Distributed Data Processing

Big Data Management

Knowledge objectives

1. Explain the CAP theorem
2. Identify the 3 configuration alternatives given by the CAP theorem
3. Explain the 4 synchronization protocols we can have
4. Explain what eventually consistency means
5. Enumerate the phases of distributed query processing
6. Explain the difference between data shipping and query shipping
7. Explain the meaning of "reconstruction" and "reduction" in syntactic optimization
8. Explain the purpose of the "exchange" operator in physical optimization
9. Enumerate the 4 different cost factors in distributed query processing
10. Distinguish between response time and query cost
11. Explain the different kinds of parallelism
12. Identify the impact of fragmentation in intra-operator parallelism
13. Explain the impact of tree topologies (i.e., linear and bushy) in inter-operator parallelism
14. Explain the limits of scalability

Understanding objectives

1. Given the overall number of machines in the cluster, identify the consistency problems that arise depending on the configuration of the number of required replicas read and written to confirm the corresponding operations
2. Given a parallel system and a workload find the number of machines maximizing throughput (i.e., find the optimal number of machines using the Universal Scalability Law)
3. Estimate the cost of a distributed query

Application objectives

1. Given a query and a database design, recognize the difficulties and opportunities behind distributed query processing

Distributed Transaction Management

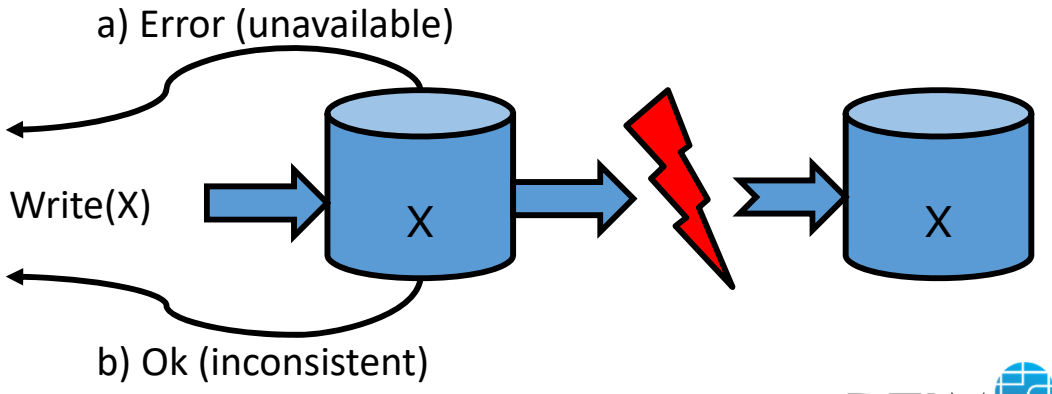
Problem III

CAP theorem

"We can only achieve two of Consistency, system Availability, and tolerance to network Partition."

Eric Brewer

- Consistency (C) equivalent to a single up-to-date copy of the data
- High availability (A) of the data (for updates)
- Tolerance to network partitions (P).



Configuration alternatives

a) Strong consistency
   - Replicas are synchonously modified and guarantee consistent query answering
   - The whole system will be declared not to be available in case of network partition

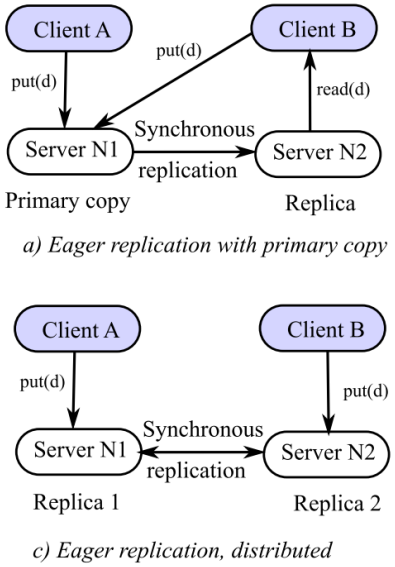
b) Eventually consistent
   - Changes are asynchronously propagated to replicas so answer to the same query depends on the replica being used
   - In case of network partition, changes will be simply delayed

