Distributed Data Processing

Big Data Management

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

DTIM www.essi.upc.edu/dtim

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

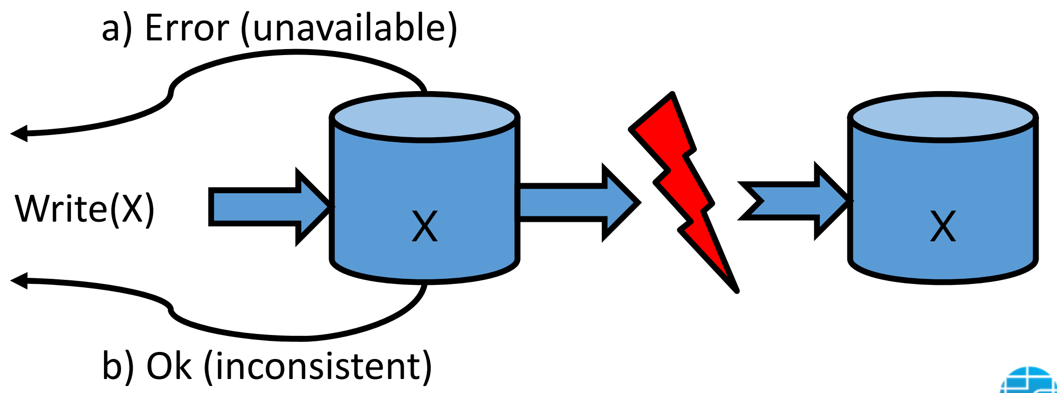
Challenge III

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 (give availability)

Give up Availability
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 (give consistency)

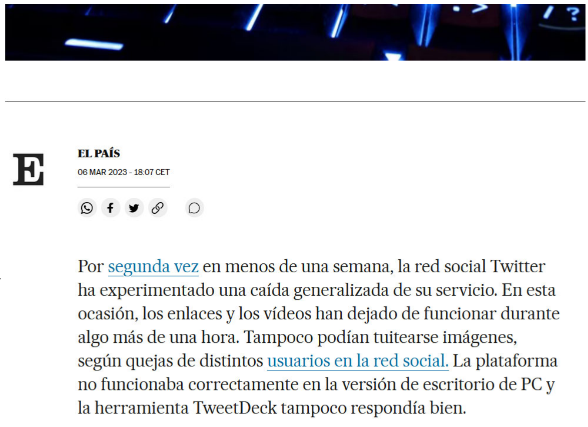
Give up Consistency
Strong Availability

- 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 (give network partitioning)

- Connectivity cannot be lost
- We can have strong consistency without affecting availability

No network partitioning
Achieve both consistency and availability



E

EL PAÍS

06 MAR 2023 - 18:07 CET

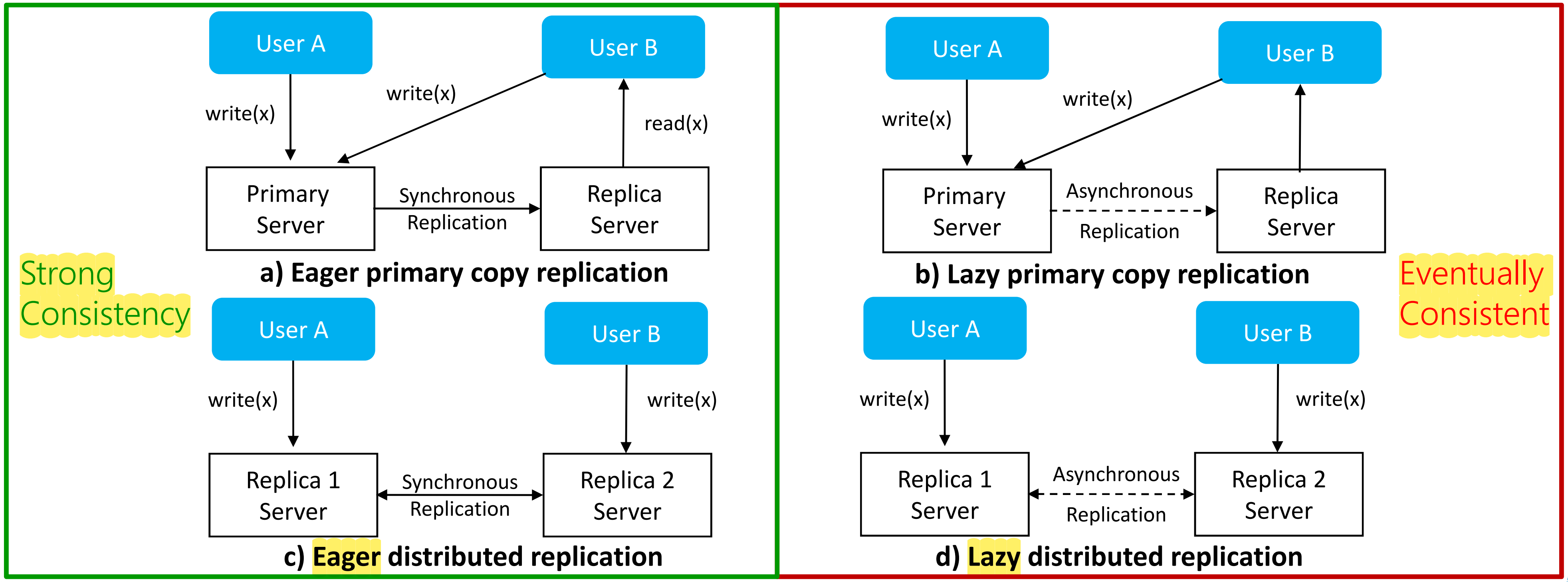
Por segunda vez en menos de una semana, la red social Twitter ha experimentado una caída generalizada de su servicio. En esta ocasión, los enlaces y los vídeos han dejado de funcionar durante algo más de una hora. Tampoco podían tuitearse imágenes, según quejas de distintos usuarios en la red social. La plataforma no funcionaba correctamente en la versión de escritorio de PC y la herramienta TweetDeck tampoco respondía bien.

