updating when data changes, 452, 467
Industrial Revolution, 541
InfiniBand (networks), 285
InfiniteGraph (database), 50
InnoDB (storage engine)
clustered index on primary key, 86
not preventing lost updates, 245
preventing write skew, 248, 257
serializable isolation, 257
snapshot isolation support, 239
inside-out databases, 504
(see also unbundling databases)
integrating different data systems (see data integration)
integrity, 524
coordination-avoiding data systems, 528
correctness of dataflow systems, 525
in consensus formalization, 365
integrity checks, 530
(see also auditing)
end-to-end, 519, 531
use of snapshot isolation, 238
maintaining despite software bugs, 529
Interface Definition Language (IDL), 117, 122
intermediate state, materialization of, 420-423
internet services, systems for implementing, 275
invariants, 225
(see also constraints)
inversion of control, 396
IP (Internet Protocol)

unreliability of, 277
ISDN (Integrated Services Digital Network), 284
isolation (in transactions), 225, 228, 555
correctness and, 515
for single-object writes, 230
serializability, 251-266
actual serial execution, 252-256
serializable snapshot isolation (SSI), 261-266
two-phase locking (2PL), 257-261
violating, 228
weak isolation levels, 233-251
preventing lost updates, 242-246
read committed, 234-237
snapshot isolation, 237-242
iterative processing, 424-426

J

Java Database Connectivity (JDBC)
distributed transaction support, 361
network drivers, 128
Java Enterprise Edition (EE), 134, 356, 361
Java Message Service (JMS), 444
(see also messaging systems)
comparison to log-based messaging, 448, 451
distributed transaction support, 361
message ordering, 446
Java Transaction API (JTA), 355, 361
Java Virtual Machine (JVM)
bytecode generation, 428
garbage collection pauses, 296
process reuse in batch processors, 422
JavaScript
in MapReduce querying, 46
setting element styles (example), 45
use in advanced queries, 48
Jena (RDF framework), 57
Jepsen (fault tolerance testing), 515
jitter (network delay), 284
joins, 555
by index lookup, 403
expressing as relational operators, 427
in relational and document databases, 34
MapReduce map-side joins, 408-410
broadcast hash joins, 409
merge joins, 410
partitioned hash joins, 409
MapReduce reduce-side joins, 403-408
handling skew, 407
sort-merge joins, 405
parallel execution of, 415
secondary indexes and, 85
stream joins, 472-476
stream-stream join, 473
stream-table join, 473
table-table join, 474
time-dependence of, 475
support in document databases, 42
JOTM (transaction coordinator), 356
JSON
Avro schema representation, 122
binary variants, 115
for application data, issues with, 114
in relational databases, 30, 42
representing a résumé (example), 31
Juttle (query language), 504

K

k-nearest neighbors, 429
Kafka (messaging), 137, 448
Kafka Connect (database integration), 457, 461
Kafka Streams (stream processor), 466, 467
fault tolerance, 479
leader-based replication, 153
log compaction, 456, 467
message offsets, 447, 478
request routing, 216
transaction support, 477
usage example, 4
Ketama (partitioning library), 213
key-value stores, 70
as batch process output, 412
hash indexes, 72-75
in-memory, 89
partitioning, 201-205
by hash of key, 203, 217
by key range, 202, 217
dynamic partitioning, 212
skew and hot spots, 205
Kryo (Java), 113
Kubernetes (cluster manager), 418, 506

L

lambda architecture, 497
Lamport timestamps, 345

Large Hadron Collider (LHC), 64
last write wins (LWW), 173, 334
  discarding concurrent writes, 186
  problems with, 292
  prone to lost updates, 246
late binding, 396
latency
  instability under two-phase locking, 259
  network latency and resource utilization, 286
  response time versus, 14
  tail latency, 15, 207
leader-based replication, 152-161
  (see also replication)
  failover, 157, 301
  handling node outages, 156
  implementation of replication logs
    change data capture, 454-457
      (see also changelogs)
    statement-based, 158
    trigger-based replication, 161
    write-ahead log (WAL) shipping, 159
  linearizability of operations, 333
  locking and leader election, 330
  log sequence number, 156, 449
  read-scaling architecture, 161
  relation to consensus, 367
  setting up new followers, 155
  synchronous versus asynchronous, 153-155
leaderless replication, 177-191
  (see also replication)
  detecting concurrent writes, 184-191
    capturing happens-before relationship, 187
    happens-before relationship and concurrency, 186
    last write wins, 186
    merging concurrently written values, 190
    version vectors, 191
  multi-datacenter, 184
  quorums, 179-182
    consistency limitations, 181-183, 334
    sloppy quorums and hinted handoff, 183
  read repair and anti-entropy, 178
leap seconds, 8, 290
  in time-of-day clocks, 288
leases, 295
  implementation with ZooKeeper, 370
  need for fencing, 302
ledgers, 460
  distributed ledger technologies, 532
legacy systems, maintenance of, 18
less (Unix tool), 397
LevelDB (storage engine), 78
leveled compaction, 79
Levenshtein automata, 88
limping (partial failure), 311
linearizability, 324-338, 555
  cost of, 335-338
    CAP theorem, 336
    memory on multi-core CPUs, 338
  definition, 325-329
  implementing with total order broadcast, 350
  in ZooKeeper, 370
  of derived data systems, 492, 524
    avoiding coordination, 527
  of different replication methods, 332-335
    using quorums, 334
  relying on, 330-332
    constraints and uniqueness, 330
    cross-channel timing dependencies, 331
    locking and leader election, 330
  stronger than causal consistency, 342
  using to implement total order broadcast, 351
  versus serializability, 329
LinkedIn
  Azkaban (workflow scheduler), 402
  Databus (change data capture), 161, 455
  Espresso (database), 31, 126, 130, 153, 216
  Helix (cluster manager) (see Helix)
  profile (example), 30
  reference to company entity (example), 34
  Rest.li (RPC framework), 135
  Voldemort (database) (see Voldemort)
Linux, leap second bug, 8, 290
liveness properties, 308
LMDB (storage engine), 82, 242
load
  approaches to coping with, 17
  describing, 11
  load testing, 16
load balancing (messaging), 444
local indexes (see document-partitioned indexes)
locality (data access), 32, 41, 555

