Databases for Big Data

# NoSQL data stores and techniques

## Explain the main reasons for why NoSQL data stores appeared.

Ett stort problem är att stora företag med mycket trafik och data började få problem med att lagra den här datan. Vanligt vis kan man bara slänga mer och större datorer och lagring (aka servrar) på problemet. Det är dock väldigt dyrt med stora och starka servrar och man kan bara servrar bättre till en viss rimlig gräns innan de blir för stora/dyra. Så istället började företag köra på mindre, billigare hårdvara, jättemånga normala datorer som är ihopkopplade till ett kluster av datorer.. Detta var dock ett problem för SQL som inte är designat att köras på många kluster utan köras som en server.

RDBMS är inte en one-size fits all, det finns många typer av data i många storlekar och det är inte nödvändigt att en relationsdatabas passar bra till det.

NoSQL blev då en lösning då det var open source och inte krävde starka datorer eller mycket beräkningskraft och kan både pris som lätt kunde ersättas.

Endel företag bytte/byter också pågrund av de strikta och rigida reglerna som relationsdatabaser ställer för att behålla ACID-egenskaperna. NoSQL har lite mindre rigida regler men med problemet att det är inte är konsistent men tack vare det så görs operationer och beräkningar betydligt mycket snabbare.

## List and describe the main characteristics of NoSQL data stores.

Runs on commodity hardware

Compromises on Consistensy in data

The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases

## Explain the difference between ACID and BASE properties.

**ACID – Atomic – Consistent – Isolation – Durability**

Atomic – All operations are executed or none of them are. En operation genomförs, eller så genomförs den inte.

Consisten – All fields of data must be complient with the set of rules given to the attribute (column). The database goes from a valid state to a valid state. Alla operationer ska ske enligt uppförda regler, t.ex. du får inte skriva ett text-element till ett fält som är specificerat som numeriskt.

Isolation – A transaction is executed independent of all other transactions submitted to the server.En transaktion genomförs oberoende av andra transaktioner och ska genomföras som om det var den enda transaktion som gjordes vid tillfället.

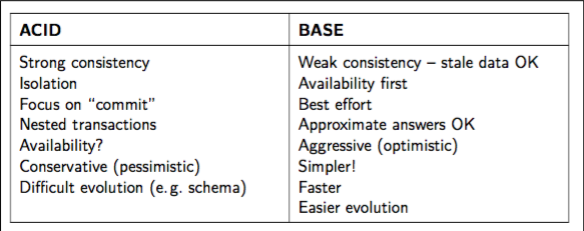
Durability – Once a transaction in carried out it will remain. Genomförda transaktioner är permanenta.

**BASE- Basicly Avaliable – Soft State – Eventual Consistent**

Basicly Avaliable – Nästan alltid tillgängligt, ibland kraschar det. (Det värsta som kan hända är att man får lägga in varan I korgen igen.)

Soft State – i En noSql server är i ständig förändring, och är icke deterministisk, vilket betyder att det finns en liten del slump i systemet.

Eventually Consisten – Någon gång kommer data vara konsistent, mao följa alla regler, men det behöver inte nödvändigtvis vara constant.



## Discuss the trade-off between consistency and availability in a distribute data store setting.

## Discuss different consistency models and why they are needed.

## Explain how consistency between replicas is achieved in a distributed data store.

The different nodes that store the replicas communicate with each other to make sure that all copies are up to date.

## Explain the CAP theorem.

**CAP – Consistency – Availibillity – Partition Tolerance**

Teoremet refererar till att man kan bara välja en av dessa tre saker att ha i sin databas.

Consistency – Om och hur ett system är konsistent efter en operation

I ett distribuerat system anses det vanligen vara “Consistent” om alla ser en uppdatering gjord av en person som har “writat” I den delade datan. (Finns fler alternativ till det här perspektivet)

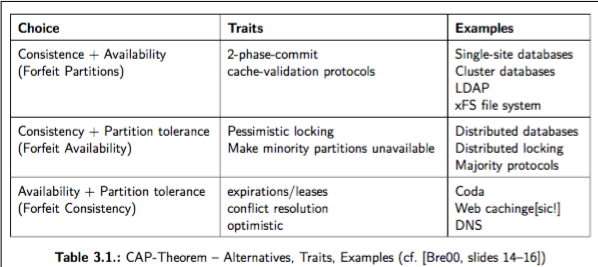
Availability – Systemet är implementerat på sådant sätt att även om några av noderna i klustret är nere (pga krasch eller updatering eller w/e) så är fortfarande systemet tillgängligt.

Partition Tolerance – Med detta menas att systemet kan fortsätta användas även fast det blidats öar av noder då andra noder ligger nere och gör att samtliga noder inte kan ansluta till varandra. En annan tolkning av det här är hur enkelt man kan ta bort och lägga till noder när systemet ligger uppe.

Om man väljer Consistency och Partition Tolerance så är ACID bäst.