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

DTIM www.essi.upc.edu/dtim

Managing replicas

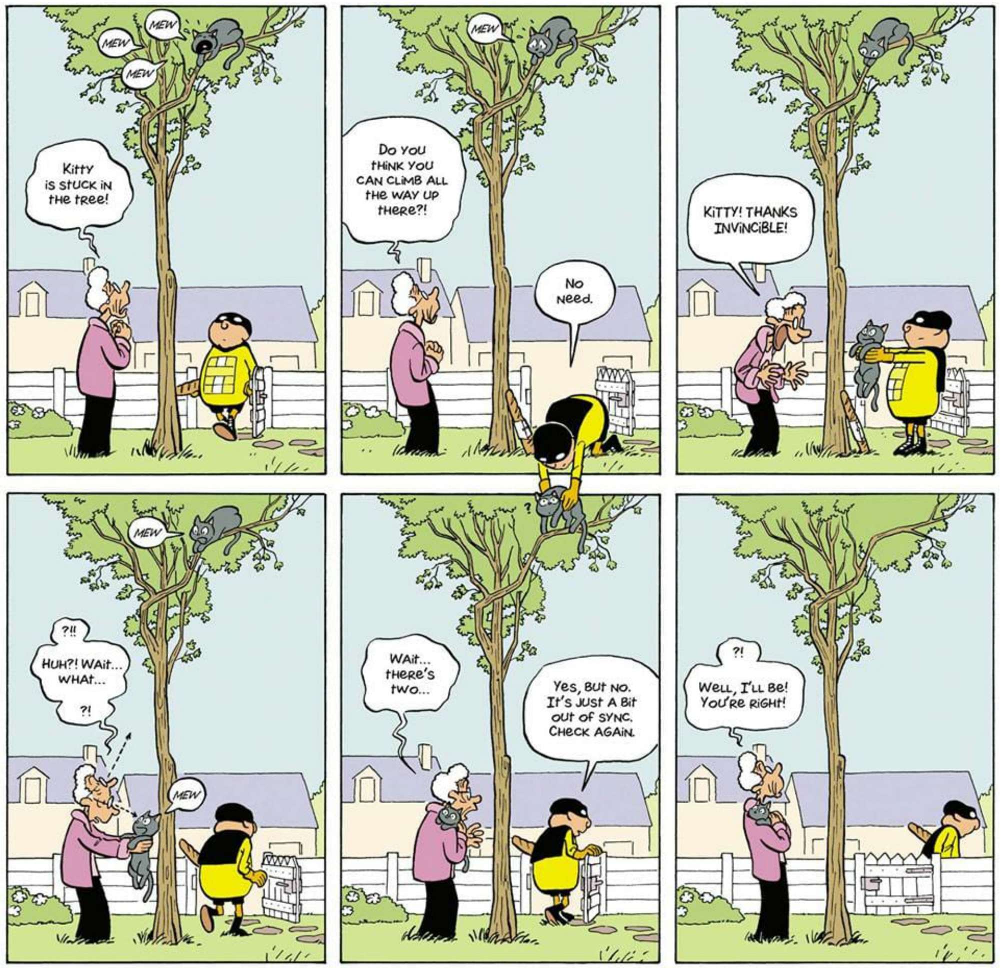
- Replicating fragments improves query latency and availability
  - Requires dealing with consistency and update (a.k.a., synchronization) performance
- Replication protocols characteristics
  - Primary – Distributed versioning
  - Eager – Lazy replication



Traditional Concept of Consistency

New Concept of Consistency

Eventual consistency


MEW
MEW
MEW
Kitty is stuck in the tree!
MEW
Do you think you can climb all the way up there?!
No need.
Kitty! Thanks Invincible!
MEW
?!!
Huh?! Wait... what...
?!
MEW
Wait... there's two...
Yes, but no. It's just a bit out of sync. Check again.
?!
Well, I'll be! You're right!