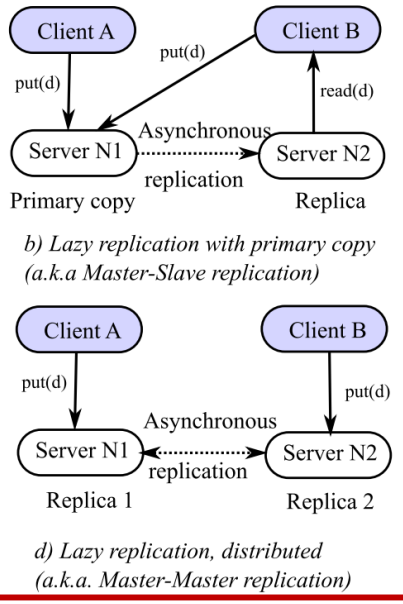
c) Non-distributed data
   - Connectivity cannot be lost
   - We can have strong consistency without affecting availability

Managing replicas

- Replicating fragments improves query latency and availability
  - Requires dealing with consistency and update (a.k.a., synchronization) performance
- Replication protocols characteristics
  - Primary – Secondary versioning
  - Eager – Lazy replication

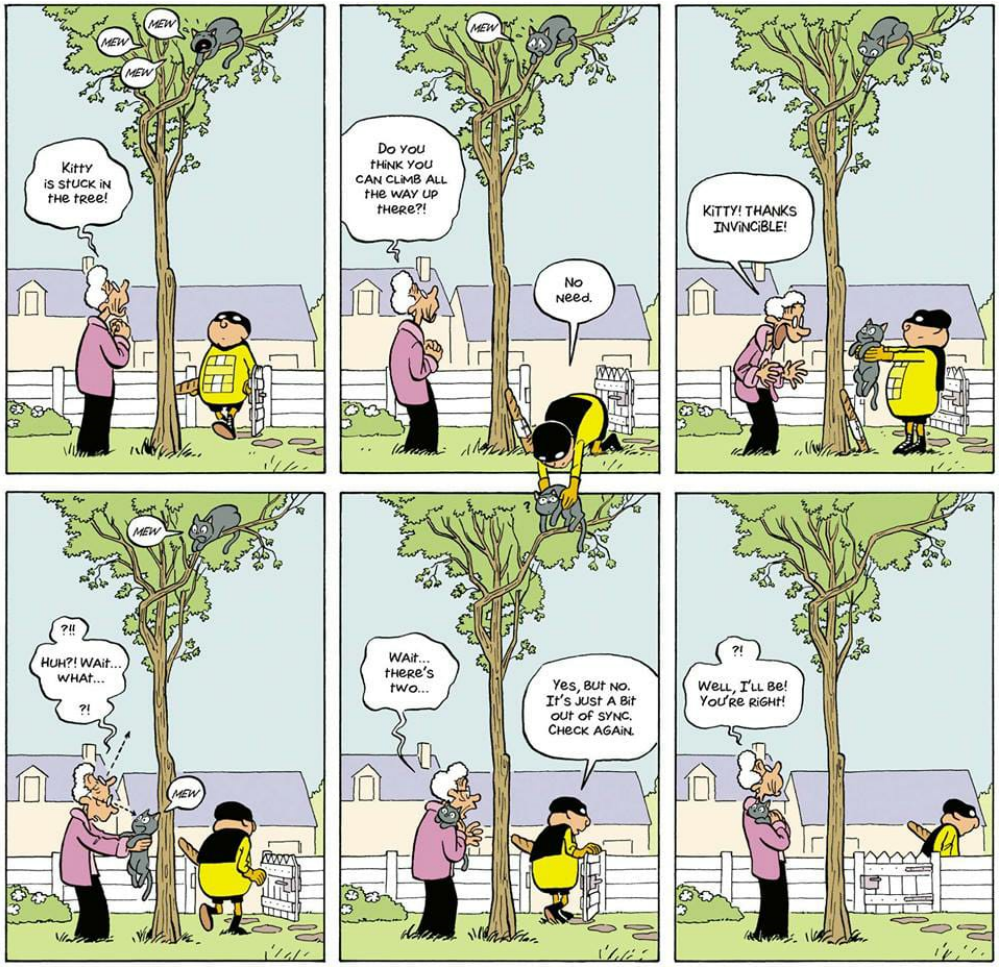
Strong Consistency





Eventually Consistent

Eventual consistency



Justin Travis Waith-Mair

Replication management configuration

- Definitions
  - N: #replicas
  - W: #replicas that have to be written
  - R: #replicas that need to be read
- Situations
  - Inconsistency window ⇒ W<N
  - Strong consistency ⇒ W+R>N
  - Eventually consistent ⇒ R+W<=N
    - Sets of machines (R and W) may not overlap
  - Potential conflict ⇒ W<(N+1)/2
- Typical configurations
  - Fault tolerant system ⇒ N=3; R=2; W=2
  - Massive replication for read scaling ⇒ R=1
  - Rear One-Write All (ROWA) ⇒ R=1; W=N
    - Fast read
    - Slow write (low probability of succeeding)

