中国对外援助培训项目教材（讲义）
Reference Material for China-Aid Training Programs

发展中国家云环境下大数据的管理及应用研修班

Seminar on the Management and Application of Big Data in Cloud Computing for Developing Countries

Overview of Big Data Systems

主办单位：中华人民共和国商务部

Sponsor: Ministry of Commerce of the People's Republic of China

实施单位：贵州科学院

Organizer：Guizhou Academy of Sciences

2015 年 7 月 15 日 8 月 4 日

From July 15th to August 4th , 2015

中国 · 贵阳

Guiyang·China

发展中国家云环境下大数据的管理及应用研修班

Seminar on the Management and Application of Big Data in Cloud Computing

for Developing Countries Course

Overview of Big Data Systems

Li-Yan Yuan

Visiting Prof. of Guizhou Academy of Sciences
Prof. of University of Alberta，Edmonton, Canada
July, 2015

Content

Auditor: Li Dan, Assoc. Prof. of Guizhou Academy of Sciences, July,2015

中国 · 贵阳 Guiyang·China

Big Data

Li-Yan Yuan
University of Alberta
Edmonton, Canada

Introduction to Big Data

What is Big Data?

What makes data, "Big" Data?

Big Data EveryWhere!

- Lots of data is being collected and warehoused
  - –Web data,
  - –Social nextowrk
    - – facebook
    - –wichat
  - –e-commerce
  - –Mobile communication
  - –GPS
  - – …

How much data?

- Google processes 20 PB a day (2008)
- Wayback Machine has 3 PB + 100 TB/month (3/2009)
- Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
- eBay has 6.5 PB of user data + 50 TB/day (5/2009)
- CERN's Large Hydron Collider (LHC) generates 15 PB a year

The Earthscope

- the world's largest science project. Designed to track North America's geological evolution, this observatory records data over 3.8 million square miles, amassing 67 terabytes of data. It analyzes seismic slips in the San Andreas fault, sure, but also the plume of magma underneath Yellowstone and much, much more. (http://www.msnbc.msn.com/id/44363598/ns/technology_and_science-future_of_technology/#.TmetOdQ--uI)

Type of Data

- Relational Data (Tables/Transaction/Legacy Data)
- Text Data (Web)
- Semi-structured Data (XML)
- Graph Data
  - Social Network, Semantic Web (RDF), …
- Streaming Data
  - You can only scan the data once

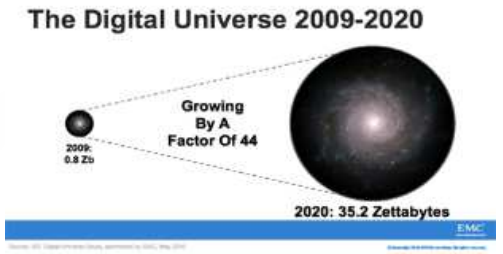
Big Data Definition

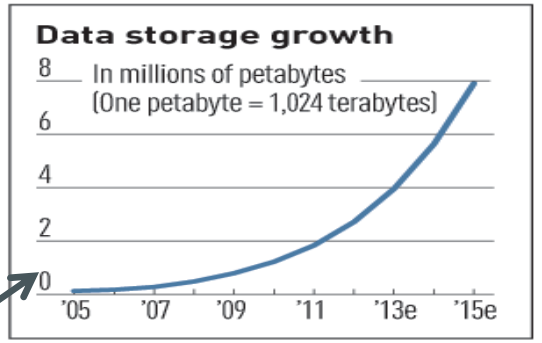
- No single standard definition…

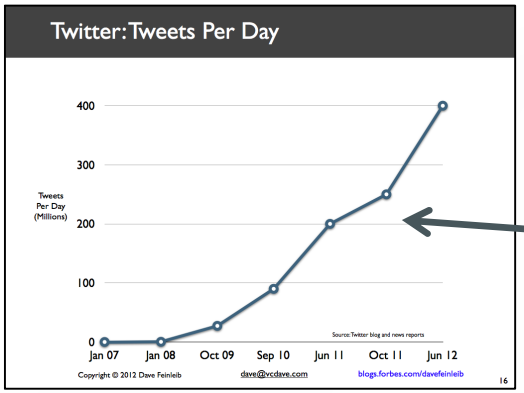
"***Big Data***" is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it…

Characteristics of Big Data:
1-Scale (Volume)

- **Data Volume**
  - 44x increase from 2009 2020
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially



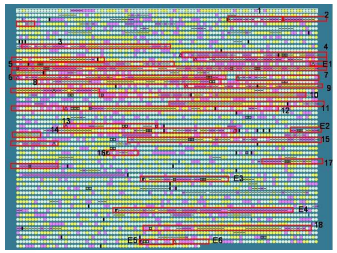


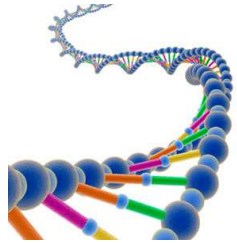


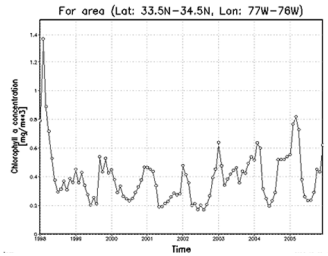
Exponential increase in collected/generated data