Justin Travis Waith-Mair

Replication management configuration

- Definitions
  - N: #replicas Number of replicas
  - W: #replicas that have to be written before commit Number of replicas in which write needs to happen before confirming to client
  - R: #replicas read that need to coincide before giving response R number of reads from replica should coincide before telling to client that this is the data
    ** No just reading form one server and saying this is result
- Named situations
  - Inconsistency window ⇒ $W<N$ Confirmed writing before reaching all replicas
  - Strong consistency ⇒ $R+W>N$
  - Eventually consistent ⇒ $R+W<=N$
    - Sets of machines (R and W) may not overlap
    - Potential conflict ⇒ $W<(N+1)/2$ Worst case
      Writing in less than half of the machine
      If one user write in one machine, another writes in another machine problem
      - Sets of writing machines (W) may not overlap
- Typical configurations
  - Fault tolerant system ⇒ N=3; W=2; R=2
  - Massive replication for read scaling ⇒ R=1
  - Read One-Write All (ROWA) ⇒ R=1; W=N (1+N>N ⇒ Strong consistency)
    - Fast read
    - Slow write (low probability of succeeding)

(Distributed) Query Processing

Challenge IV

Challenges in distributed query processing

Most Complex Problem

- 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
  - Exploit parallelism (!)
    - Who executes what

A centralized optimizer minimizes the number of accesses to disk

A distributed optimizer minimizes the use of network bandwidth

Allocation selection

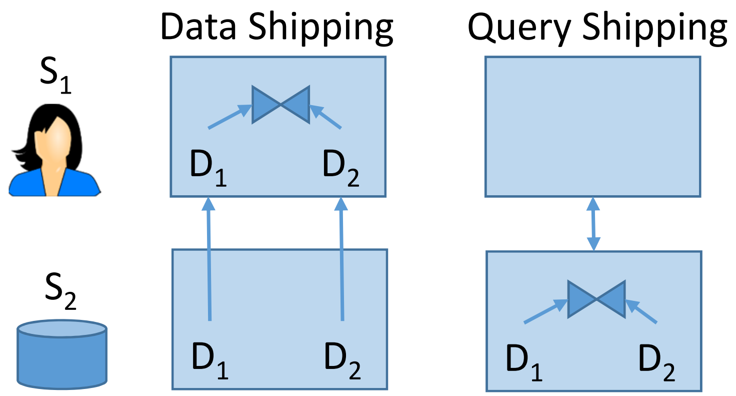
- Data shipping Send data where the query is
  - The data is retrieved from the stored site to the site executing the query
    - Avoid bottlenecks on frequently used data

  Useful when Result of operation is larger than its inputs

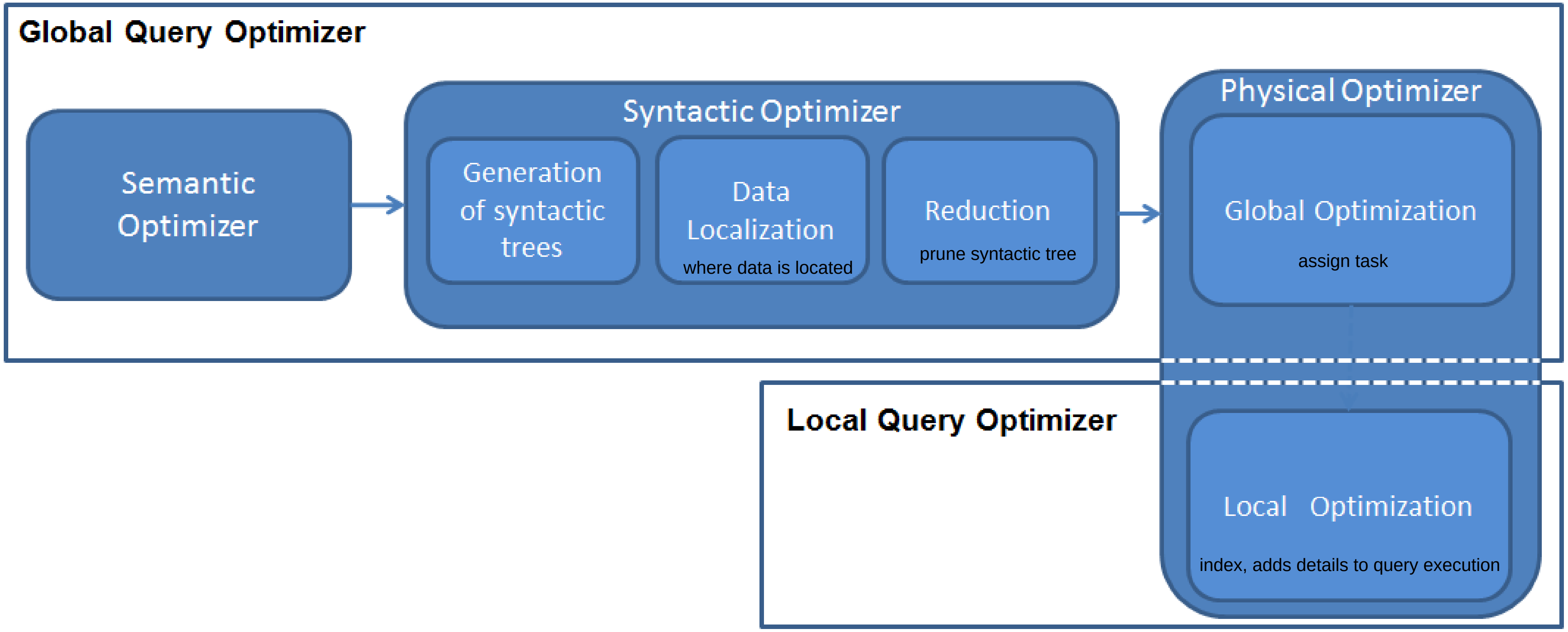
- Query shipping Send query where the data is present
  - The evaluation of the query is delegated to the site where it is stored
    - Avoid transferring large amounts of data