Distributed query processing

Problem IV

Challenges in distributed query processing

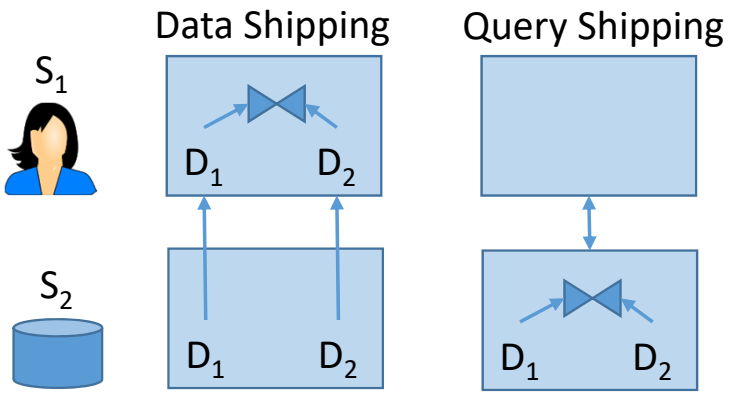
- Communication cost (data shipping)
  - Not that critical for LAN networks
    - Assuming high enough I/O cost
- Fragmentation / Replication
  - Metadata and statistics about fragments (and replicas) in the global catalog
- Join Optimization
  - Joins order
  - Semi-join strategy
- How to decide the execution plan
  - Who executes what
  - Exploit parallelism (!)

A centralized optimizer reduces the number of accesses to disk

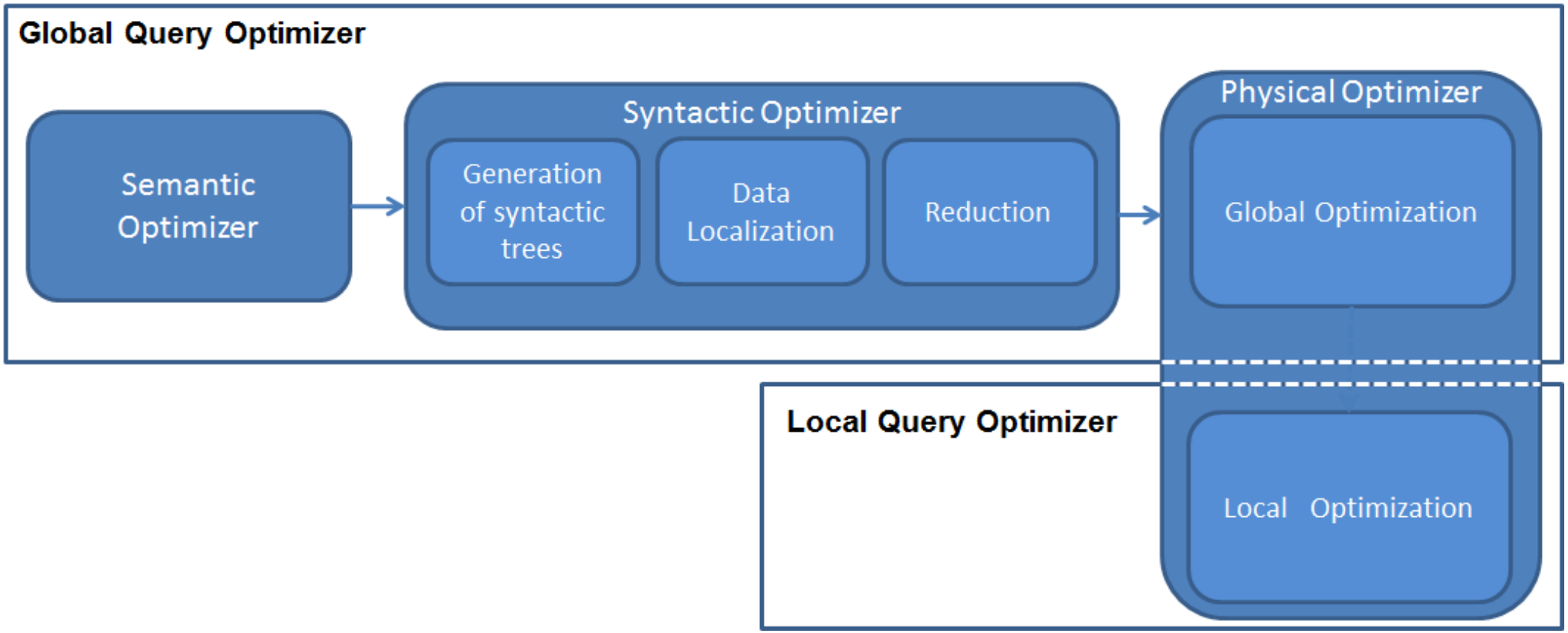
The distributed optimizer reduces the use of network bandwidth