Om man väljer Availability och Partition Tolerance så är BASE bäst

Avaibility och Consistency är det tredje och uppfyller INTE BASE eller ACID



## Explain the differences between vertical and horizontal scalability.

Horizontal scalability är när man sk. In/out scalability och är när man lägger till/tar bort noder (datorer) till sitt system .

Vertical scalability(up/ner) är när man lägger till / tar bort minne/CPU eller RAM från enskilda noder. Datorns hårdvara med andra ord.

Att lägga fler till datorer (Horizontal) gör det mer komplext att dela upp antalet datorer medans problemet med Vertical är att det är dyrt med riktigt bra hårdvara.

## Explain how consistent hashing works and what are the problems it addresses.

Problemet:

Om vi har ett kluster med mycket data på och bestämmer oss för att lägga till fler noder så kommer vi behöva distribuera om all data och ränka ut nya hash-funktioner för allt. Det är väldigt beräkningstungt och utöver det kommer vi (troligen) få flytta om all data i noderna. Även detta tidskrävande.

Lösningen : Consisten Hashing

I Consisten Hashing är hash funktionerna ordnade i en ring runt noder som lagrar data och hashfunktionen distrubuerar. Om ett nytt key-value pair tillkommer skickas den i klockvisordning till nästkommande nod. Om det skulle vara så att det tillkommer en ny nod mellan den tidigare data noden och hashfunktionen kommer det tidigare nämnda key-valupaiert skickas tillbaka (mot klockvis) för lagring där.

Om en nod skulle krascha så ger de två närliggande noderna datan till den nod som tappat datan.

## Explain how vector clocks work and what are the problems they address.

Vektor klockor är lösningen till att se till att det inte blir några kausala problem och kan därmed lösa eventuella konsistensproblem.

Från början börjar varje nod med en tupel med N stycken nollor där varje nolla representerar en nod (inkluderad sig själv). Varje gång noden gör någon form av förändring i sin data så lägger den till ett värde i sin del i tupeln. Det här kan vara ett klockslag eller bara en etta eller liknande. Nästa gång den här noden kommunicerar med en annan nod så uppdaterar de vad de vet om de andra noderna. (sk Gossip)

## List and describe dimensions that can be used to classify NoSQL data stores.

How data is stored

How the data is located, i.e. hashin-function, b-trees osv.

## List and describe the main characteristics and applications of NoSQL data stores according to their data models.

# HDFS

## Explain what HDFS is and for what types of applications it is (not) good for.

Hadoop Distributed File System – it is a part of Apache Hadoop and is a file system with data distributed over a cluster with many nodes. It is designed to handle large amounts of data and keep replicas of each set of data to safely store the data.

In this way, the map and reduce functions can be executed in parallel on smaller subsets of your larger data sets. and this provides the scalability that is needed for big data processing. It is designed to have crashes and that is not a problem since each part of the data is stored multiple times on different nodes.

Känns lite tomt här…

## Explain the organization of HDFS.

NameNode – I Name Noden lagras all information om vart alla filer och directorys finns. De är respresenterade som innodes som noterar attribut som premissions,modification och acccess tid, namespace och diskspace quotas. NameNoden håller koll på den fysiska platsen för datan vilket är dess huvud-uppgift då det är den som ger vägen till MapReduce-jobb.

NameSpace finns i RAM-minnet för HDFSen och håller även en journal över alla händelser. Den här journalen kan även den sparas på andra servrar som backup för ökad hållbarhet om systemet skulle krascha.

DataNodes – Här lagras data, varje block replica är representerad av två filer, en med data och en med metadata om vilken ”genereation” data är av.

HDFS Client – Den som ger jobb

Image and Journal – Namespace image är en fil som har filsystemets metadata som beskriver organisationen av filstrukturen och data. En namespace image som är skriven till disk är kallat en checkpoint.

Journalen är en ”write-ahead comit log” för ändringar som måste vara persistent. Varje transaktion som är initierad av HDFS klienten skrivs ner i journalen.

Om journal eller checkpointen saknas eller är korrupt så kommer namespace informationen försvinna helt. Det är därför viktigt att spara dessa på olika ställe och ha en back up på en annan fristående server.

Checkpoint node – checkpoint noden kombinerar existerande checkpoint med journalen periodvis och på såvis skapar en ny checkpoint som blir rapporterad till NameNoden. När det här görs så skapas en ny tom journal.

BackupNode – Funkar som en CheckpointNode men håller även en uppdaterad bild av filsystemets namespace som alltid är synkad med NameNoden.

Snapshot – en bild över allt som det såg ut precis vid tillfället som kan sparas och användas för roll-back till denna om en misslyckad uppdatering eller liknande skulle göras.

## Explain the process of reading and writing to HDFS.

When writing data, the client requests the NameNode to nominate a suite of three DataNodes to host the block replicas. The client then writes data to the DataNodes in a pipeline fashion. The current design has a single NameNode for each cluster.