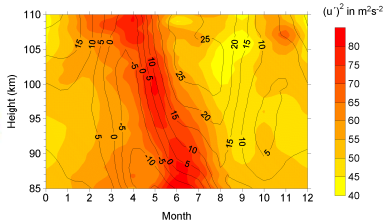
Characteristics of Big Data:
2-Complexity (Varity)

- Various formats, types, and structures
- Text, numerical, images, audio, video, sequences, time series, social media data, multi-dim arrays, etc…
- Static data vs. streaming data
- A single application can be generating/collecting many types of data









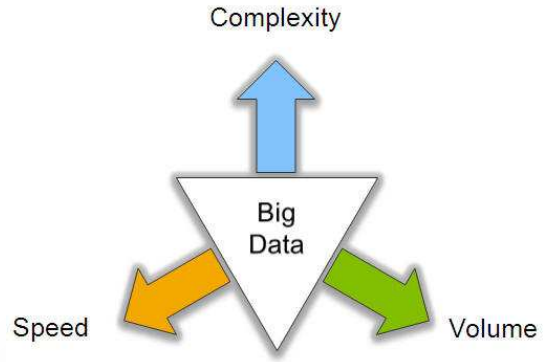
To extract knowledge➔ all these types of data need to linked together

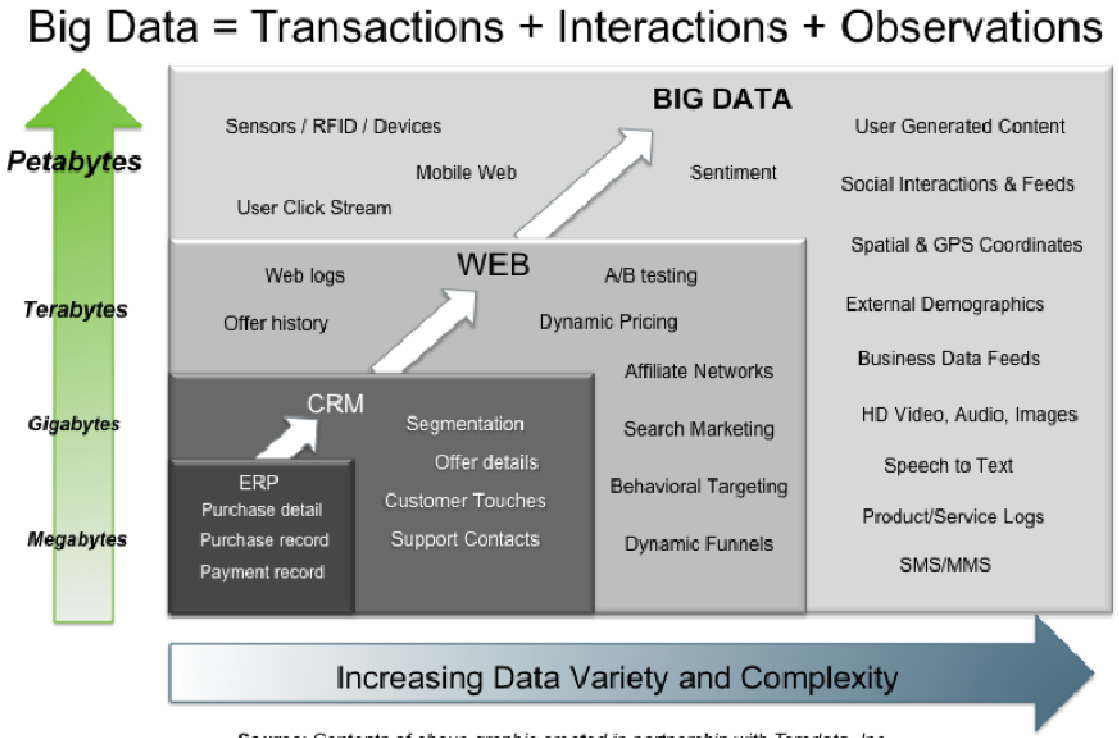
Characteristics of Big Data: 3-Speed (Velocity)

- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities

- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like ➔ send promotions right now for store next to you
  - **Healthcare monitoring:** sensors monitoring your activities and body ➔ any abnormal measurements require immediate reaction