Allocation selection

- Data shipping
  - The data is retrieved from the stored site to the site executing the query
    - Avoid bottlenecks on frequently used data
- Query shipping
  - The evaluation of the query is delegated to the site where it is stored
    - To avoid transferring large amounts of data
- Hybrid strategy
  - Dynamically decide data or query shipping

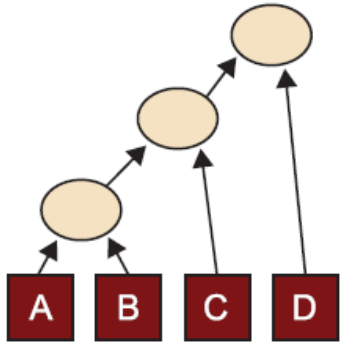


Phases of distributed query processing



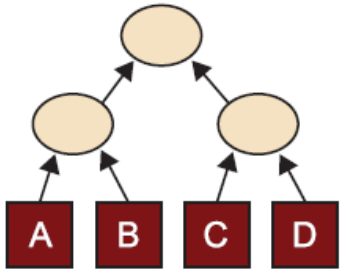
Syntactic optimizer

- Ordering
  - Left or right deep trees
  - Bushy trees



left-deep tree

Hard to parallelize



bushy tree

Easy to parallelize

- Site selection (exploit Data Location)
  - More difficult for joins (multi-way joins)
  - Comparing size of the datasets
    - Size of the intermediate joins must be considered
- Algorithms to process the query operators
  - Parallelism (!)

Physical optimizer

- Transforms an internal query representation into an efficient plan
  - Replaces the logical query operators by specific algorithms (plan operators) and access methods
  - Decides in which order to execute them
- This is done by…
  - Enumerating alternative but equivalent *plans*
  - Estimating their costs
  - Searching for the best solution
    - Using available statistics regarding the physical state of the system

Criteria to choose the access plan

- Usually with the goal to optimize response time
  - Time needed to execute a query
  - Benefits from parallelism
- Cost Model
  - Sum of local cost and communication cost
    - Local cost
      - Cost of central unit processing (#cycles),
      - Unit cost of I/O operation (#I/O ops)
    - Communication cost (commonly assumed it is linear in the number of bytes transmitted)
      - Cost of initiating a message and sending a message (#messages)
      - Cost of transmitting one byte (#bytes)
  - Knowledge required
    - Size of elementary data units processed
    - Selectivity of operations to estimate intermediate results