When reading data the namenode is contacdet by the Client that returns the locale of the data the client wants to access.

## Explain how high availability is achieved in HDFS.

By providing the option of running two redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby. This allows a fast failover to a new NameNode in the case that a machine crashes, or a graceful administrator-initiated failover for the purpose of planned maintenance.

Känns lite tomt här

# Dynamo

**Amazon’s Dynamo is a Key-/Value-Store**

## Explain the data model and list main applications of Dynamo.

Key-/value-stores have a simple data model in common: a map/dictionary, allowing clients to put and request values per key. Besides the data-model and the API, modern key-value stores favor high scalability over consistency and therefore most of them also **omit** rich ad-hoc querying and analytics features (especially joins and aggregate operations are set aside). Often, the length of keys to be stored is limited to a certain number of bytes while there is less limitation on values

## Explain the Dynamo design considerations and what are the advantages of Dynamo in comparison to RDBMSs.

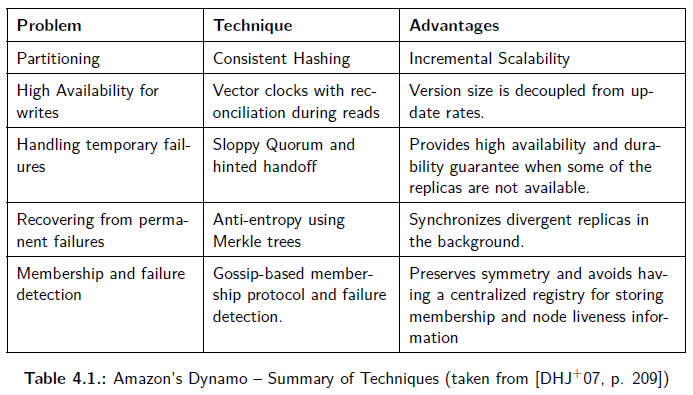
The infrastructure is designed to operate over tens of thousands of servers and network components located in datacenters all over the world. It is designed to work on commodity hardware where failures are a given not a risk.

Its goal is to maximize performance, reliability and efficiency where reliability is of utmost importance, since any form of downtime makes an financial impact on the company since the customers cant shop. To maintain this the system needs to me highly scalable.

against RDMBSs at Amazon as most services there “only store and retrieve data

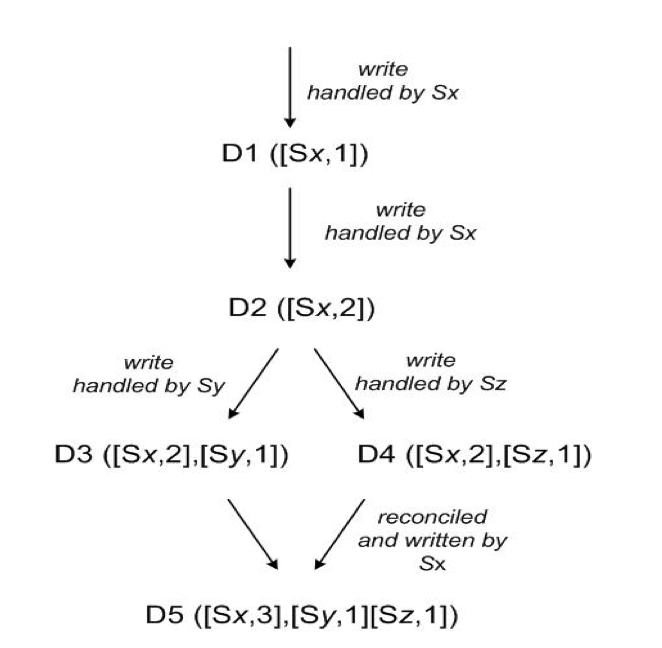
by primary key and do not require the complex querying and management functionality offered by an RDBMS”. Furthermore they consider the “available replication technologies” for RDBMSs as “limited and typically choosing consistency over availability”.

## Explain how basic NoSQL techniques are applied in Dynamo.



## Explain versioning and semantic reconciliation in Dynamo.

Designed to eventually be consistent and read-requests following each other might not return the same value since the update isn’t applied to all replicas of the data at the same time. For conflict control Dynamo uses Vector Clocks.



Let’s understand how vector clocks works: A client writes a new object. Node Sx handles this write and creates a vector clock [(Sx, 1)] for the object D1. If the client now updates the object and node Sx again handles the request, we get a new object D2 and its vector clock [(Sx, 2)]. The client updates the object again and this time node Sy handles the request leading to object D3 with the vector clock [(Sx, 2), (Sy, 1)]. When a different client tries to update the object after reading D2, the new vector clock entry for object D4 is [(Sx, 2), (Sz, 1)] where Sz is the node that handled the request. Now when a new write request is issued by the client, it sees that there are already D3 and D4 objects. If node Sx is handling the request, it performs the reconciliation process and the new data object D5 is created with the vector clock [(Sx, 3), (Sy, 1), (Sz, 1)].