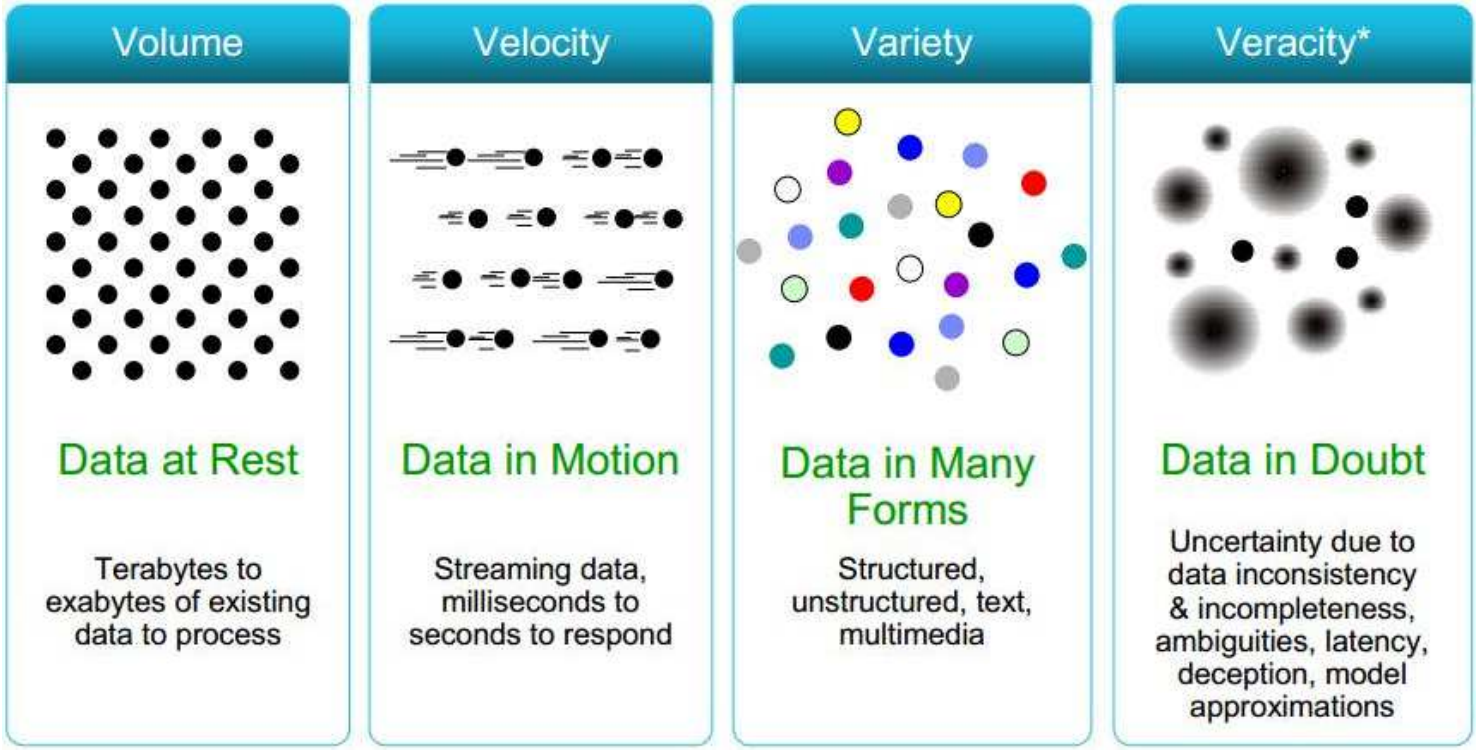
Big Data: 3V's



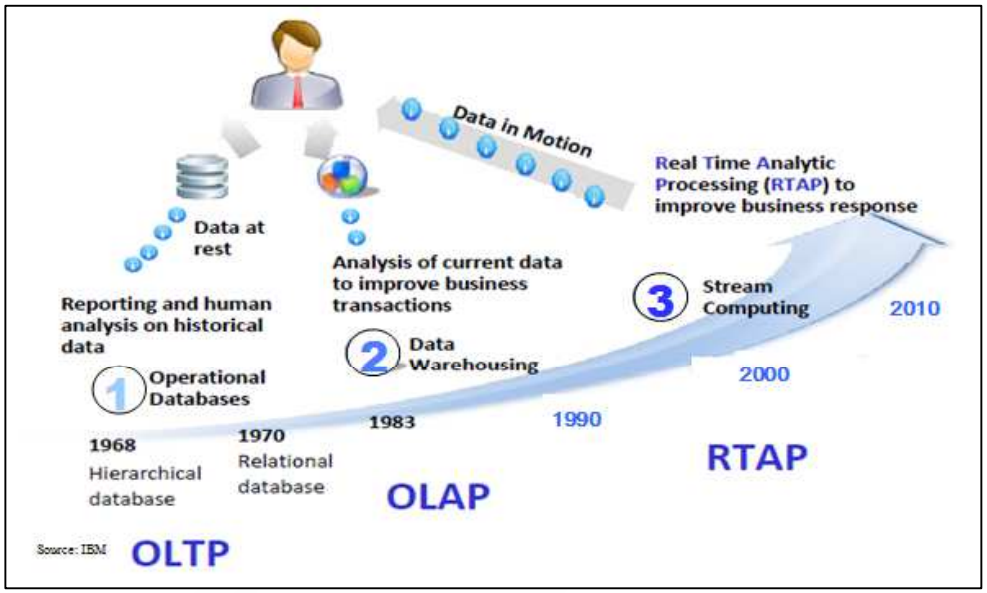


***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

Some Make it 4V's



Harnessing Big Data



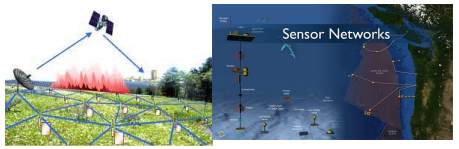
- **OLTP:** Online Transaction Processing (DBMSs)
- **OLAP:** Online Analytical Processing (Data Warehousing)
- **RTAP:** Real-Time Analytics Processing (Big Data Architecture & technology)