  Useful when inputs are larger than result

- Hybrid strategy
  - Dynamically decide data or query shipping

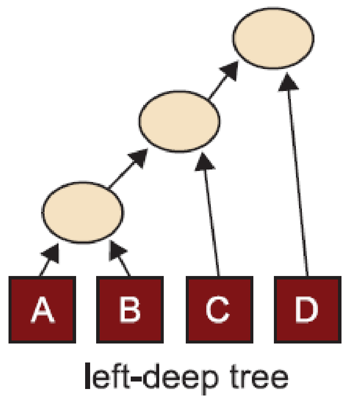


Phases of distributed query processing

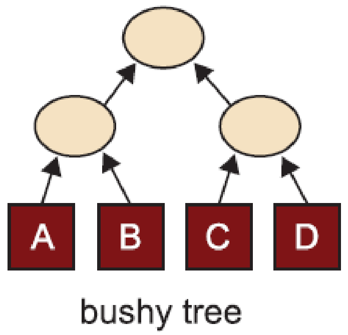


Syntactic optimizer

- Ordering operators
  - Left or right deep trees
  - Bushy trees



Hard to parallelize



Easy to parallelize

- Added difficulties
  - Consider multi-way joins
  - Consider the size of the datasets
    - Specially the size of the intermediate joins

Physical optimizer

- Transforms an internal query representation into an efficient plan
  - Replaces the logical query operators by
    1) Specific algorithms (plan operators)
    2) Access methods
  - Decides in which order to execute them
    - Parallelism (!)
  - Selects where to execute them (exploit Data Location)
    - More difficult for joins (multi-way joins)
- 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

Latency - Time taken to fetch first row
Response Time - Time taken to fetch last row

- Usually with the goal to optimize response time
  - Time needed to execute a query (i.e., latency or response time)
  - 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 — Size of different elements like rows, attributes
    - Selectivity of operations to estimate intermediate results — What % of data is selected by intermediate query?

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:

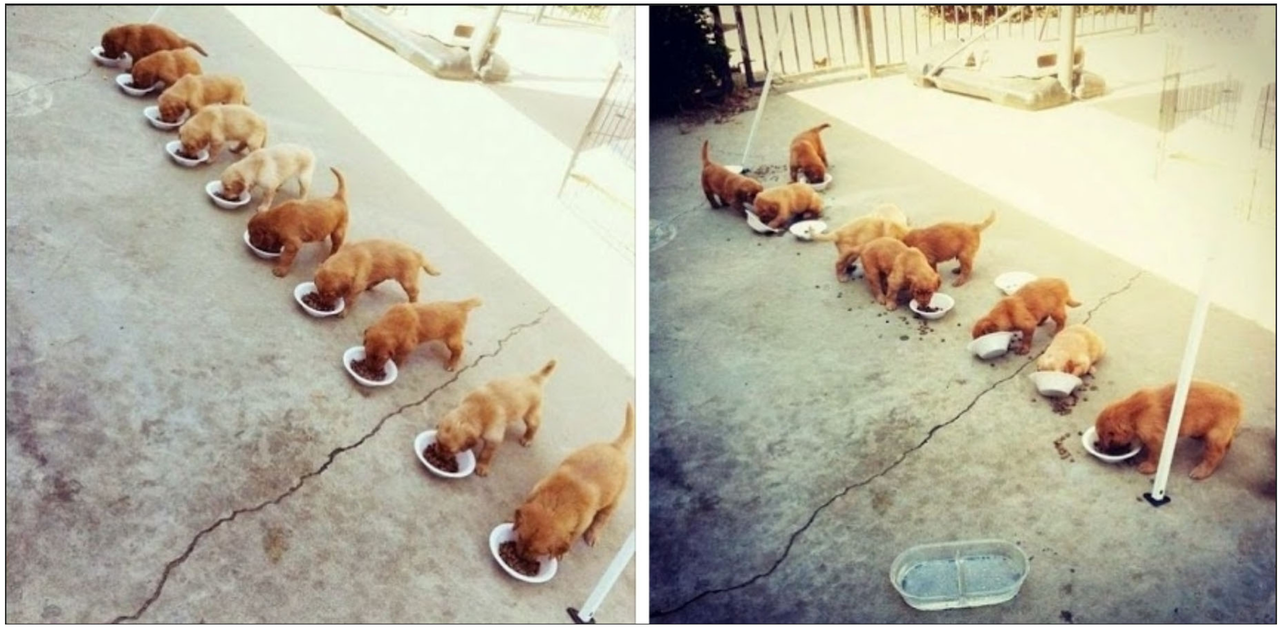
$$\text{Resources Used} = W_{cpu} * T_{cpu} * \#inst + W_{I/O} * T_{I/O} * \#I/Os + W_{Msg} * T_{Msg} * \#msgs + W_{TR} * T_{TR} * \#bytes$$