From <http://cloudacademy.com/blog/data-versioning-with-dynamodb-an-inside-look-into-nosql-part-5/> also in the Amazon DynamoDB paper found online via google.

# HBase

**The Hadoop Database, används ovanpå HDFS, använder inte MapReduce mellan utan ligger direkt på.**

## Explain the difference between column-oriented and row-oriented storage.

https://www.quora.com/What-are-the-main-differences-between-the-four-types-of-NoSql-databases-KeyValue-Store-Column-Oriented-Store-Document-Oriented-Graph-Database

Row-oriented is the “regular-way” of thinking where one of the columns are a key (which must be unique) and the other ones represent values. They are extremely fast for writing, and extremely fast for reading and updating...if you have the key.  They are slow on multiple updates and if you have to query the entire store.    
You see Key-value stores used a lot as caching stores because of their fast reads. (see: Redis, Riak, memcached, Azure's tablestore, etc)

**Column stores** seem to store data in related rows, but they actually serialize the data into columns.    
With a row based database you would have:  
ID,firstname, lastname, websitename  
1:bart, loews, quora  
2:jim, finnegan, beginagain  
3:don, quixote, windmill

Column-Oriented databases are most similar to traditional row based databases since both are usually relational. Columnar databases simply store data in columns instead of rows, thus making searching on each column much faster. However inserts and updates are generally slower in column databases.   
  
In practice, columnar databases often store data in column families. In this case you would probably not store the street, city, zip in completely separate columns, but instead in a column family where they are stored together. The differences between row based and column based databases start to blur even more in this scenario.

The column store stores columns together, like this:  
1:bart,2:jim,3:don  
1:loews,2:finnegan,3:quixote  
1:quora,2:beginagain,3:windmill  
This allows for much faster querying and processing of data while storing data that's somewhat related (druid boasts billions of records per second).

## Explain the data model of HBase.

HBase uses LSM-trees (Log structured merge-trees) to access and store data.

When something is written it is written to the Memtable which is stored in the RAM-memory. This gets written down to disk from time to time in SStables. De här blir efter ett tag komprimerade i större enskilda filer, detta för HDFS jobbar snabbare och bättre med få stora filer än många små filer.

Every entery in a Table is indexed by a RowKey and for every RowKey there can be an unlimited amount of attributes (columns) stored. When a new data is written it doesn’t overwrite the data but instead saves a new version with a new time-stamp. The latest version is always the first.

Data with the same RowKey and Column family is stored together.

## Give example applications of HBase.

Facebook messages employs HBase to manage all messages. Also services like Twitter and other social media sites like Linkedin that require fast access for the user.

# Hive and Shark/SparkSQL

##  Explain the problem that Hive and Shark/SparkSQL address.

The problem was that it is a very daunting task to even write simple tasks as counting and averages for “End users” that weren’t used to code in MapReduce. Many user had to spend many hours and sometimes even days writing simple operations. This is not a productive way to work for a company that leans towards analytics and data processing.

HiveQL brings a SQL-like data format with data frames and the ability to write SQL-queries while it still enjoying the key-features from the HDFS such as speed and scalability.

##  Explain the data model of Hive and Shark/SparkSQL.

Hive structures data into the well-understood database concepts like tables, columns, rows, and partitions. It supports all the major primitive types – integers, floats, doubles and strings – as well as complex types such as maps, lists and structs.

Similar to traditional databases, Hive stores data in tables, where each table consists of a number of rows, and each row consists of a specified number of columns. Each column has an associated type. The type is either a primitive type or a complex type. Currently, the following primitive types are supported:

• Integers – bigint(8 bytes), int(4 bytes), smallint(2 bytes), tinyint(1 byte). All integer types are signed.

• Floating point numbers – float(single precision), double(double precision)

• String Hive also natively supports the following complex types:

• Associative arrays – map<key-type, value-type>

• Lists – list<element-type>

• Structs – struct<file-name: field-type, … >

##  Discuss the trade-off between schema-on-read and schema-on-write approaches.

##  Explain the difference between OLAP and OLTP.

http://datawarehouse4u.info/OLTP-vs-OLAP.html

We can divide IT systems into transactional (OLTP) and analytical (OLAP). In general we can assume that OLTP systems provide source data to data warehouses, whereas OLAP systems help to analyze it.

**OLTP (On-line Transaction Processing)** is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second. In OLTP database there is detailed and current data, and schema used to store transactional databases is the entity model (usually 3NF).   
  
- **OLAP (On-line Analytical Processing)** is characterized by relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems a response time is an effectiveness measure. OLAP applications are widely used by Data Mining techniques. In OLAP database there is aggregated, historical data, stored in multi-dimensional schemas (usually star schema).

## Explain the main differences between Hive and Shark and what are the advantages they lead to.

## Explain how fault tolerance is achieved in Shark/SparkSQL

# Parallelcomputing

## PAR-Q1: Define the following technical terms: (Be thorough and general. An example is not a definition.)

* + **Cluster (in high-performance resp. big-data computing)**

A **computer cluster** consists of a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system.