Who's Generating Big Data

Social media and networks
(all of us are generating data)

Scientific instruments
(collecting all sorts of data)

Mobile devices
(tracking all objects all the time)



Sensor technology and networks
(measuring all kinds of data)

- The progress and innovation is no longer hindered by the ability to collect data
- But, by the ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

The Model Has Changed…

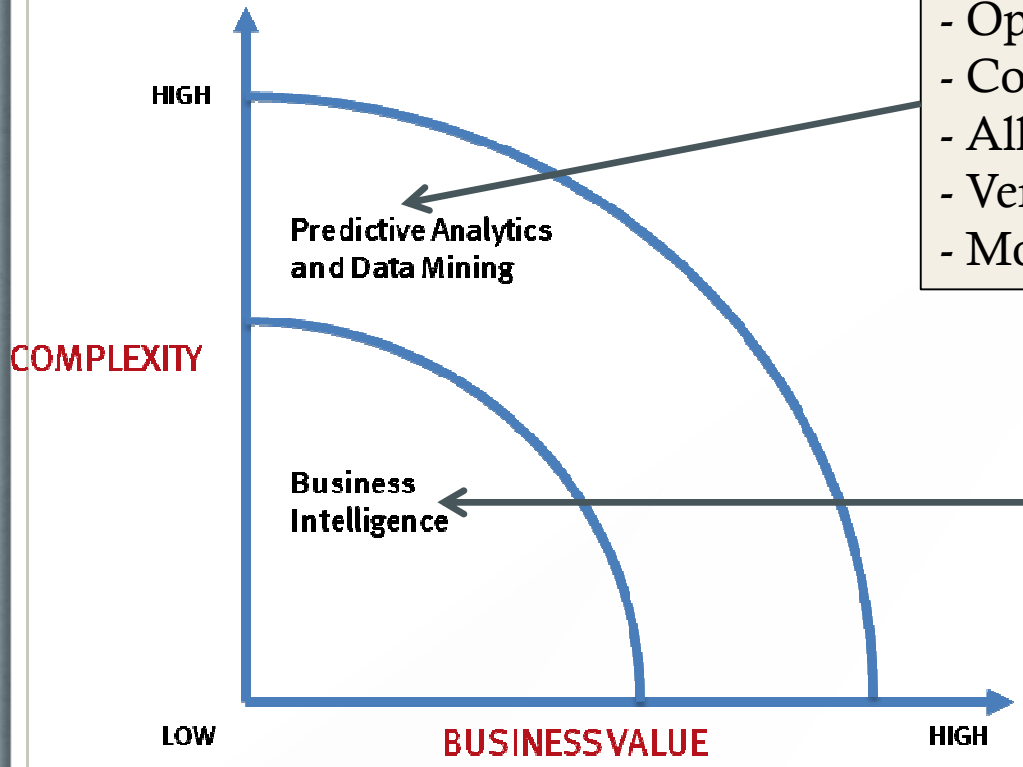
- **The Model of Generating/Consuming Data has Changed**

Old Model: Few companies are generating data, all others are consuming data


New Model: all of us are generating data, and all of us are consuming data



What's driving Big Data

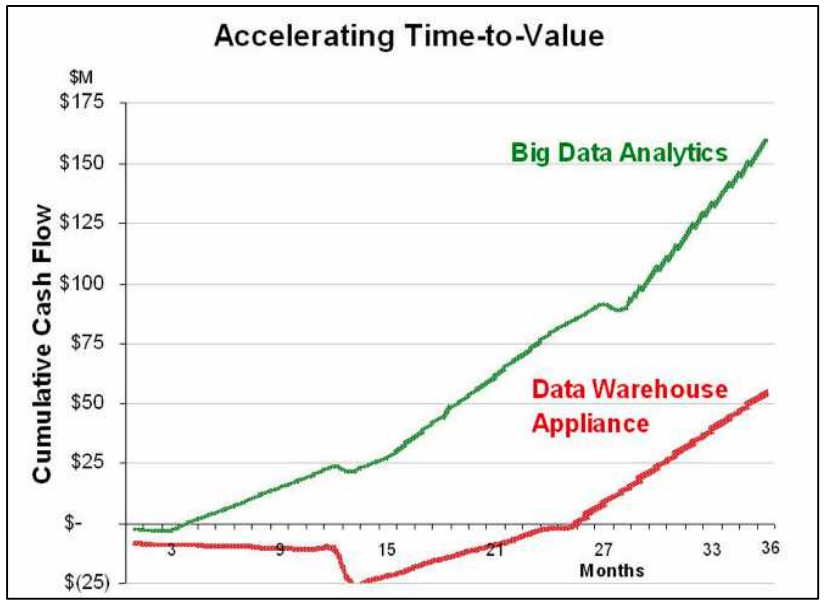


- Optimizations and predictive analytics
- Complex statistical analysis
- All types of data, and many sources
- Very large datasets
- More of a real-time