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

Response Time need to consider parallelism
Knowing what is going to happen in parallel apriori is not that easy

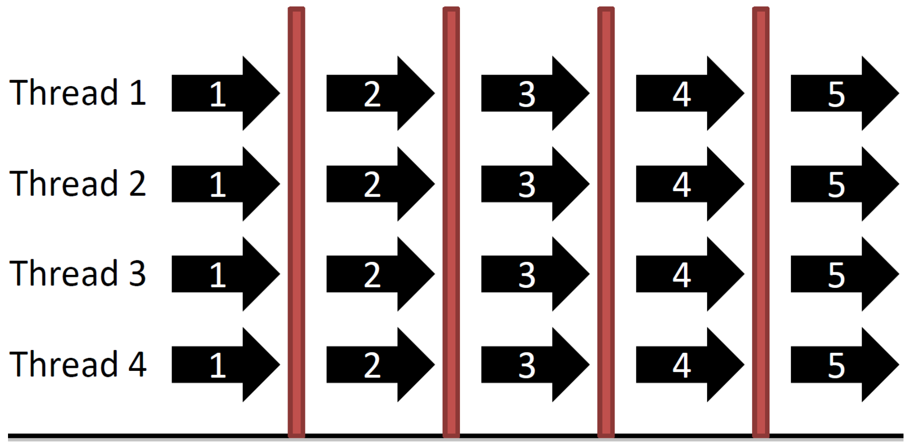
The problem of parallelism

Theory Practice



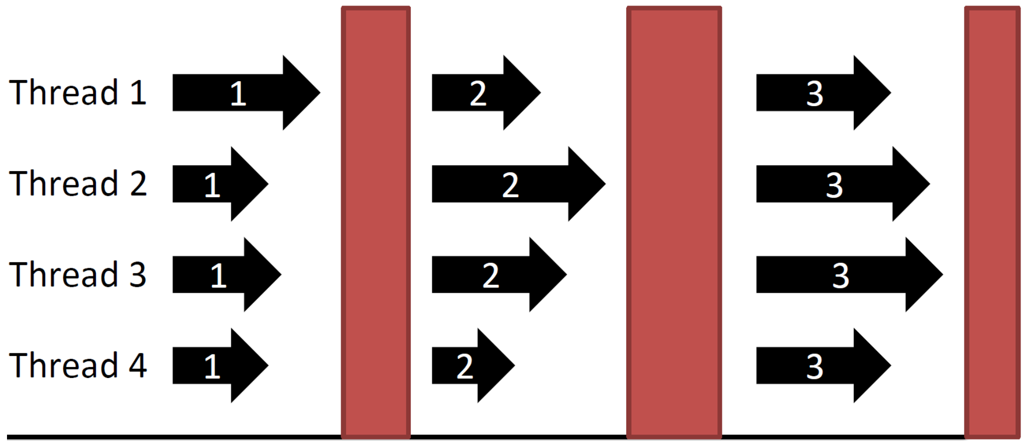
Samuel Yee

Bulk Synchronous Parallel Model



Ideal

Real



Everybody has to wait for the lowest thread

Kinds of parallelism

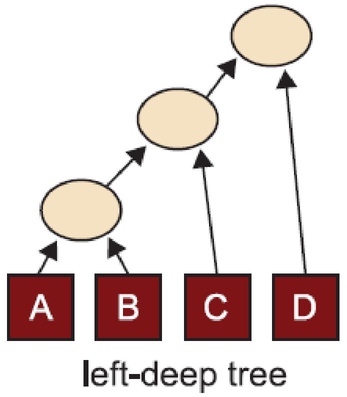
- Inter-query Different queries can be executed in parallel
- Intra-query Parallelism inside the query
  - Intra-operator
    - Unary
      - Static partitioning
    - Binary
      - Static or dynamic partitioning
  - Inter-operator
    - Independent
    - Pipelined

if operators are independent
one of one is not input of another
easy to achieve parallelism

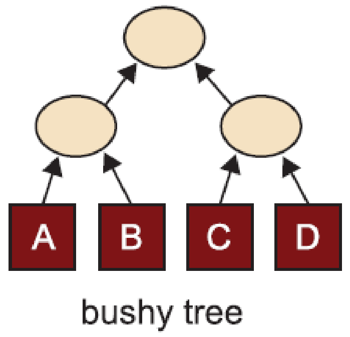
input of one operator depends on output of another operator
parallelism is difficult

but if processing is happening in row level,
still possible to achieve parallelism

In Pipelined Inter Operator query
Looks like parallelism not possible but it is possible
Parallelism not possible only if it is blocking