Enligt Principles of PP page 39: Clusters are parallel computers made from commodity parts. The nodes are containing one or a few processors, RAM memory, and often times, disk storage. The nodes are connected by commodity interconnect, which is avalliable in several forms including Gibabit Ethernet and other.

* + **Parallel work (of a parallel algorithm)**

Slide 41 in Lecture 5: Introduction to Parallel Computing

Parallel work of algorithm A on an input of size n = max. number of instructions performed by all procs during execution of A, where in each (parallel) time step as many processors are available as needed to execute the step in constant time.

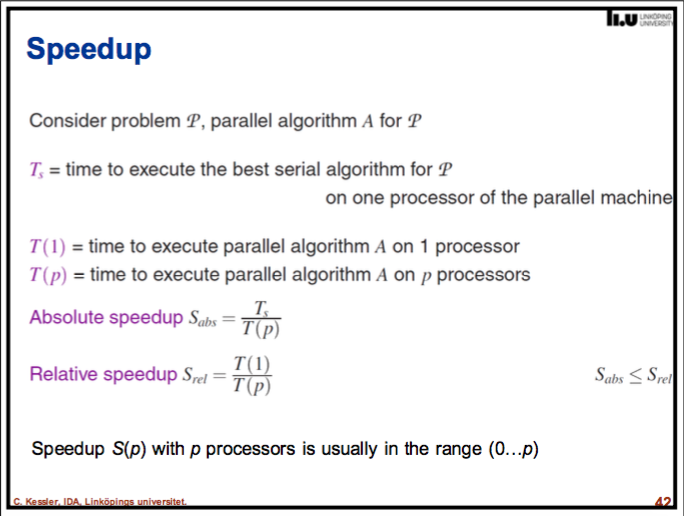
Slide 38 in Lecture 5: Introduction to Parallel Computing

The total number of performed elementary operations

* + **Parallel speed-up**

Slide 38 in Lecture 5: Introduction to Parallel Computing

The factor by how much faster we can solve a problem with p processors than with 1 processor, usually in range (0 ... p )



* + **Communication latency (for sending a message from node Pi to node Pj)**

Communication latency refers to the granularity of the operation which is the frequency of communication between the nodes.

* + **Temporal data locality**

Slide 49 in Lecture 5: Introduction to Parallel Computing

Temporal locality – re-access same data element multiple times within a short time interval

From PoPP sid 73.

Temporal data locality – memory references that are clustered in time

“By working on a contiguous block of memor, each thread explotis spatial localicty; accumulating intermediate sums in a local variable, each thread exploits temporal locality; by each thread updating the shared sum only once, the code minimizes communication, which improves locality and reduces overhead and communication.”

**On Data Locality in general (not a question)**

Data locality = property of a computation: keeping the working set small during a computation

* + - Temporal locality – re-access same data element multiple times within a short time interval
    - Spatial locality – re-access neighbored memory addresses multiple times within a short time interval
  + **Dynamic task scheduling**

Dynamic task scheduling is splitting tasks parts and distributing them whenever a worker is available. This is to improve the load (work) balance between nodes and processors.

Slide 36 in Lecture 5: Introduction to Parallel Computing

Requires a **task-based runtime system** with dynamic scheduler

Each newly created task is dispatched at runtime to an available worker processor.

Load balancing (overhead)

Central task queue where idle workers fetch next task to execute

Local task queues + Work stealing – idle workers steal a task from some other processor

## PAR-Q2: Explain the following parallel algorithmic paradigm: Parallel Divide-and-Conquer.

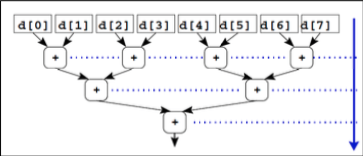
Divide and conquer is an elegant method for solving a problem: You divide the problem into smaller problems of the same kind, solve the smaller problems sepa-rately, and combine the partial results into a complete solution. The method is used recursively to split the problem into smaller and smaller problems until you reach a point where each problem is easy to solve.

## PAR-Q3: Discuss the performance effects of using large vs. small packet sizes in streaming.

## PAR-Q4: Why should servers (cluster nodes) in datacenters that are running I/O-intensive tasks (such as file/database accesses) get (many) more tasks to run than they have cores?

We subdivide the map phase into *M* pieces and the reduce phase into *R* pieces as described previously. Ideally, *M* and *R* should be much larger than the number of worker machines. Having each worker per- form many different tasks improves dynamic load balancing and also speeds up recovery when a worker fails: the many map tasks it has completed can be spread out across all the other worker machines.