in batch processing, 400, 405, 421
in stateful clients, 170, 511
in stream processing, 474, 478, 508, 522
location transparency, 134
in the actor model, 138
locks, 556
deadlock, 258
distributed locking, 301-304, 330
fencing tokens, 303
implementation with ZooKeeper, 370
relation to consensus, 374
for transaction isolation
in snapshot isolation, 239
in two-phase locking (2PL), 257-261
making operations atomic, 243
performance, 258
preventing dirty writes, 236
preventing phantoms with index-range locks, 260, 265
read locks (shared mode), 236, 258
shared mode and exclusive mode, 258
in two-phase commit (2PC)
deadlock detection, 364
in-doubt transactions holding locks, 362
materializing conflicts with, 251
preventing lost updates by explicit locking, 244
log sequence number, 156, 449
logic programming languages, 504
logical clocks, 293, 343, 494
for read-after-write consistency, 164
logical logs, 160
logs (data structure), 71, 556
advantages of immutability, 460
compaction, 73, 79, 456, 460
for stream operator state, 479
creating using total order broadcast, 349
implementing uniqueness constraints, 522
log-based messaging, 446-451
comparison to traditional messaging, 448, 451
consumer offsets, 449
disk space usage, 450
replaying old messages, 451, 496, 498
slow consumers, 450
using logs for message storage, 447
log-structured storage, 71-79
log-structured merge tree (see LSM-trees)
replication, 152, 158-161
change data capture, 454-457
(see also changelogs)
coordination with snapshot, 156
logical (row-based) replication, 160
statement-based replication, 158
trigger-based replication, 161
write-ahead log (WAL) shipping, 159
scalability limits, 493
loose coupling, 396, 419, 502
lost updates (see updates)
LSM-trees (indexes), 78-79
comparison to B-trees, 83-85
Lucene (storage engine), 79
building indexes in batch processes, 411
similarity search, 88
Luigi (workflow scheduler), 402
LWW (see last write wins)

M

machine learning
ethical considerations, 534
(see also ethics)
iterative processing, 424
models derived from training data, 505
statistical and numerical algorithms, 428
MADlib (machine learning toolkit), 428
magic scaling sauce, 18
Mahout (machine learning toolkit), 428
maintainability, 18-22, 489
defined, 23
design principles for software systems, 19
evolvability (see evolvability)
operability, 19
simplicity and managing complexity, 20
many-to-many relationships
in document model versus relational model, 39
modeling as graphs, 49
many-to-one and many-to-many relationships, 33-36
many-to-one relationships, 34
MapReduce (batch processing), 390, 399-400
accessing external services within job, 404, 412
comparison to distributed databases
designing for frequent faults, 417
diversity of processing models, 416
diversity of storage, 415

comparison to stream processing, 464
comparison to Unix, 413-414
disadvantages and limitations of, 419
fault tolerance, 406, 414, 422
higher-level tools, 403, 426
implementation in Hadoop, 400-403
the shuffle, 402
implementation in MongoDB, 46-48
machine learning, 428
map-side processing, 408-410
broadcast hash joins, 409
merge joins, 410
partitioned hash joins, 409
mapper and reducer functions, 399
materialization of intermediate state, 419-423
output of batch workflows, 411-413
building search indexes, 411
key-value stores, 412
reduce-side processing, 403-408
analysis of user activity events (example), 404
grouping records by same key, 406
handling skew, 407
sort-merge joins, 405
workflows, 402
marshalling (see encoding)
massively parallel processing (MPP), 216
comparison to composing storage technologies, 502
comparison to Hadoop, 414-418, 428
master-master replication (see multi-leader replication)
master-slave replication (see leader-based replication)
materialization, 556
aggregate values, 101
conflicts, 251
intermediate state (batch processing), 420-423
materialized views, 101
as derived data, 386, 499-504
maintaining, using stream processing, 467, 475
Maven (Java build tool), 428
Maxwell (change data capture), 455
mean, 14
media monitoring, 467
median, 14
meeting room booking (example), 249, 259, 521
membership services, 372
Memcached (caching server), 4, 89
memory
in-memory databases, 88
durability, 227
serial transaction execution, 253
in-memory representation of data, 112
random bit-flips in, 529
use by indexes, 72, 77
memory barrier (CPU instruction), 338
MemSQL (database)
in-memory storage, 89
read committed isolation, 236
memtable (in LSM-trees), 78
Mercurial (version control system), 463
merge joins, MapReduce map-side, 410
mergeable persistent data structures, 174
merging sorted files, 76, 402, 405
Merkle trees, 532
Mesos (cluster manager), 418, 506
message brokers (see messaging systems)
message-passing, 136-139
advantages over direct RPC, 137
distributed actor frameworks, 138
evolvability, 138
MessagePack (encoding format), 116
messages
exactly-once semantics, 360, 476
loss of, 442
using total order broadcast, 348
messaging systems, 440-451
(see also streams)
backpressure, buffering, or dropping messages, 441
brokerless messaging, 442
event logs, 446-451
comparison to traditional messaging, 448, 451
consumer offsets, 449
replaying old messages, 451, 496, 498
slow consumers, 450
message brokers, 443-446
acknowledgements and redelivery, 445
comparison to event logs, 448, 451
multiple consumers of same topic, 444
reliability, 442
uniqueness in log-based messaging, 522