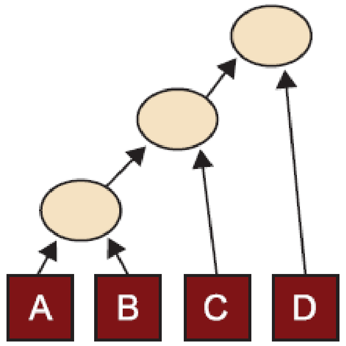
Pipelined



Independent operators

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



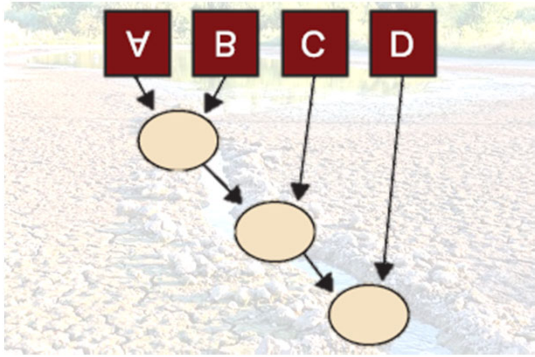
left-deep tree

Producer-Driven Pipelining

tuples are pushed from leaf to the root of tree

- 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	Execution time
Serial			T	T
Inter-operator Parallelism	Independent	No stalls	T/N	h·T/N
	Pipelining	Stalls	T/N+k	T/N+k+(h-1)(T/N)/|R|

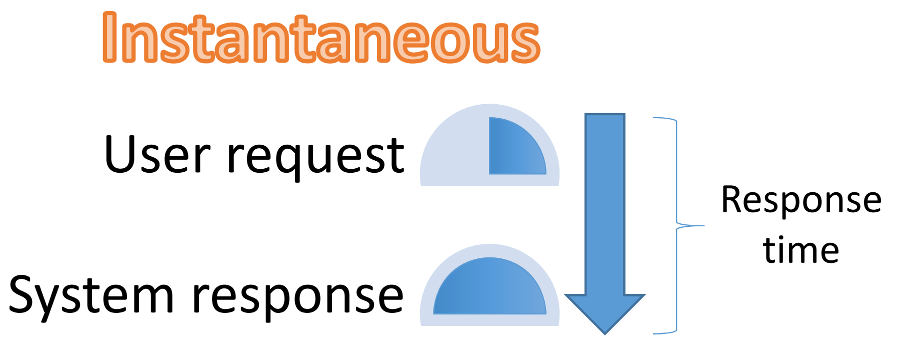
T = time units required for the whole query
N = operators in the process tree
|R| = tuples to be processed
Execution time = time to process the query
Occupancy = time until it can accept more work
h = height of the process tree
k = delay imposed by imbalance and communication overhead

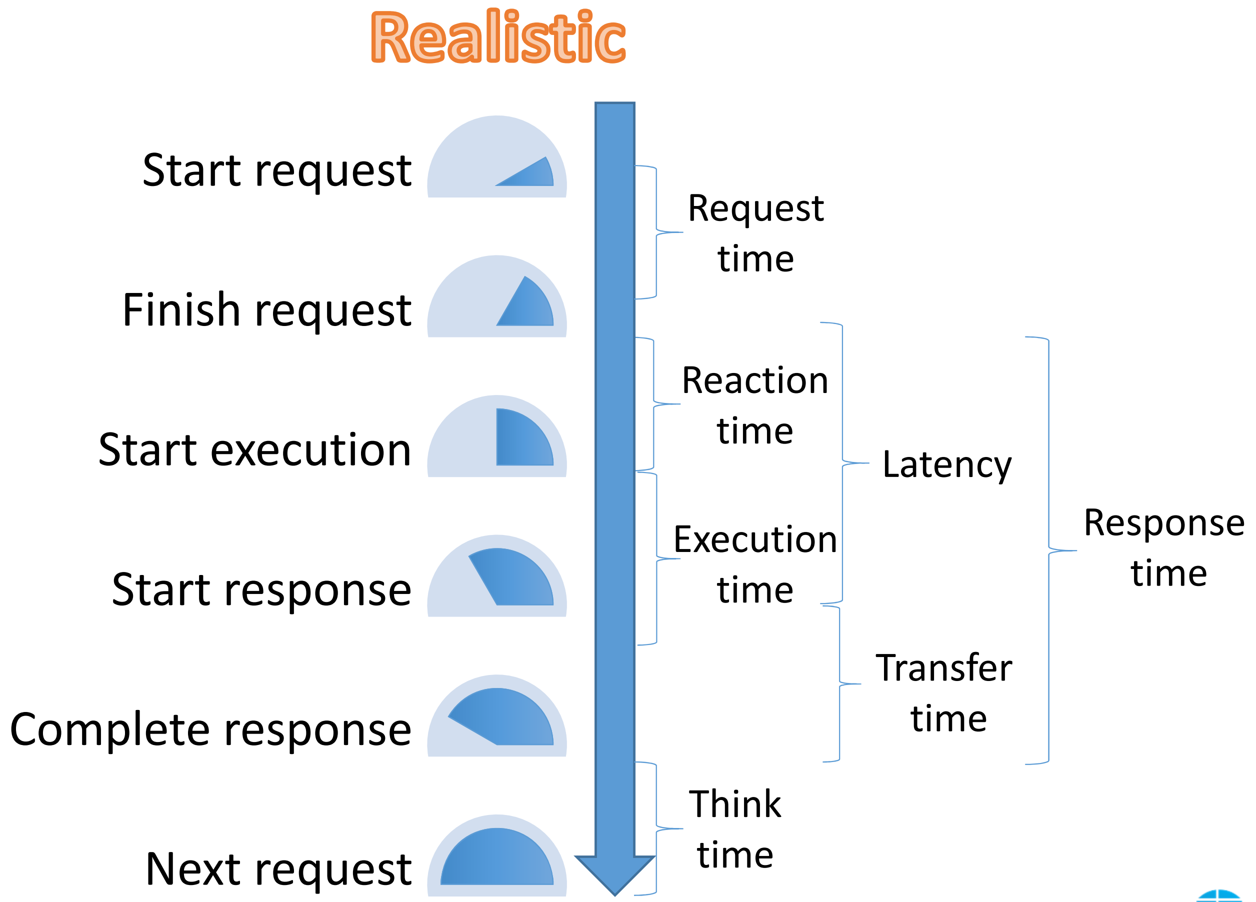
occupancy - time when first component involved in query is done