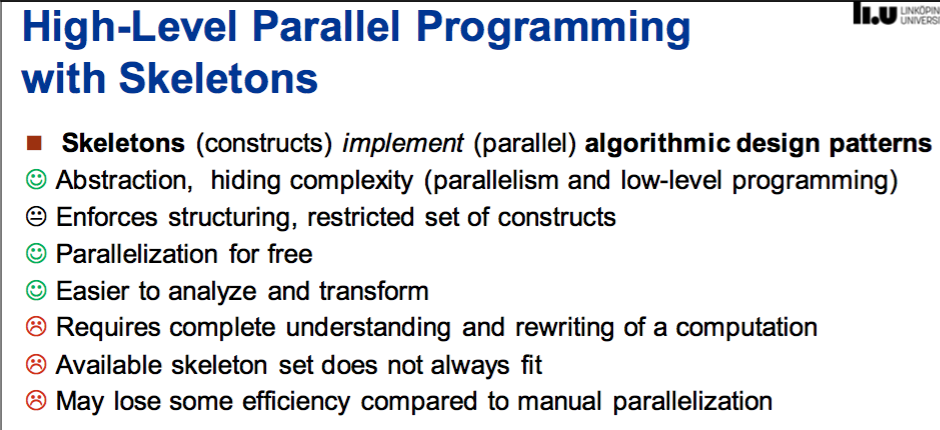
## PAR-Q5: In skeleton programming, which skeleton will you need to use for computing the maximum element in a large array? Sketch the resulting pseudocode (explain your code).

****

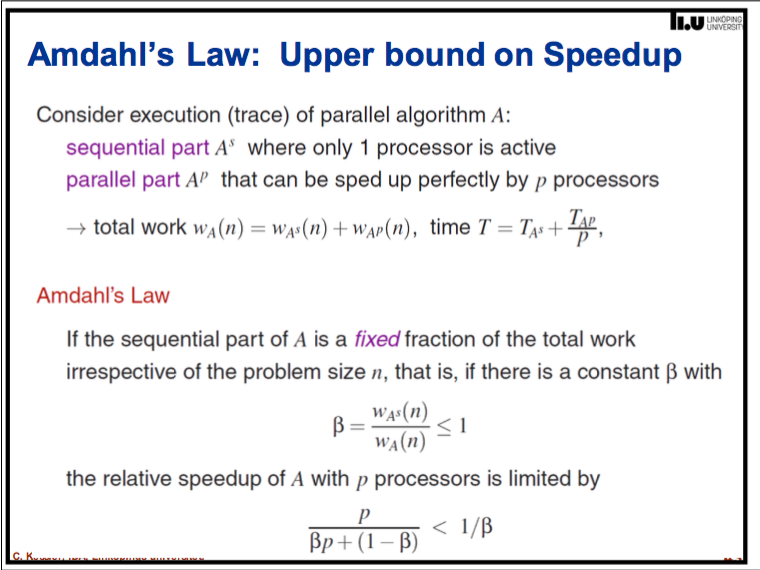
**From Wikipedia under Parallel Algorithm:** [**https://en.wikipedia.org/wiki/Prefix\_sum**](https://en.wikipedia.org/wiki/Prefix_sum)

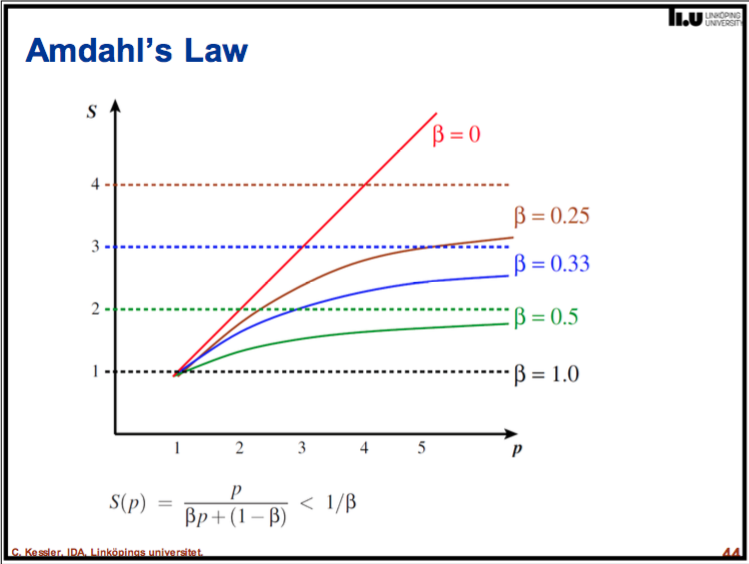
1. Compute the sums of consecutive pairs of items in which the first item of the pair has an even index: *z*0 = *x*0 + *x*1, *z*1 = *x*2 + *x*3, etc.
2. Recursively compute the prefix sum *w*0, *w*1, *w*2, ... of the sequence *z*0, *z*1, *z*2, ...
3. Express each term of the final sequence *y*0, *y*1, *y*2, ... as the sum of up to two terms of these intermediate sequences: *y*0 = *x*0, *y*1 = *z*0, *y*2 = *z*0 + *x*2, *y*3 = *w*0, etc. After the first value, each successive number *yi* is either copied from a position half as far through the *w* sequence, or is the previous value added to one value in the *x* sequence.