Meteor (web framework), 456
microbatching, 477, 495
microservices, 132
  (see also services)
  causal dependencies across services, 493
  loose coupling, 502
  relation to batch/stream processors, 389, 508
Microsoft
  Azure Service Bus (messaging), 444
  Azure Storage, 155, 398
  Azure Stream Analytics, 466
  DCOM (Distributed Component Object Model), 134
  MSDTC (transaction coordinator), 356
  Orleans (see Orleans)
  SQL Server (see SQL Server)
migrating (rewriting) data, 40, 130, 461, 497
modulus operator (%), 210
MongoDB (database)
  aggregation pipeline, 48
  atomic operations, 243
  BSON, 41
  document data model, 31
  hash partitioning (sharding), 203-204
  key-range partitioning, 202
  lack of join support, 34, 42
  leader-based replication, 153
  MapReduce support, 46, 400
  oplog parsing, 455, 456
  partition splitting, 212
  request routing, 216
  secondary indexes, 207
Mongoriver (change data capture), 455
monitoring, 10, 19
monotonic clocks, 288
monotonic reads, 164
MPP (see massively parallel processing)
MSMQ (messaging), 361
multi-column indexes, 87
multi-leader replication, 168-177
  (see also replication)
  handling write conflicts, 171
    conflict avoidance, 172
    converging toward a consistent state, 172
    custom conflict resolution logic, 173
    determining what is a conflict, 174
  linearizability, lack of, 333
  replication topologies, 175-177
  use cases, 168
    clients with offline operation, 170
    collaborative editing, 170
    multi-datacenter replication, 168, 335
multi-object transactions, 228
  need for, 231
Multi-Paxos (total order broadcast), 367
multi-table index cluster tables (Oracle), 41
multi-tenancy, 284
multi-version concurrency control (MVCC), 239, 266
  detecting stale MVCC reads, 263
  indexes and snapshot isolation, 241
mutual exclusion, 261
  (see also locks)
MySQL (database)
  binlog coordinates, 156
  binlog parsing for change data capture, 455
  circular replication topology, 175
  consistent snapshots, 156
  distributed transaction support, 361
  InnoDB storage engine (see InnoDB)
  JSON support, 30, 42
  leader-based replication, 153
  performance of XA transactions, 360
  row-based replication, 160
  schema changes in, 40
  snapshot isolation support, 242
    (see also InnoDB)
  statement-based replication, 159
  Tungsten Replicator (multi-leader replication), 170
    conflict detection, 177

N

nanomsg (messaging library), 442
Narayana (transaction coordinator), 356
NATS (messaging), 137
near-real-time (nearline) processing, 390
  (see also stream processing)
Neo4j (database)
  Cypher query language, 52
  graph data model, 50
Nephele (dataflow engine), 421
netcat (Unix tool), 397
Netflix Chaos Monkey, 7, 280
Network Attached Storage (NAS), 146, 398
network model, 36

graph databases versus, 60
imperative query APIs, 46
Network Time Protocol (see NTP)
networks
congestion and queueing, 282
datacenter network topologies, 276
faults (see faults)
linearizability and network delays, 338
network partitions, 279, 337
timeouts and unbounded delays, 281
next-key locking, 260
nodes (in graphs) (see vertices)
nodes (processes), 556
handling outages in leader-based replication, 156
system models for failure, 307
noisy neighbors, 284
nonblocking atomic commit, 359
nondeterministic operations
accidental nondeterminism, 423
partial failures in distributed systems, 275
nonfunctional requirements, 22
nonrepeatable reads, 238
(see also read skew)
normalization (data representation), 33, 556
executing joins, 39, 42, 403
foreign key references, 231
in systems of record, 386
versus denormalization, 462
NoSQL, 29, 499
transactions and, 223
Notation3 (N3), 56
npm (package manager), 428
NTP (Network Time Protocol), 287
accuracy, 289, 293
adjustments to monotonic clocks, 289
multiple server addresses, 306
numbers, in XML and JSON encodings, 114

O

object-relational mapping (ORM) frameworks, 30
error handling and aborted transactions, 232
unsafe read-modify-write cycle code, 244
object-relational mismatch, 29
observer pattern, 506
offline systems, 390
(see also batch processing)
stateful, offline-capable clients, 170, 511
offline-first applications, 511
offsets
consumer offsets in partitioned logs, 449
messages in partitioned logs, 447
OLAP (online analytic processing), 91, 556
data cubes, 102
OLTP (online transaction processing), 90, 556
analytics queries versus, 411
workload characteristics, 253
one-to-many relationships, 30
JSON representation, 32
online systems, 389
(see also services)
Oozie (workflow scheduler), 402
OpenAPI (service definition format), 133
OpenStack
Nova (cloud infrastructure)
use of ZooKeeper, 370
Swift (object storage), 398
operability, 19
operating systems versus databases, 499
operation identifiers, 518, 522
operational transformation, 174
operators, 421
flow of data between, 424
in stream processing, 464
optimistic concurrency control, 261
Oracle (database)
distributed transaction support, 361
GoldenGate (change data capture), 161, 170, 455
lack of serializability, 226
leader-based replication, 153
multi-table index cluster tables, 41
not preventing write skew, 248
partitioned indexes, 209
PL/SQL language, 255
preventing lost updates, 245
read committed isolation, 236
Real Application Clusters (RAC), 330
recursive query support, 54
snapshot isolation support, 239, 242
TimesTen (in-memory database), 89
WAL-based replication, 160
XML support, 30
ordering, 339-352
by sequence numbers, 343-348
causal ordering, 339-343

partial order, 341
limits of total ordering, 493
total order broadcast, 348-352
Orleans (actor framework), 139
outliers (response time), 14
Oz (programming language), 504

P

package managers, 428, 505
packet switching, 285
packets
corruption of, 306
sending via UDP, 442
PageRank (algorithm), 49, 424
paging (see virtual memory)
ParAccel (database), 93
parallel databases (see massively parallel processing)
parallel execution
of graph analysis algorithms, 426
queries in MPP databases, 216
Parquet (data format), 96, 131
(see also column-oriented storage)
use in Hadoop, 414
partial failures, 275, 310
limping, 311
partial order, 341
partitioning, 199-218, 556
and replication, 200
in batch processing, 429
multi-partition operations, 514
enforcing constraints, 522
secondary index maintenance, 495
of key-value data, 201-205
by key range, 202
skew and hot spots, 205
rebalancing partitions, 209-214
automatic or manual rebalancing, 213
problems with hash mod N, 210
using dynamic partitioning, 212
using fixed number of partitions, 210
using N partitions per node, 212
replication and, 147
request routing, 214-216
secondary indexes, 206-209
document-based partitioning, 206
term-based partitioning, 208
serial execution of transactions and, 255
Paxos (consensus algorithm), 366
ballot number, 368
Multi-Paxos (total order broadcast), 367
percentiles, 14, 556
calculating efficiently, 16
importance of high percentiles, 16
use in service level agreements (SLAs), 15
Percona XtraBackup (MySQL tool), 156
performance
describing, 13
of distributed transactions, 360
of in-memory databases, 89
of linearizability, 338
of multi-leader replication, 169
perpetual inconsistency, 525
pessimistic concurrency control, 261
phantoms (transaction isolation), 250
materializing conflicts, 251
preventing, in serializability, 259
physical clocks (see clocks)
pickle (Python), 113
Pig (dataflow language), 419, 427
replicated joins, 409
skewed joins, 407
workflows, 403
Pinball (workflow scheduler), 402
pipelined execution, 423
in Unix, 394
point in time, 287
polyglot persistence, 29
polystores, 501
PostgreSQL (database)
BDR (multi-leader replication), 170
causal ordering of writes, 177
Bottled Water (change data capture), 455
Bucardo (trigger-based replication), 161, 173
distributed transaction support, 361
foreign data wrappers, 501
full text search support, 490
leader-based replication, 153
log sequence number, 156
MVCC implementation, 239, 241
PL/pgSQL language, 255
PostGIS geospatial indexes, 87
preventing lost updates, 245
preventing write skew, 248, 261
read committed isolation, 236
recursive query support, 54
representing graphs, 51