Measures of parallelism

Ahmdal's law and the Universal Scalability Law

Response time





Parallel processing

- Goal
  - Reduce response time
  - Increase throughput (adding resources)
    - Employ parallel hardware effectively   Don't waste resources
- Means
  - Process pieces of input in different processors
    a) Divide the dataset into (disjoint) subsets
    b) Adapt serial algorithms to multi-processor environments
- Consequences
  - Use more resources

Measuring scalability

- Scalability is normally measured in terms of
  - Speed-up: measuring performance on 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

Speed up - Given that the database size is constant, add N sites to reduce time from T to T/N

Scale up - Problem side is altered
Solve problem which is N time bigger in same time T
For this, we have N different sites

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

measures the theoretical speed-up that a system can achieve by introducing parallelism

- $S(p, N)$ - theoretical speed-up, where p is the fraction of parallelizable code using N processors (hardware)

p - percentage of code that can be parallelized
N - number of machines

Amdahl’s law

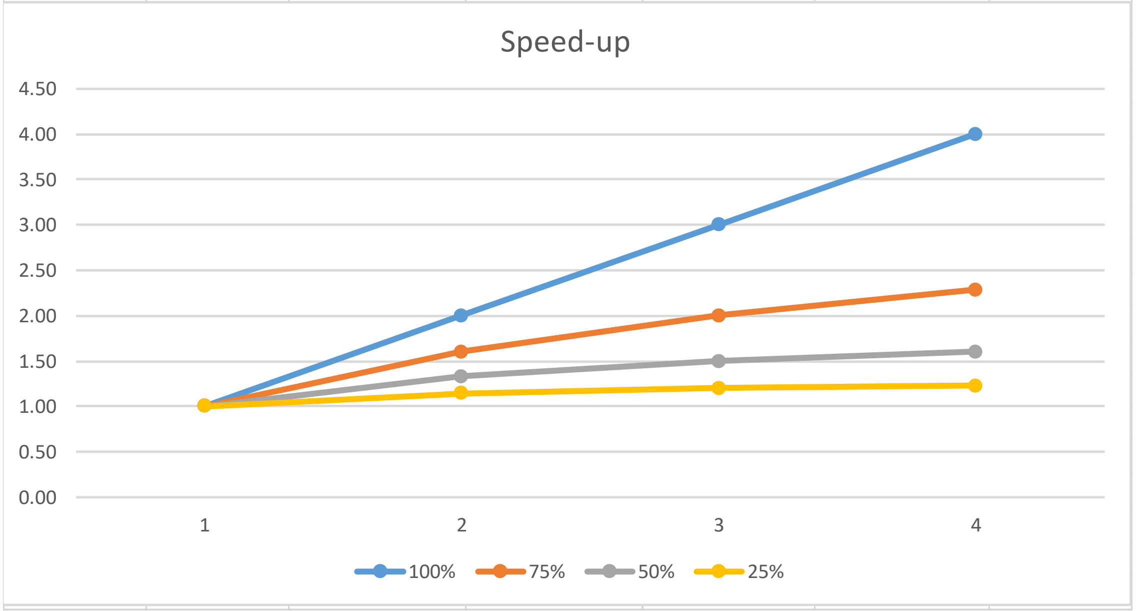
$$S(p,N) = \frac{N}{N - p(N-1)}$$

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

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

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

N\p	100%	75%	50%	25%
1	1.00	1.00	1.00	1.00
2	2.00	1.60	1.33	1.14
3	3.00	2.00	1.50	1.20
4	4.00	2.29	1.60	1.23