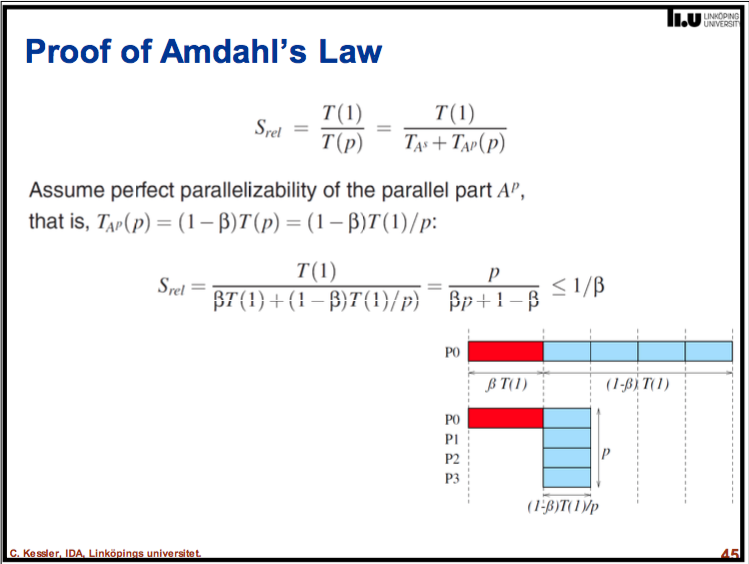
## PAR-Q6: Describe the advantages/strengths and the drawbacks/limitations of high-level parallel programming using algorithmic skeletons.

****

## PAR-Q7: Derive Amdahl's Law and give its interpretation.

****

****

****

## PAR-Q8: What is the difference between relative and absolute parallel speed-up? Which of these is expected to be higher?

Relative speed up is when comparing no. processors to one processor

Absolute speed up is comparing no processors to a perfectly serialized code running on one of the processors of a parallel machine.

Relative speed up will be expected to be faster since it is relative to doing the work on one processor.

## PAR-Q9: The PRAM (Parallel Random Access Machine) computation model has the simplest-possible parallel cost model. Which aspects of a real-world parallel computer does it represent, and which aspects does it abstract from?

**From Wikipedia** [**https://en.wikipedia.org/wiki/Parallel\_random-access\_machine**](https://en.wikipedia.org/wiki/Parallel_random-access_machine)

A parallel random-access machine (PRAM) is a shared-memory abstract machine. As its name indicates, the PRAM was intended as the parallel-computing analogy to the random-access machine (RAM). In the same way that the RAM is used by sequential-algorithm designers to model algorithmic performance (such as time complexity), the PRAM is used by parallel-algorithm designers to model parallel algorithmic performance (such as time complexity, where the number of processors assumed is typically also stated). Similar to the way in which the RAM model neglects practical issues, such as access time to cache memory versus main memory, the PRAM model neglects such issues as synchronization and communication, but provides any (problem-size-dependent) number of processors. Algorithm cost, for instance, is estimated using two parameters O(time) and O(time × processor\_number).

## PAR-Q10: Which property of streaming computations makes it possible to overlap computation with data transfer?

# MapReduce

## MR-Q1: A MapReduce computation should process 12.8 TB of data in a distributed file with block (shard) size 64MB. How many mapper tasks will be created, by default? (Hint: 1 TB (Terabyte) = 10^12 byte)

10 000 000 mb = 10^7 mb = 1 Terrabyte

12.8 TB = 12 800 000

12 800 000 / 64 mb = 200 000

The number of mapper tasks created for this task is 200 000

## MR-Q2: Discuss the design decision to offer just one MapReduce construct that covers both mapping, shuffle+sort and reducing. Wouldn't it be easier to provide one separate construct for each phase instead? What would be the performance implications of such a design operating on distributed files?

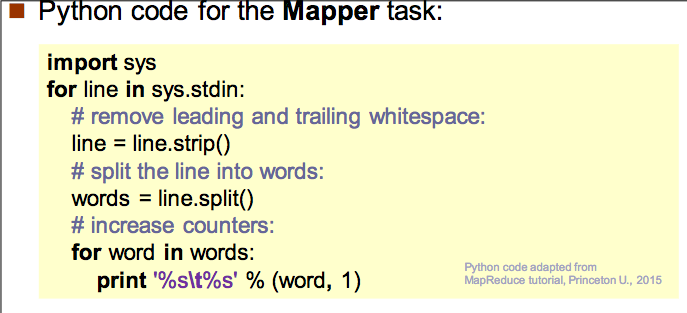
First of all shuffling is the process of transfering data from the mappers to the reducers, so I think it is obvious that it is necessary for the reducers, since otherwise, they wouldn't be able to have any input (or input from every mapper). Shuffling can start even before the map phase has finished, to save some time. That's why you can see a reduce status greater than 0% (but less than 33%) when the map status is not yet 100%.