serializable snapshot isolation (SSI), 261
snapshot isolation support, 239, 242
WAL-based replication, 160
XML and JSON support, 30, 42
pre-splitting, 212
Precision Time Protocol (PTP), 290
predicate locks, 259
predictive analytics, 533-536
amplifying bias, 534
ethics of (see ethics)
feedback loops, 536
preemption
of datacenter resources, 418
of threads, 298
Pregel processing model, 425
primary keys, 85, 556
compound primary key (Cassandra), 204
primary-secondary replication (see leader-based replication)
privacy, 536-543
consent and freedom of choice, 538
data as assets and power, 540
deleting data, 463
ethical considerations (see ethics)
legislation and self-regulation, 542
meaning of, 539
surveillance, 537
tracking behavioral data, 536
probabilistic algorithms, 16, 466
process pauses, 295-299
processing time (of events), 469
producers (message streams), 440
programming languages
dataflow languages, 504
for stored procedures, 255
functional reactive programming (FRP), 504
logic programming, 504
Prolog (language), 61
(see also Datalog)
promises (asynchronous operations), 135
property graphs, 50
Cypher query language, 52
Protocol Buffers (data format), 117-121
field tags and schema evolution, 120
provenance of data, 531
publish/subscribe model, 441
publishers (message streams), 440
punch card tabulating machines, 390
pure functions, 48
putting computation near data, 400

Q

Qpid (messaging), 444
quality of service (QoS), 285
Quantcast File System (distributed filesystem), 398
query languages, 42-48
aggregation pipeline, 48
CSS and XSL, 44
Cypher, 52
Datalog, 60
Juttle, 504
MapReduce querying, 46-48
recursive SQL queries, 53
relational algebra and SQL, 42
SPARQL, 59
query optimizers, 37, 427
queueing delays (networks), 282
head-of-line blocking, 15
latency and response time, 14
queues (messaging), 137
quorums, 179-182, 556
for leaderless replication, 179
in consensus algorithms, 368
limitations of consistency, 181-183, 334
making decisions in distributed systems, 301
monitoring staleness, 182
multi-datacenter replication, 184
relying on durability, 309
sloppy quorums and hinted handoff, 183

R

R-trees (indexes), 87
RabbitMQ (messaging), 137, 444
leader-based replication, 153
race conditions, 225
(see also concurrency)
avoiding with linearizability, 331
caused by dual writes, 452
dirty writes, 235
in counter increments, 235
lost updates, 242-246
preventing with event logs, 462, 507
preventing with serializable isolation, 252
write skew, 246-251
Raft (consensus algorithm), 366

sensitivity to network problems, 369
term number, 368
use in etcd, 353
RAID (Redundant Array of Independent Disks), 7, 398
railways, schema migration on, 496
RAMCloud (in-memory storage), 89
ranking algorithms, 424
RDF (Resource Description Framework), 57
querying with SPARQL, 59
RDMA (Remote Direct Memory Access), 276
read committed isolation level, 234-237
implementing, 236
multi-version concurrency control (MVCC), 239
no dirty reads, 234
no dirty writes, 235
read path (derived data), 509
read repair (leaderless replication), 178
for linearizability, 335
read replicas (see leader-based replication)
read skew (transaction isolation), 238, 266
as violation of causality, 340
read-after-write consistency, 163, 524
cross-device, 164
read-modify-write cycle, 243
read-scaling architecture, 161
reads as events, 513
real-time
collaborative editing, 170
near-real-time processing, 390
(see also stream processing)
publish/subscribe dataflow, 513
response time guarantees, 298
time-of-day clocks, 288
rebalancing partitions, 209-214, 556
(see also partitioning)
automatic or manual rebalancing, 213
dynamic partitioning, 212
fixed number of partitions, 210
fixed number of partitions per node, 212
problems with hash mod N, 210
recency guarantee, 324
recommendation engines
batch process outputs, 412
batch workflows, 403, 420
iterative processing, 424
statistical and numerical algorithms, 428
records, 399
events in stream processing, 440
recursive common table expressions (SQL), 54
redelivery (messaging), 445
Redis (database)
atomic operations, 243
durability, 89
Lua scripting, 255
single-threaded execution, 253
usage example, 4
redundancy
hardware components, 7
of derived data, 386
(see also derived data)
Reed–Solomon codes (error correction), 398
refactoring, 22
(see also evolvability)
regions (partitioning), 199
register (data structure), 325
relational data model, 28-42
comparison to document model, 38-42
graph queries in SQL, 53
in-memory databases with, 89
many-to-one and many-to-many relationships, 33
multi-object transactions, need for, 231
NoSQL as alternative to, 29
object-relational mismatch, 29
relational algebra and SQL, 42
versus document model
convergence of models, 41
data locality, 41
relational databases
eventual consistency, 162
history, 28
leader-based replication, 153
logical logs, 160
philosophy compared to Unix, 499, 501
schema changes, 40, 111, 130
statement-based replication, 158
use of B-tree indexes, 80
relationships (see edges)
reliability, 6-10, 489
building a reliable system from unreliable components, 276
defined, 6, 22
hardware faults, 7
human errors, 9
importance of, 10
of messaging systems, 442