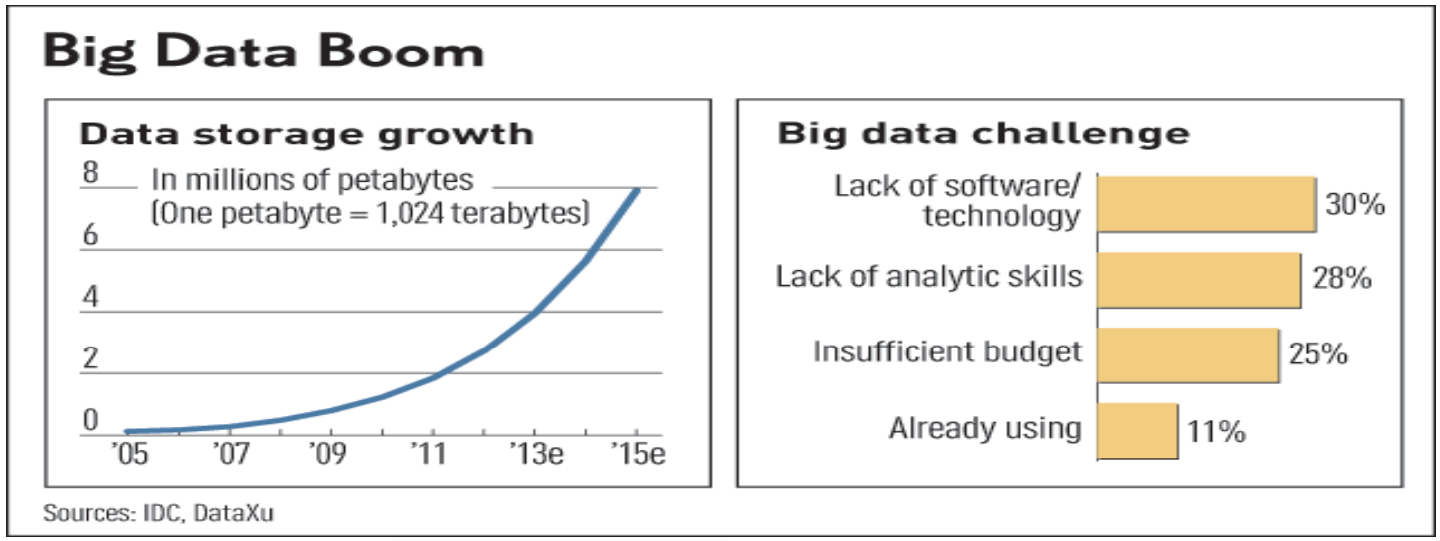
- Ad-hoc querying and reporting
- Data mining techniques
- Structured data, typical sources
- Small to mid-size datasets

Value of Big Data Analytics

- Big data is more real-time in nature than traditional DW applications
- Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data apps
- Shared nothing, massively parallel processing, scale out architectures are well-suited for big data apps