Cost model example

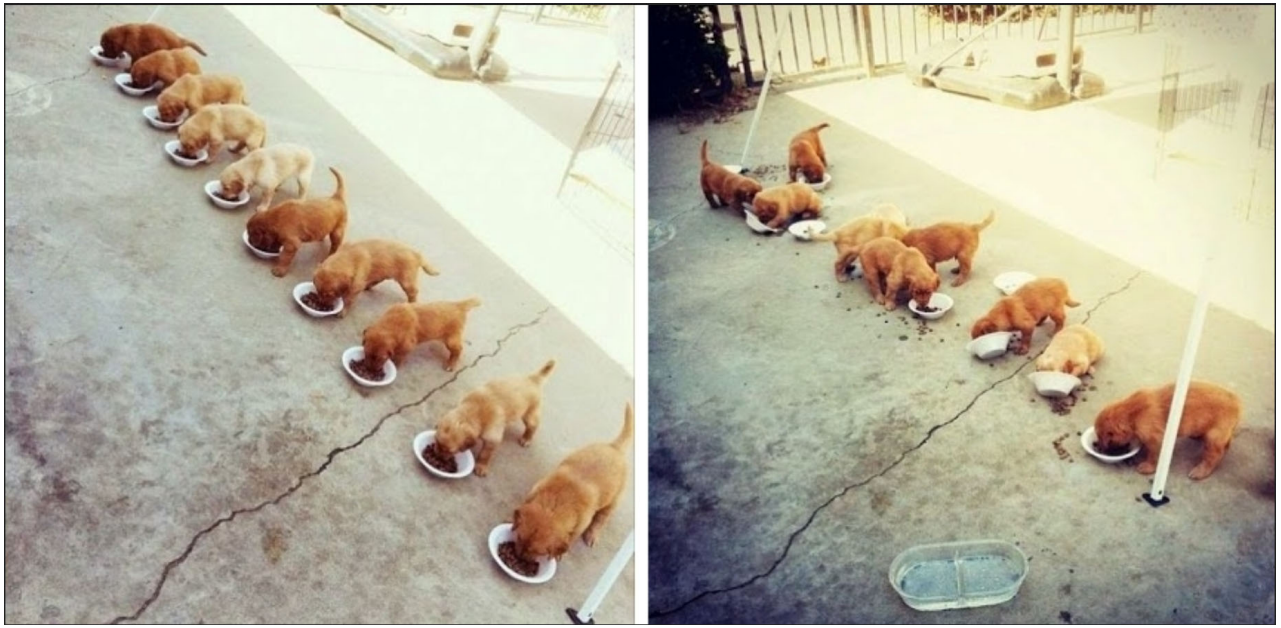
- Parameters:
  - Local processing:
    - Average CPU time to process an instance (T_{cpu})
    - Number of instances processed (#inst)
    - I/O time per operation ($T_{I/O}$)
    - Number of I/O operations (#I/Os)
  - Global processing:
    - Message time (T_{Msg})
    - Number of messages issued (#msgs)
    - Transfer time (send a byte from one site to another) (T_{TR})
    - Number of bytes transferred (#bytes)
- Calculations:

$$\text{Resources Used} = W_{cpu} * T_{cpu} * \#inst + W_{I/O} * T_{I/O} * \#I/Os + W_{Msg} * T_{Msg} * \#msgs + W_{TR} * T_{TR} * \#bytes$$

$$\text{Response Time} = T_{cpu} * seq_{\#inst} + T_{I/O} * seq_{\#I/Os} + T_{Msg} * seq_{\#msgs} + T_{TR} * seq_{\#bytes}$$