software errors, 8
Remote Method Invocation (Java RMI), 134
remote procedure calls (RPCs), 134-136
(see also services)
based on futures, 135
data encoding and evolution, 136
issues with, 134
using Avro, 126, 135
using Thrift, 135
versus message brokers, 137
repeatable reads (transaction isolation), 242
replicas, 152
replication, 151-193, 556
and durability, 227
chain replication, 155
conflict resolution and, 246
consistency properties, 161-167
consistent prefix reads, 165
monotonic reads, 164
reading your own writes, 162
in distributed filesystems, 398
leaderless, 177-191
detecting concurrent writes, 184-191
limitations of quorum consistency, 181-183, 334
sloppy quorums and hinted handoff, 183
monitoring staleness, 182
multi-leader, 168-177
across multiple datacenters, 168, 335
handling write conflicts, 171-175
replication topologies, 175-177
partitioning and, 147, 200
reasons for using, 145, 151
single-leader, 152-161
failover, 157
implementation of replication logs, 158-161
relation to consensus, 367
setting up new followers, 155
synchronous versus asynchronous, 153-155
state machine replication, 349, 452
using erasure coding, 398
with heterogeneous data systems, 453
replication logs (see logs)
reprocessing data, 496, 498
(see also evolvability)
from log-based messaging, 451
request routing, 214-216
approaches to, 214
parallel query execution, 216
resilient systems, 6
(see also fault tolerance)
response time
as performance metric for services, 13, 389
guarantees on, 298
latency versus, 14
mean and percentiles, 14
user experience, 15
responsibility and accountability, 535
REST (Representational State Transfer), 133
(see also services)
RethinkDB (database)
document data model, 31
dynamic partitioning, 212
join support, 34, 42
key-range partitioning, 202
leader-based replication, 153
subscribing to changes, 456
Riak (database)
Bitcask storage engine, 72
CRDTs, 174, 191
dotted version vectors, 191
gossip protocol, 216
hash partitioning, 203-204, 211
last-write-wins conflict resolution, 186
leaderless replication, 177
LevelDB storage engine, 78
linearizability, lack of, 335
multi-datacenter support, 184
preventing lost updates across replicas, 246
rebalancing, 213
search feature, 209
secondary indexes, 207
siblings (concurrently written values), 190
sloppy quorums, 184
ring buffers, 450
Ripple (cryptocurrency), 532
rockets, 10, 36, 305
RocksDB (storage engine), 78
leveled compaction, 79
rollbacks (transactions), 222
rolling upgrades, 8, 112
routing (see request routing)
row-oriented storage, 96
row-based replication, 160
rowhammer (memory corruption), 529
RPCs (see remote procedure calls)

Rubygems (package manager), 428
rules (Datalog), 61

S

safety and liveness properties, 308
  in consensus algorithms, 366
  in transactions, 222
sagas (see compensating transactions)
Samza (stream processor), 466, 467
  fault tolerance, 479
  streaming SQL support, 466
sandboxes, 9
SAP HANA (database), 93
scalability, 10-18, 489
  approaches for coping with load, 17
  defined, 22
  describing load, 11
  describing performance, 13
  partitioning and, 199
  replication and, 161
  scaling up versus scaling out, 146
scaling out, 17, 146
  (see also shared-nothing architecture)
scaling up, 17, 146
scatter/gather approach, querying partitioned databases, 207
SCD (slowly changing dimension), 476
schema-on-read, 39
  comparison to evolvable schema, 128
  in distributed filesystems, 415
schema-on-write, 39
schemaless databases (see schema-on-read)
schemas, 557
  Avro, 122-127
    reader determining writer's schema, 125
    schema evolution, 123
  dynamically generated, 126
  evolution of, 496
    affecting application code, 111
    compatibility checking, 126
    in databases, 129-131
    in message-passing, 138
    in service calls, 136
  flexibility in document model, 39
  for analytics, 93-95
  for JSON and XML, 115
  merits of, 127
  schema migration on railways, 496
  Thrift and Protocol Buffers, 117-121
    schema evolution, 120
  traditional approach to design, fallacy in, 462
searches
  building search indexes in batch processes, 411
  k-nearest neighbors, 429
  on streams, 467
  partitioned secondary indexes, 206
secondaries (see leader-based replication)
secondary indexes, 85, 557
  partitioning, 206-209, 217
    document-partitioned, 206
    index maintenance, 495
    term-partitioned, 208
  problems with dual writes, 452, 491
  updating, transaction isolation and, 231
secondary sorts, 405
sed (Unix tool), 392
self-describing files, 127
self-joins, 480
self-validating systems, 530
semantic web, 57
semi-synchronous replication, 154
sequence number ordering, 343-348
  generators, 294, 344
  insufficiency for enforcing constraints, 347
  Lamport timestamps, 345
  use of timestamps, 291, 295, 345
sequential consistency, 351
serializability, 225, 233, 251-266, 557
  linearizability versus, 329
  pessimistic versus optimistic concurrency control, 261
  serial execution, 252-256
    partitioning, 255
    using stored procedures, 253, 349
  serializable snapshot isolation (SSI), 261-266
    detecting stale MVCC reads, 263
    detecting writes that affect prior reads, 264
    distributed execution, 265, 364
    performance of SSI, 265
    preventing write skew, 262-265
  two-phase locking (2PL), 257-261
    index-range locks, 260
    performance, 258
Serializable (Java), 113