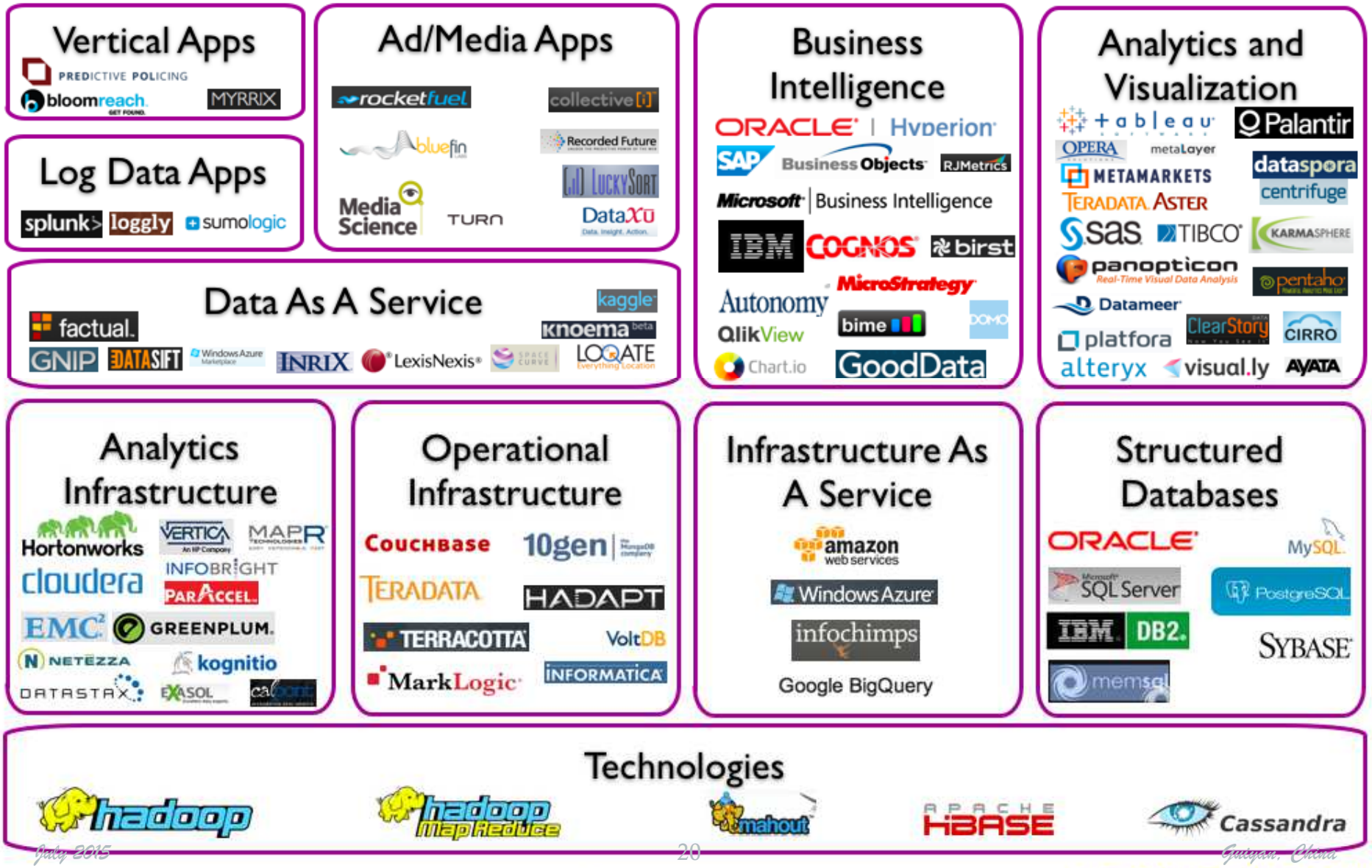
Challenges in Handling Big Data



- **The Bottleneck is in technology**
  - New architecture, algorithms, techniques are needed
- **Also in technical skills**
  - Experts in using the new technology and dealing with big data

What Technology Do We Have For Big Data ??