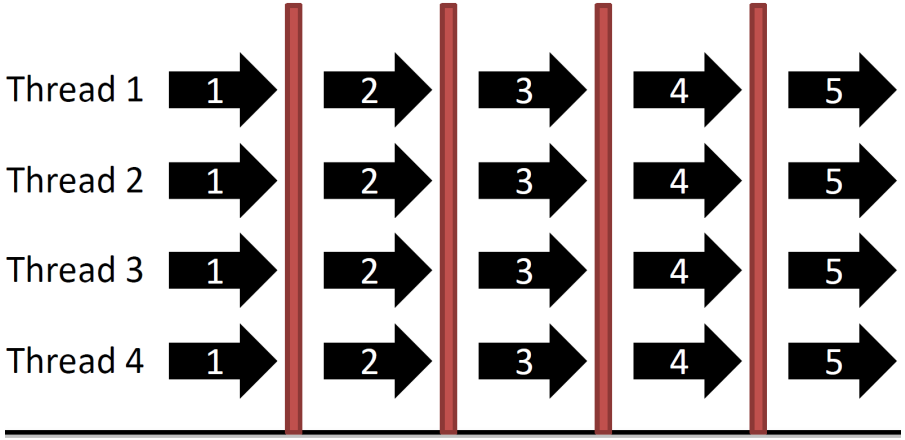
The problem of parallelism

Theory Practice



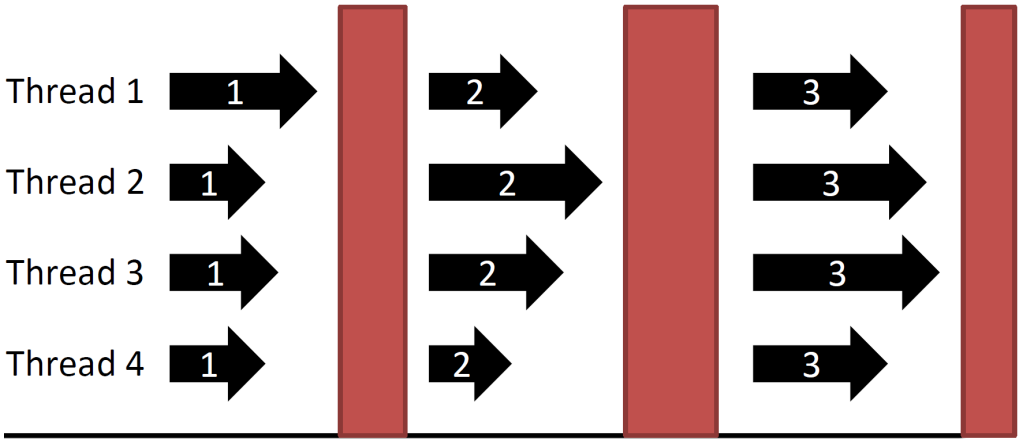
Samuel Yee

Bulk Synchronous Parallel Model



Ideal

Real



SAILING lab slides

Kinds of parallelism

- Inter-query
- Intra-query
  - Intra-operator
    - Unary
      - Static partitioning
    - Binary
      - Static or dynamic partitioning
  - Inter-operator
    - Independent
    - Pipelined

Demand-Driven Pipelining

- Each operator supports:
  - Open
  - Next
  - Close
- In principle, not parallel
  - Parent requests activate the execution
  - Nevertheless, a buffer can be used
    - This is similar to producer-driven

Producer-Driven Pipelining

- Generate output tuples eagerly
  - Until the buffers become full
- Pipeline stalls when
  a) An operator becomes ready and no new inputs are available
     - It is propagated downstream through the pipeline like a bubble
  b) An operator has input data available, but its output buffer is full
     - It is propagated upstream through the pipeline like a bubble

		Occupancy	Latency
Serial		T	T
Parallel	No stalls	T/N	h·T/N
	Stalls	T/N+k	h·(T/N+k)

N = operators in the pipeline
T = time units required for the whole query
Latency = time to process the query
Occupancy = time until it can accept more work
h = height of the process tree
k = delay imposed by imbalance and communication overhead

Measures for parallelism

Ahmdal's law and the Universal Scalability Law

Parallel processing

- Goal
  - Reduce response time / increase throughput
    - Employ parallel hardware effectively
- Means
  - Process pieces of input in different processors
    a) Divide the dataset into (disjoint) subsets
    b) Adapt serial algorithms to multi-thread environments

Measuring scalability

- Scalability is normally measured in terms of
  - Speed-up – measuring performance when adding hardware for a constant problem size
    - Linear speed-up means that N sites solve in T/N time, a problem solved in T time by a sequential version of the code
  - Scale-up - measuring performance when the problem size is altered with resources
    - Linear scale-up means that N sites solve a problem N times bigger in T time, the same code run in 1 site solves the same problem also in T time

Amdahl's law

- Principle - parallel computing with many processors is useful only for highly parallelizable programs
- Amdahl's law measures the maximum improvement possible by improving a particular part of a system
  - Sequential/serial processing vs.
  - Parallel processing
- $S(p, N)$ - theoretical speed-up, where p is the fraction of parallelizable code using N processors (hardware)