serialization, 113
  (see also encoding)
service discovery, 135, 214, 372
  using DNS, 216, 372
service level agreements (SLAs), 15
service-oriented architecture (SOA), 132
  (see also services)
services, 131-136
  microservices, 132
    causal dependencies across services, 493
    loose coupling, 502
  relation to batch/stream processors, 389, 508
  remote procedure calls (RPCs), 134-136
    issues with, 134
  similarity to databases, 132
  web services, 132, 135
session windows (stream processing), 472
  (see also windows)
sessionization, 407
sharding (see partitioning)
shared mode (locks), 258
shared-disk architecture, 146, 398
shared-memory architecture, 146
shared-nothing architecture, 17, 146-147, 557
  (see also replication)
  distributed filesystems, 398
    (see also distributed filesystems)
  partitioning, 199
  use of network, 277
sharks
  biting undersea cables, 279
  counting (example), 46-48
  finding (example), 42
  website about (example), 44
shredding (in relational model), 38
siblings (concurrent values), 190, 246
  (see also conflicts)
similarity search
  edit distance, 88
  genome data, 63
  k-nearest neighbors, 429
single-leader replication (see leader-based replication)
single-threaded execution, 243, 252
  in batch processing, 406, 421, 426
  in stream processing, 448, 463, 522
size-tiered compaction, 79
skew, 557
  clock skew, 291-294, 334
  in transaction isolation
    read skew, 238, 266
    write skew, 246-251, 262-265
      (see also write skew)
  meanings of, 238
  unbalanced workload, 201
    compensating for, 205
    due to celebrities, 205
    for time-series data, 203
    in batch processing, 407
slaves (see leader-based replication)
sliding windows (stream processing), 472
  (see also windows)
sloppy quorums, 183
  (see also quorums)
  lack of linearizability, 334
slowly changing dimension (data warehouses), 476
smearing (leap seconds adjustments), 290
snapshots (databases)
  causal consistency, 340
  computing derived data, 500
  in change data capture, 455
  serializable snapshot isolation (SSI), 261-266, 329
  setting up a new replica, 156
  snapshot isolation and repeatable read, 237-242
    implementing with MVCC, 239
    indexes and MVCC, 241
    visibility rules, 240
  synchronized clocks for global snapshots, 294
snowflake schemas, 95
SOAP, 133
  (see also services)
  evolvability, 136
software bugs, 8
  maintaining integrity, 529
solid state drives (SSDs)
  access patterns, 84
  detecting corruption, 519, 530
  faults in, 227
  sequential write throughput, 75
Solr (search server)
  building indexes in batch processes, 411
  document-partitioned indexes, 207
  request routing, 216

usage example, 4
use of Lucene, 79
sort (Unix tool), 392, 394, 395
sort-merge joins (MapReduce), 405
Sorted String Tables (see SSTables)
sorting
sort order in column storage, 99
source of truth (see systems of record)
Spanner (database)
data locality, 41
snapshot isolation using clocks, 295
TrueTime API, 294
Spark (processing framework), 421-423
bytecode generation, 428
dataflow APIs, 427
fault tolerance, 422
for data warehouses, 93
GraphX API (graph processing), 425
machine learning, 428
query optimizer, 427
Spark Streaming, 466
microbatching, 477
stream processing on top of batch processing, 495
SPARQL (query language), 59
spatial algorithms, 429
split brain, 158, 557
in consensus algorithms, 352, 367
preventing, 322, 333
using fencing tokens to avoid, 302-304
spreadsheets, dataflow programming capabilities, 504
SQL (Structured Query Language), 21, 28, 43
advantages and limitations of, 416
distributed query execution, 48
graph queries in, 53
isolation levels standard, issues with, 242
query execution on Hadoop, 416
résumé (example), 30
SQL injection vulnerability, 305
SQL on Hadoop, 93
statement-based replication, 158
stored procedures, 255
SQL Server (database)
data warehousing support, 93
distributed transaction support, 361
leader-based replication, 153
preventing lost updates, 245
preventing write skew, 248, 257
read committed isolation, 236
recursive query support, 54
serializable isolation, 257
snapshot isolation support, 239
T-SQL language, 255
XML support, 30
SQLstream (stream analytics), 466
SSDs (see solid state drives)
SSTables (storage format), 76-79
advantages over hash indexes, 76
concatenated index, 204
constructing and maintaining, 78
making LSM-Tree from, 78
staleness (old data), 162
cross-channel timing dependencies, 331
in leaderless databases, 178
in multi-version concurrency control, 263
monitoring for, 182
of client state, 512
versus linearizability, 324
versus timeliness, 524
standbys (see leader-based replication)
star replication topologies, 175
star schemas, 93-95
similarity to event sourcing, 458
Star Wars analogy (event time versus processing time), 469
state
derived from log of immutable events, 459
deriving current state from the event log, 458
interplay between state changes and application code, 507
maintaining derived state, 495
maintenance by stream processor in stream-stream joins, 473
observing derived state, 509-515
rebuilding after stream processor failure, 478
separation of application code and, 505
state machine replication, 349, 452
statement-based replication, 158
statically typed languages
analogy to schema-on-write, 40
code generation and, 127
statistical and numerical algorithms, 428
StatsD (metrics aggregator), 442
stdin, stdout, 395, 396
Stellar (cryptocurrency), 532

stock market feeds, 442
STONITH (Shoot The Other Node In The Head), 158
stop-the-world (see garbage collection)
storage
  composing data storage technologies, 499-504
  diversity of, in MapReduce, 415
Storage Area Network (SAN), 146, 398
storage engines, 69-104
  column-oriented, 95-101
    column compression, 97-99
    defined, 96
    distinction between column families and, 99
    Parquet, 96, 131
    sort order in, 99-100
    writing to, 101
  comparing requirements for transaction processing and analytics, 90-96
  in-memory storage, 88
    durability, 227
  row-oriented, 70-90
    B-trees, 79-83
    comparing B-trees and LSM-trees, 83-85
    defined, 96
    log-structured, 72-79
stored procedures, 161, 253-255, 557
  and total order broadcast, 349
  pros and cons of, 255
  similarity to stream processors, 505
Storm (stream processor), 466
  distributed RPC, 468, 514
  Trident state handling, 478
straggler events, 470, 498
stream processing, 464-481, 557
  accessing external services within job, 474, 477, 478, 517
  combining with batch processing
    lambda architecture, 497
    unifying technologies, 498
  comparison to batch processing, 464
  complex event processing (CEP), 465
  fault tolerance, 476-479
    atomic commit, 477
    idempotence, 478
    microbatching and checkpointing, 477
    rebuilding state after a failure, 478
  for data integration, 494-498
  maintaining derived state, 495
  maintenance of materialized views, 467
  messaging systems (see messaging systems)
  reasoning about time, 468-472
    event time versus processing time, 469, 477, 498
    knowing when window is ready, 470
    types of windows, 472
  relation to databases (see streams)
  relation to services, 508
  search on streams, 467
  single-threaded execution, 448, 463
  stream analytics, 466
  stream joins, 472-476
    stream-stream join, 473
    stream-table join, 473
    table-table join, 474
    time-dependence of, 475
streams, 440-451
  end-to-end, pushing events to clients, 512
  messaging systems (see messaging systems)
  processing (see stream processing)
  relation to databases, 451-464
    (see also changelogs)
    API support for change streams, 456
    change data capture, 454-457
    derivative of state by time, 460
    event sourcing, 457-459
    keeping systems in sync, 452-453
    philosophy of immutable events, 459-464
  topics, 440
strict serializability, 329
strong consistency (see linearizability)
strong one-copy serializability, 329
subjects, predicates, and objects (in triple-stores), 55
subscribers (message streams), 440
  (see also consumers)
supercomputers, 275
surveillance, 537
  (see also privacy)
Swagger (service definition format), 133
swapping to disk (see virtual memory)
synchronous networks, 285, 557
  comparison to asynchronous networks, 284
  formal model, 307
synchronous replication, 154, 557
  chain replication, 155

conflict detection, 172
system models, 300, 306-310
assumptions in, 528
correctness of algorithms, 308
mapping to the real world, 309
safety and liveness, 308
systems of record, 386, 557
change data capture, 454, 491
treating event log as, 460
systems thinking, 536