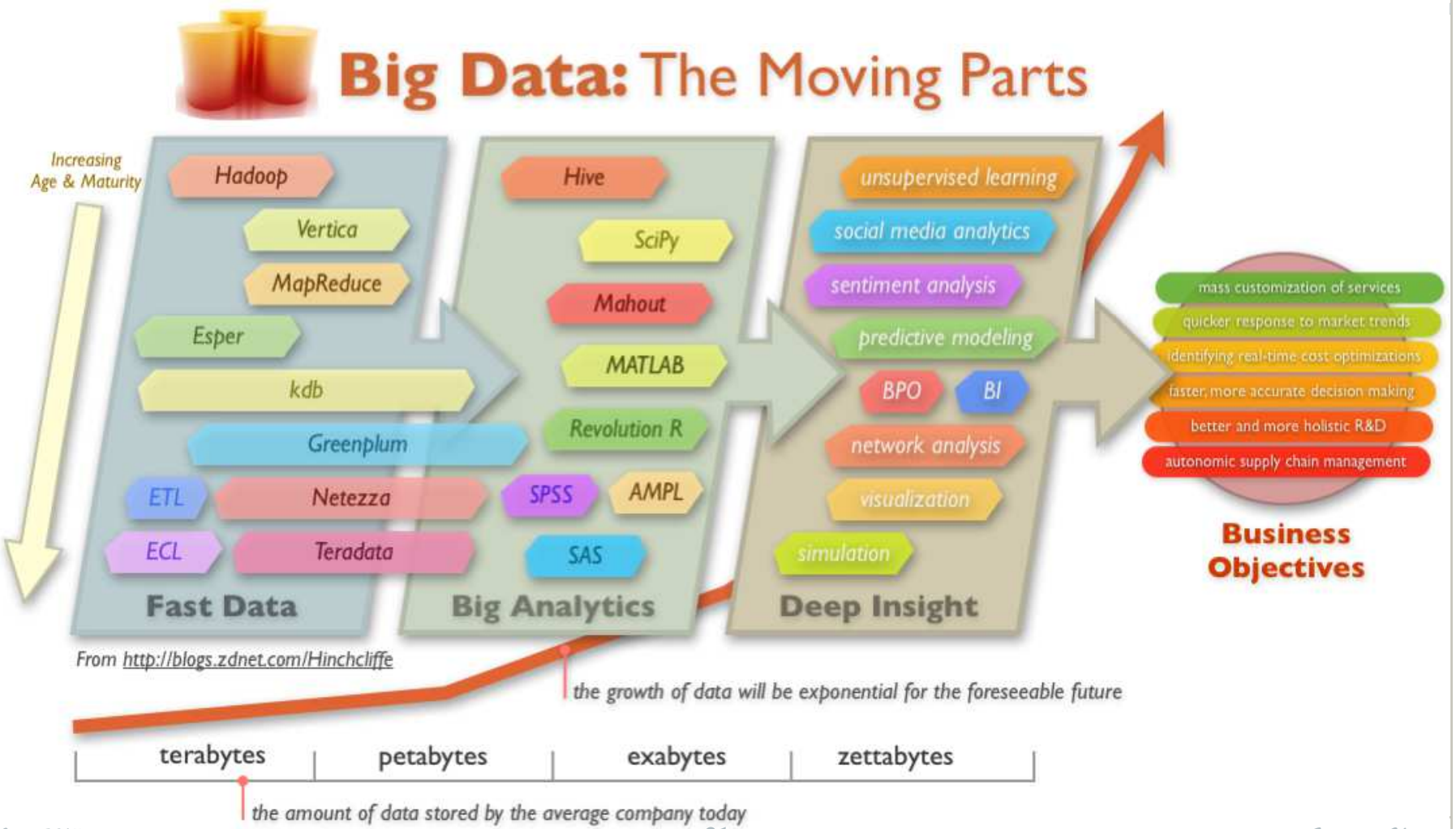
Big Data Landscape



dave@vcdave.com

blogs.forbes.com/davefeinleib

Big Data Technology



What to Discuss

- **How to handle big data**
  - **Classic SQL**
  - **No SQL**
  - **NewSQL**
- **How to analyze big data**
  - **OLTP**
  - **OLAP**
  - **Data Mining**

NoSQL Database Systems

- What is NoSQL?
- CAP Theorem
- Types of NoSQL
- Data Model
- Frameworks

What is NoSQL?

- NoSQL:
  - Not Only SQL (not No-SQL) database
  - Non relational data storage system
  - No ACID property
- A **NoSQL** database provides a mechanism for storage and retrieval of data that
  - Uses looser consistency model than traditional relational databases
  - is highly scalable; and
  - Has higher availability.

Why NoSQL?

- A large collection of structured or semi structure data
  - Social media sites such as Facebook, Twitter
  - Lack of scalability of relational databases
- High availability,
  - Not ACID
- Dynamic schemas
  - No fixed table schemas
- Frequent simpler queries
  - No joins
  - Nested queries

Scaling Up

- RDBMS were not designed to be distributed and/or grided
  - Distribution and ACID property
  - Distribution and join operations
- Multiple node relational database systems
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

How did we get here?

- Explosion of social media sites
  - (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions
  - Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages
  - a shift to dynamically-typed data with frequent schema changes
  - Ruby/Groovy
- Open-source community

Some influential events

- Some major papers/systems were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem
  - MapReduce and Hadoop

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

Brewer's CAP Theorem

A distributed system can support only two of the following characteristics:

- Consistency
- Availability
- Partition tolerance

Consistency

- all nodes see the same data at the same time
- client perceives that a set of operations has occurred all at once
- More like Atomic in ACID transaction properties

Availability

- node failures do not prevent survivors from continuing to operate
- Every operation must terminate in an intended response

Partition Tolerance

- the system continues to operate despite arbitrary message loss
- Operations will complete, even if individual components are unavailable

Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
  - Consistency is a continuum with tradeoffs
  - For NoSQL, the answer would be: maybe
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service

BASE Transactions

- **B**asically **A**vailable,
- **S**oft state
  - consistency is the developer's problem and should not be handled by the database
- **E**ventually Consistent

BASE Transactions

- Characteristics
  - Weak consistency
    - stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

What kinds of NoSQL databases

- NoSQL solutions fall into two major areas:
  - Key/Value or ‘the big hash table’.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
  - Schema-less which comes in multiple flavors
    - column-based (Cassandra, HBase)
    - document-based (CouchDB)
    - graph-based (Neo4J)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

List of NoSQL systems (www.nosql-database.org)

- Cassandre
- Haddop & Hbase
- CouchDB
- MongoDB
- StupidDB
- MapReduce
- Trift
- Couddata
- Etc.

What are we giving up?

- joins
- group by
- order by
- ACID transactions
- SQL as a sometimes frustrating but still powerful query language
- easy integration with other applications that support SQL

Cassandra

- Originally developed at Facebook
- Follows the BigTable data model
  - column-oriented
- Uses the Dynamo Eventual Consistency model
- Written in Java
- Open-sourced and exists within the Apache family
- Uses Apache Thrift as it’s API

Thrift

- Created at Facebook along with Cassandra
- Is a cross-language, service-generation framework
- Binary Protocol (like Google Protocol Buffers)
- Compiles to: C++, Java, PHP, Ruby, Erlang, Perl, ...

Searching

- Relational
  - SELECT `column` FROM `database`,`table` WHERE `id` = key;
  - SELECT product_name FROM rockets WHERE id = 123;
- Cassandra (standard)
  - keyspace.getSlice(key, “column_family”, "column")
  - keyspace.getSlice(123, **new** ColumnParent(“rockets”), *getSlicePredicate*());

Typical NoSQL API

- Basic API access:
  - get(key) -- Extract the value given a key
  - put(key, value) -- Create or update the value given its key
  - delete(key) -- Remove the key and its associated value
  - execute(key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc).

Data Model

- Within Cassandra, you will refer to data this way:
  - **Column:** smallest data element, a tuple with a name and a value

    :Rockets, '1' might return:

    {'name' => ‘Rocket-Powered Roller Skates',

    ‘toon' => ‘Ready Set Zoom',

    ‘inventoryQty' => ‘5‘,

    ‘productUrl’ => ‘rockets\1.gif’}

Data Model Continued

- **ColumnFamily**: There's a single structure used to group both the Columns and SuperColumns. Called a ColumnFamily (think table), it has two types, Standard & Super.
  - Column families must be defined at startup
- **Key**: the permanent name of the record
- **Keyspace**: the outer-most level of organization. This is usually the name of the application. For example, 'Acme' (think database name).