Sorting saves time for the reducer, helping it easily distinguish when a new reduce task should start. It simply starts a new reduce task, when the next key in the sorted input data is different than the previous, to put it simply. Each reduce task takes a list of key-value pairs, but it has to call the reduce() method which takes a key-list(value) input, so it has to group values by key. It's easy to do so, if input data is pre-sorted (locally) in the map phase and simply merge-sorted in the reduce phase (since the reducers get data from many mappers).

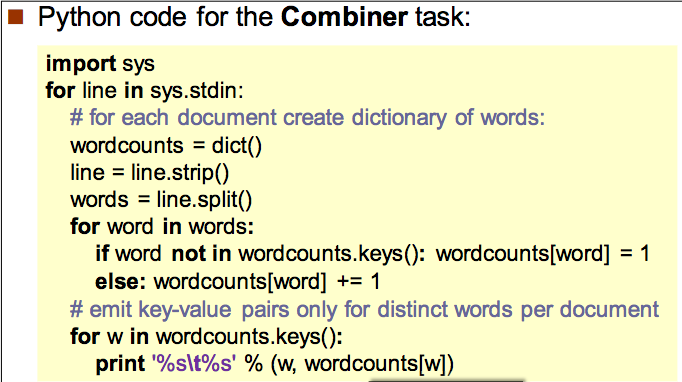
Partitioning, that you mentioned in one of the answers, is a different process. It determines in which reducer a (key, value) pair, output of the map phase, will be sent. The default Partitioner uses a hashing on the keys to distribute them to the reduce tasks, but you can override it and use your own custom Partitioner.

## MR-Q3: Reformulate the wordcount example program to use no Combiner.

Eh? What? Don’t know how to do that, here is the wordcount in mapreduce



The mapper task makes key-value pairs of (word, 1) so that the combiner can sum them up like the reduceByKey step in Spark.



Don’t realy know how to remove the combiner in this case but I guess that I would try to build the combiner in to the last for-loop of the mapper-task.

## MR-Q4: Consider the local reduction performed by a Combiner: Why should the user-defined Reduce function be associative and commutative? Give examples for reduce functions that are associative and commutative, and such that are not.

Associative – when you can rearrange the numbers but the value isn’t alterd. Example from Wikipedia (2 + 3) + 4 = 2 + (3 +4 ) = 9 or 2\*(3\*4) = (2\*3)\*4 = 24

Commutative – Independent of order i.e. a \* b = b \* a

This is because we do the computations distributed on many different computers (workers) we have the need to divide the computations. If some kind of ordering is needed every computation needs to be done sequential.

## MR-Q5: Extend the wordcount program to discard words shorter than 4 characters.

## MR-Q6: Write a wordcount program to only count all words of odd and of even length. (There are several possibilities.)

## MR-Q7: Show how to calculate a database join with MapReduce.

How about NO.

## MR-Q8: Sometimes, workers might be temporarily slowed down (e.g. repeated disk read errors) without being broken. Such workers could delay the completion of an entire MapReduce computation considerably. How could the master speed up the overall MapReduce processing if it observes that some worker is late?

Reassign the task to another worker and along with that the reducer-task is informed of this new worker

## Spark-Q1: Why can MapReduce emulate any distributed computation?

Because you can chain together multiple MapReduce steps.

## Spark-Q2: For a Spark program consisting of 2 subsequent Map computations, show how Spark execution differs from Hadoop/Mapreduce execution.

This is just my guess based

A Spark program wouldn’t repeat the whole Map-Shuffle-Reduce step as MapReduce would and instead would do Map Map Reduce

## Spark-Q3: Given is a text file containing integer numbers. Write a Spark program that adds them up.

Data = sc.textFile(“mydata”)

Myints = Data.map(lambda x: x.split(“ ”) # Assuming that the ints are separated with

#a space

Lines = Data.map(lambda x: (“sum”,int(x[0]) ))

Mysum = Lines.reduceByKey(lambda a,b: a + b )

Print Mysum.take(1)

## Spark-Q4: Write a wordcount program for Spark. (Solution proposal: see last slide in lecture 8.)

Data = sc.textFile(“mypath”)

Words = Data.map(lambda x: x.split(“ “))

Wordparts = Words.map(lambda x: (x[0],1) )

Wordsum = Wordparts.reduceByKey(lambda a,b: a + b)

Print Wordsum.take(10)

## Spark-Q5: Modify the wordcount program by only considering words with at least 4 characters.

Data = sc.textFile(”mypathdata”)

Words = Data.split(lambda x: x.split(” ”) )

Words5 = Words.filter(lambda x: len(x) > 5)

myWords = Words5.map(lamba x: (x[0],1) )

Wordsum = myWords.reduceByKey(lambda a,b: a +b)

Print Wordsum.take(10)