T

t-digest (algorithm), 16
table-table joins, 474
Tableau (data visualization software), 416
tail (Unix tool), 447
tail vertex (property graphs), 51
Tajo (query engine), 93
Tandem NonStop SQL (database), 200
TCP (Transmission Control Protocol), 277
comparison to circuit switching, 285
comparison to UDP, 283
connection failures, 280
flow control, 282, 441
packet checksums, 306, 519, 529
reliability and duplicate suppression, 517
retransmission timeouts, 284
use for transaction sessions, 229
telemetry (see monitoring)
Teradata (database), 93, 200
term-partitioned indexes, 208, 217
termination (consensus), 365
Terrapin (database), 413
Tez (dataflow engine), 421-423
fault tolerance, 422
support by higher-level tools, 427
thrashing (out of memory), 297
threads (concurrency)
actor model, 138, 468
(see also message-passing)
atomic operations, 223
background threads, 73, 85
execution pauses, 286, 296-298
memory barriers, 338
preemption, 298
single (see single-threaded execution)
three-phase commit, 359
Thrift (data format), 117-121
BinaryProtocol, 118
CompactProtocol, 119
field tags and schema evolution, 120
throughput, 13, 390
TIBCO, 137
Enterprise Message Service, 444
StreamBase (stream analytics), 466
time
concurrency and, 187
cross-channel timing dependencies, 331
in distributed systems, 287-299
(see also clocks)
clock synchronization and accuracy, 289
relying on synchronized clocks, 291-295
process pauses, 295-299
reasoning about, in stream processors, 468-472
event time versus processing time, 469, 477, 498
knowing when window is ready, 470
timestamp of events, 471
types of windows, 472
system models for distributed systems, 307
time-dependence in stream joins, 475
time-of-day clocks, 288
timeliness, 524
coordination-avoiding data systems, 528
correctness of dataflow systems, 525
timeouts, 279, 557
dynamic configuration of, 284
for failover, 158
length of, 281
timestamps, 343
assigning to events in stream processing, 471
for read-after-write consistency, 163
for transaction ordering, 295
insufficiency for enforcing constraints, 347
key range partitioning by, 203
Lamport, 345
logical, 494
ordering events, 291, 345
Titan (database), 50
tombstones, 74, 191, 456
topics (messaging), 137, 440
total order, 341, 557
limits of, 493
sequence numbers or timestamps, 344
total order broadcast, 348-352, 493, 522
consensus algorithms and, 366-368

implementation in ZooKeeper and etcd, 370
implementing with linearizable storage, 351
using, 349
using to implement linearizable storage, 350
tracking behavioral data, 536
(see also privacy)
transaction coordinator (see coordinator)
transaction manager (see coordinator)
transaction processing, 28, 90-95
comparison to analytics, 91
comparison to data warehousing, 93
transactions, 221-267, 558
ACID properties of, 223
atomicity, 223
consistency, 224
durability, 226
isolation, 225
compensating (see compensating transactions)
concept of, 222
distributed transactions, 352-364
avoiding, 492, 502, 521-528
failure amplification, 364, 495
in doubt/uncertain status, 358, 362
two-phase commit, 354-359
use of, 360-361
XA transactions, 361-364
OLTP versus analytics queries, 411
purpose of, 222
serializability, 251-266
actual serial execution, 252-256
pessimistic versus optimistic concurrency control, 261
serializable snapshot isolation (SSI), 261-266
two-phase locking (2PL), 257-261
single-object and multi-object, 228-232
handling errors and aborts, 231
need for multi-object transactions, 231
single-object writes, 230
snapshot isolation (see snapshots)
weak isolation levels, 233-251
preventing lost updates, 242-246
read committed, 234-238
transitive closure (graph algorithm), 424
trie (data structure), 88
triggers (databases), 161, 441
implementing change data capture, 455
implementing replication, 161
triple-stores, 55-59
SPARQL query language, 59
tumbling windows (stream processing), 472
(see also windows)
in microbatching, 477
tuple spaces (programming model), 507
Turtle (RDF data format), 56
Twitter
constructing home timelines (example), 11, 462, 474, 511
DistributedLog (event log), 448
Finagle (RPC framework), 135
Snowflake (sequence number generator), 294
Summingbird (processing library), 497
two-phase commit (2PC), 353, 355-359, 558
confusion with two-phase locking, 356
coordinator failure, 358
coordinator recovery, 363
how it works, 357
issues in practice, 363
performance cost, 360
transactions holding locks, 362
two-phase locking (2PL), 257-261, 329, 558
confusion with two-phase commit, 356
index-range locks, 260
performance of, 258
type checking, dynamic versus static, 40

U

UDP (User Datagram Protocol)
comparison to TCP, 283
multicast, 442
unbounded datasets, 439, 558
(see also streams)
unbounded delays, 558
in networks, 282
process pauses, 296
unbundling databases, 499-515
composing data storage technologies, 499-504
federation versus unbundling, 501
need for high-level language, 503
designing applications around dataflow, 504-509
observing derived state, 509-515
materialized views and caching, 510
multi-partition data processing, 514
pushing state changes to clients, 512

uncertain (transaction status) (see in doubt)
uniform consensus, 365
  (see also consensus)
uniform interfaces, 395
union type (in Avro), 125
uniq (Unix tool), 392
uniqueness constraints
  asynchronously checked, 526
  requiring consensus, 521
  requiring linearizability, 330
  uniqueness in log-based messaging, 522
Unix philosophy, 394-397
  command-line batch processing, 391-394
    Unix pipes versus dataflow engines, 423
  comparison to Hadoop, 413-414
  comparison to relational databases, 499, 501
  comparison to stream processing, 464
  composability and uniform interfaces, 395
  loose coupling, 396
  pipes, 394
  relation to Hadoop, 499
UPDATE statement (SQL), 40
updates
  preventing lost updates, 242-246
    atomic write operations, 243
    automatically detecting lost updates, 245
    compare-and-set operations, 245
    conflict resolution and replication, 246
    using explicit locking, 244
  preventing write skew, 246-251