Amdahl's law

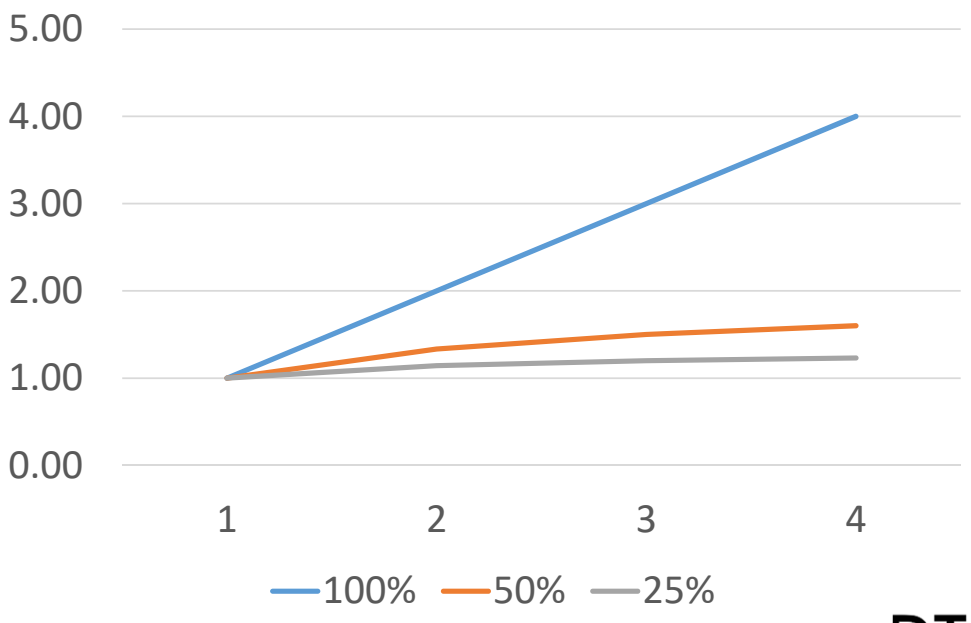
$$S(p,N) = \frac{N}{N - p(N-1)}$$

$$S(p,N) = \frac{N}{N - pN + p}$$

$$S(p,N) = \frac{1}{\frac{N-pN+p}{N}}$$

$$S(p,N) = \frac{1}{1 - p + \frac{p}{N}}$$

	100%	50%	25%	
1	1,00	1,00	1,00	
2	2,00	1,33	1,14	
3	3,00	1,50	1,20	
4	4,00	1,60	1,23	



Universal Scalability Law (I) – Generalization of Amdahl's law

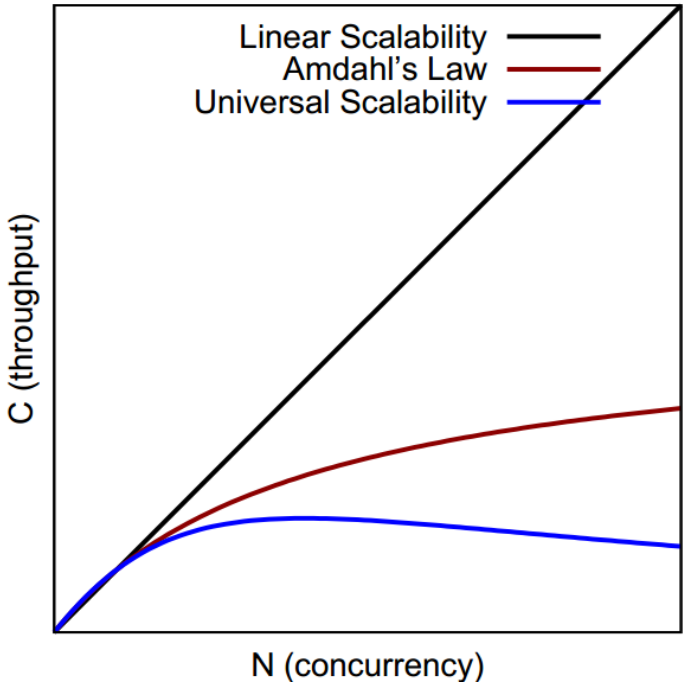
- Mathematical definition of scalability (both for SW or HW scalability)
  - Shows that linear scalability is hardly achievable
- USL is defined as follows

$$C(N) = \frac{N}{1 + \sigma(N-1) + \kappa N(N-1)}$$

- C: System's capacity (i.e., throughput) improvement (e.g., increment of #queries per second)
- N: System's size (SW - number of concurrent threads)/(HW - number of CPUs)
- σ: System's contention (performance degradation due to serial instead of parallel processing)
  - Could be avoided (i.e., σ = 0) if our code has no serial chunks (everything parallelizable)
- κ: System's consistency delay (aka coherency delay), extra work needed to keep synchronized shared data (i.e., inter-process communication)
  - Could be avoided (i.e., κ = 0) if replicas can be synchronized without sending messages

Universal Scalability Law (II)

- If κ = 0, it simplifies to Amdahl's law
- If both σ = 0 and κ = 0, we obtain linear scalability



https://www.percona.com/sites/default/files/white-paper-forecasting-mysql-scalability.pdf