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

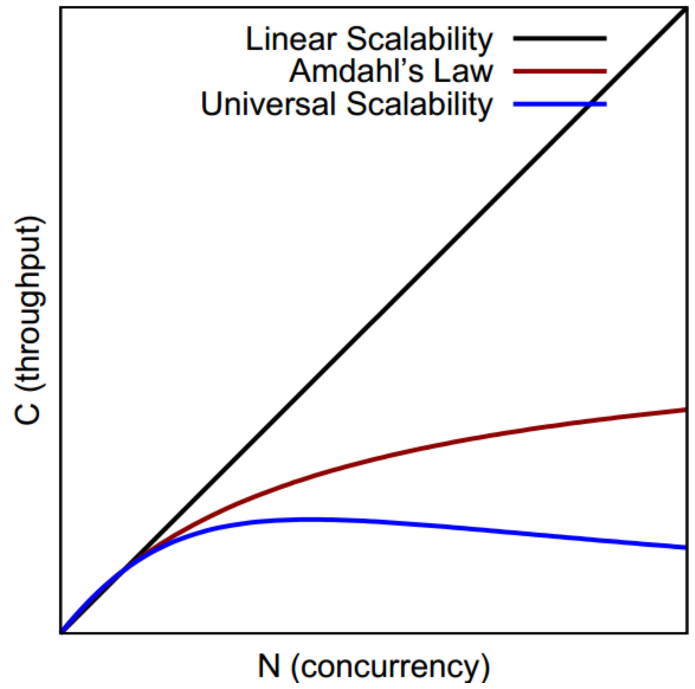
- Mathematical definition of scale-up (both for SW or HW scalability)
  - Shows that linear scalability is hardly achievable
- USL is defined as follows

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

- C: System's capacity (i.e., throughput) improvement (e.g., increment of #queries per second)
- N: System's size (SW - number of concurrent threads or 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 (a.k.a. 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

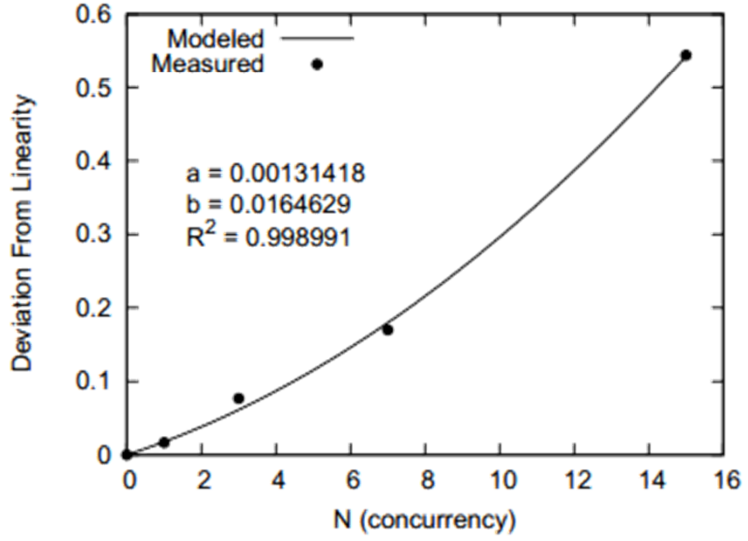
1. Empirical analysis: Measure C_i (throughput) for different values of N (concurrency)
2. Perform least-squares regression against gathered data
    1. $x = N - 1$ (interpreted as concurrency)
    2. $y = \frac{N}{C(N)} - 1$ (interpreted as deviation from linearity), where $C(N) = \frac{C_N}{C_1}$
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

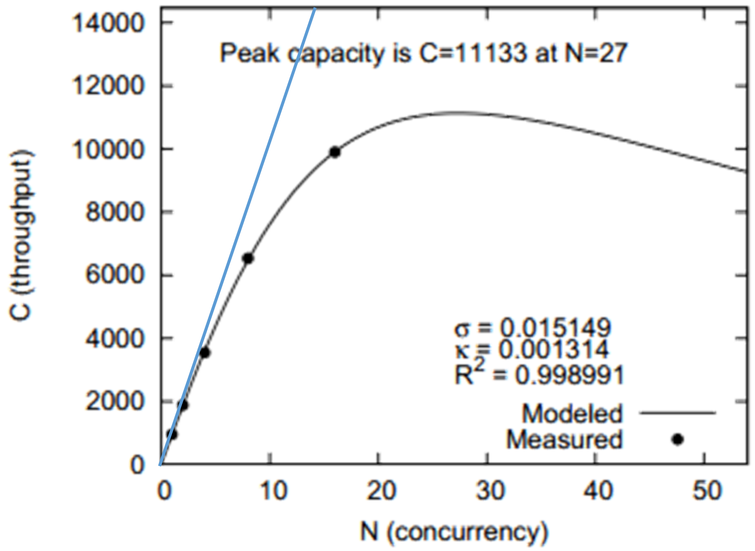
USL application method: Example

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

		x	y
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 Transaction Management
  - CAP theorem
  - Eventual consistency
- Distributed Query Processing
  - Kinds of Parallelism
  - Cost estimation
- 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