V

validity (consensus), 365
vBuckets (partitioning), 199
vector clocks, 191
  (see also version vectors)
vectorized processing, 99, 428
verification, 528-533
  avoiding blind trust, 530
  culture of, 530
  designing for auditability, 531
  end-to-end integrity checks, 531
  tools for auditable data systems, 532
version control systems, reliance on immutable data, 463
version vectors, 177, 191
  capturing causal dependencies, 343
  versus vector clocks, 191
Vertica (database), 93
  handling writes, 101
  replicas using different sort orders, 100
vertical scaling (see scaling up)
vertices (in graphs), 49
  property graph model, 50
Viewstamped Replication (consensus algorithm), 366
  view number, 368
virtual machines, 146
  (see also cloud computing)
  context switches, 297
  network performance, 282
  noisy neighbors, 284
  reliability in cloud services, 8
  virtualized clocks in, 290
virtual memory
  process pauses due to page faults, 14, 297
  versus memory management by databases, 89
VisiCalc (spreadsheets), 504
vnodes (partitioning), 199
Voice over IP (VoIP), 283
Voldemort (database)
  building read-only stores in batch processes, 413
  hash partitioning, 203-204, 211
  leaderless replication, 177
  multi-datacenter support, 184
  rebalancing, 213
  reliance on read repair, 179
  sloppy quorums, 184
VoltDB (database)
  cross-partition serializability, 256
  deterministic stored procedures, 255
  in-memory storage, 89
  output streams, 456
  secondary indexes, 207
  serial execution of transactions, 253
  statement-based replication, 159, 479
  transactions in stream processing, 477

W

WAL (write-ahead log), 82
web services (see services)
Web Services Description Language (WSDL), 133
webhooks, 443
webMethods (messaging), 137
WebSocket (protocol), 512

windows (stream processing), 466, 468-472
  infinite windows for changelogs, 467, 474
  knowing when all events have arrived, 470
  stream joins within a window, 473
  types of windows, 472
winners (conflict resolution), 173
WITH RECURSIVE syntax (SQL), 54
workflows (MapReduce), 402
  outputs, 411-414
    key-value stores, 412
    search indexes, 411
  with map-side joins, 410
working set, 393
write amplification, 84
write path (derived data), 509
write skew (transaction isolation), 246-251
  characterizing, 246-251, 262
  examples of, 247, 249
  materializing conflicts, 251
  occurrence in practice, 529
  phantoms, 250
  preventing
    in snapshot isolation, 262-265
    in two-phase locking, 259-261
    options for, 248
write-ahead log (WAL), 82, 159
writes (database)
  atomic write operations, 243
  detecting writes affecting prior reads, 264
  preventing dirty writes with read committed, 235
WS-* framework, 133
  (see also services)
WS-AtomicTransaction (2PC), 355

X

XA transactions, 355, 361-364
  heuristic decisions, 363
  limitations of, 363
xargs (Unix tool), 392, 396
XML
  binary variants, 115
  encoding RDF data, 57
  for application data, issues with, 114
  in relational databases, 30, 41
XSL/XPath, 45

Y

Yahoo!
  Pistachio (database), 461
  Sherpa (database), 455
YARN (job scheduler), 416, 506
  preemption of jobs, 418
  use of ZooKeeper, 370

Z

Zab (consensus algorithm), 366
  use in ZooKeeper, 353
ZeroMQ (messaging library), 442
ZooKeeper (coordination service), 370-373
  generating fencing tokens, 303, 349, 370
  linearizable operations, 333, 351
  locks and leader election, 330
  service discovery, 372
  use for partition assignment, 215, 371
  use of Zab algorithm, 349, 353, 366

About the Author

Martin Kleppmann is a researcher in distributed systems at the University of Cambridge, UK. Previously he was a software engineer and entrepreneur at internet companies including LinkedIn and Rapportive, where he worked on large-scale data infrastructure. In the process he learned a few things the hard way, and he hopes this book will save you from repeating the same mistakes.

Martin is a regular conference speaker, blogger, and open source contributor. He believes that profound technical ideas should be accessible to everyone, and that deeper understanding will help us develop better software.

Colophon

The animal on the cover of *Designing Data-Intensive Applications* is an Indian wild boar (*Sus scrofa cristatus*), a subspecies of wild boar found in India, Myanmar, Nepal, Sri Lanka, and Thailand. They are distinctive from European boars in that they have higher back bristles, no woolly undercoat, and a larger, straighter skull.

The Indian wild boar has a coat of gray or black hair, with stiff bristles running along the spine. Males have protruding canine teeth (called tushes) that are used to fight with rivals or fend off predators. Males are larger than females, but the species averages 33–35 inches tall at the shoulder and 200–300 pounds in weight. Their natural predators include bears, tigers, and various big cats.

These animals are nocturnal and omnivorous—they eat a wide variety of things, including roots, insects, carrion, nuts, berries, and small animals. Wild boars are also known to root through garbage and crop fields, causing a great deal of destruction and earning the enmity of farmers. They need to eat 4,000–4,500 calories a day. Boars have a well-developed sense of smell, which helps them forage for underground plant material and burrowing animals. However, their eyesight is poor.

Wild boars have long held significance in human culture. In Hindu lore, the boar is an avatar of the god Vishnu. In ancient Greek funerary monuments, it was a symbol of a gallant loser (in contrast to the victorious lion). Due to its aggression, it was depicted on the armor and weapons of Scandinavian, Germanic, and Anglo-Saxon warriors. In the Chinese zodiac, it symbolizes determination and impetuosity.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from Shaw's *Zoology*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the font in diagrams is Adobe Myriad Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.