USL application method

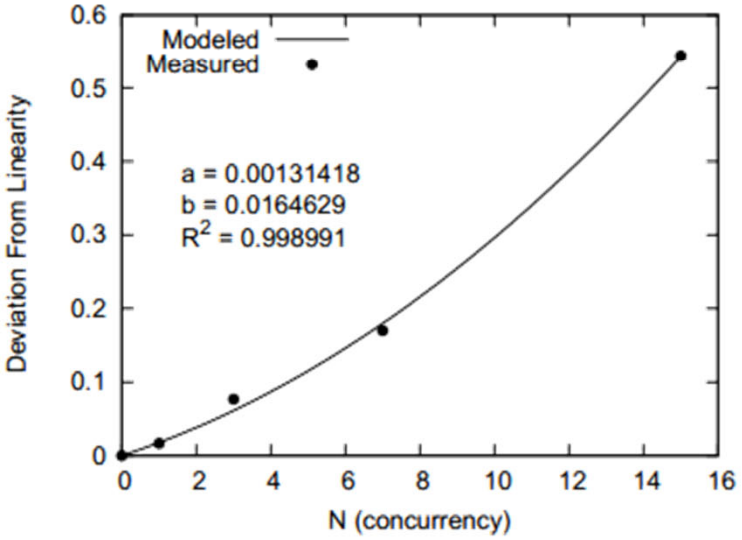
1. Empirical analysis: Compute C (throughput) for different values of N (concurrency)
2. Perform least-squares regression against gathered data
   1. $x = N - 1$
   2. $y = \frac{N}{C(N)} - 1$, where
   3. Fit a second-order polynomial $C(N) = \frac{C_N}{C_1}$
3. Reverse the transformation $y = ax^2 + bx + 0$
   1. $\sigma = b - a$ (contention)
   2. $\kappa = a$ (consistency delay)

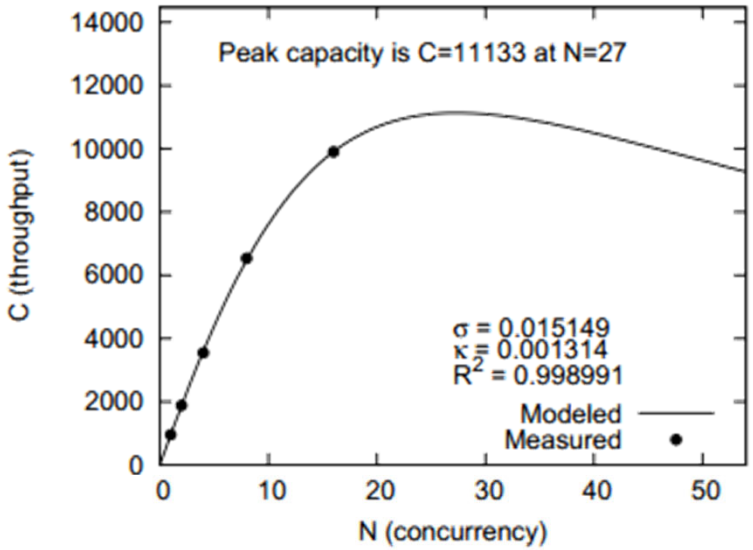
USL application method: Example

Concurrency (N)	Throughput (C)
1	955.16
2	1878.91
4	3548.68
8	6531.08
16	9897.24

N	C	$N-1$	$\frac{N}{C/C(1)}-1$
1	955.16	0	0.0000
2	1878.91	1	0.0167
4	3548.68	3	0.0766
8	6531.08	7	0.1699
16	9897.24	15	0.5441

Points are fit in a second-order polynomial: ax^2+bx+0, and a and b are computed





σ and κ are next computed from a and b; and we can plot the USL function

https://www.percona.com/sites/default/files/white-paper-forecasting-mysql-scalability.pdf

Closing

Summary

- Distributed Query Processing
  - Kinds of Parallelism
  - Cost estimation
- Data access
  - Random access vs. sequential reads
- Scalability measures
  - Amdahl's law
  - Universal scalability law

References

- G. Graefe. *Query Evaluation Techniques*. In ACM Computing Surveys, 25(2), 1993
- L. Liu, M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*, 3rd Ed. Springer, 2011
- L. G. Valiant. *A bridging model for parallel computation*. Commun. ACM. August 1990