Cassandra and Consistency

- Talked previous about eventual consistency
- Cassandra has programmable read/writable consistency
  - One: Return from the first node that responds
  - Quorom: Query from all nodes and respond with the one that has latest timestamp once a majority of nodes responded
  - All: Query from all nodes and respond with the one that has latest timestamp once all nodes responded. An unresponsive node will fail the node

Classification of NoSQL databases

- Physical layout
- Consistency model
- Architecture

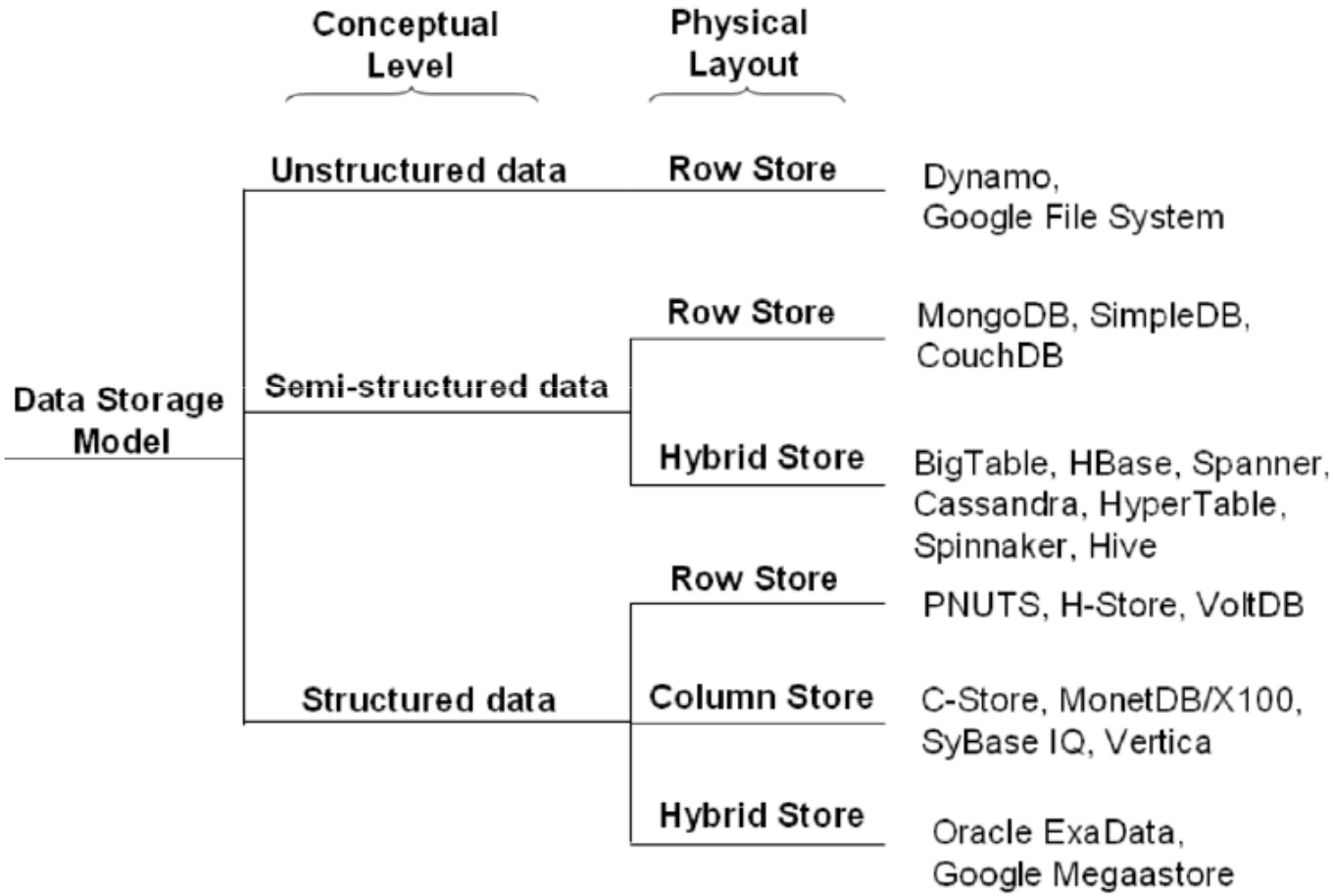
Physical Layout

- Row storage
  - Dominant in transitional DBMS
- Column storage
  - Efficient scan-level access
  - Easy compression
  - Optimized analytical workloads
  - SyBase IQ, C-Store, MonetDB-X100
- Hybrid
  - combination of row and column stores
  - Table: ne-gained hybrid
  - Block: partition attributes across

Conceptual mode

- Unstructured data
  - uninterrupted, isolated and stored in binary object
  - simplest format and maximum exibility
- Semi-structured data
  - inner structure without xed schema
  - document-oriented store, column-family store
- Structured data
  - based on the relational model
  - structured entities with strict relationships according to schema
  - constraints: entity integrity and referential integrity

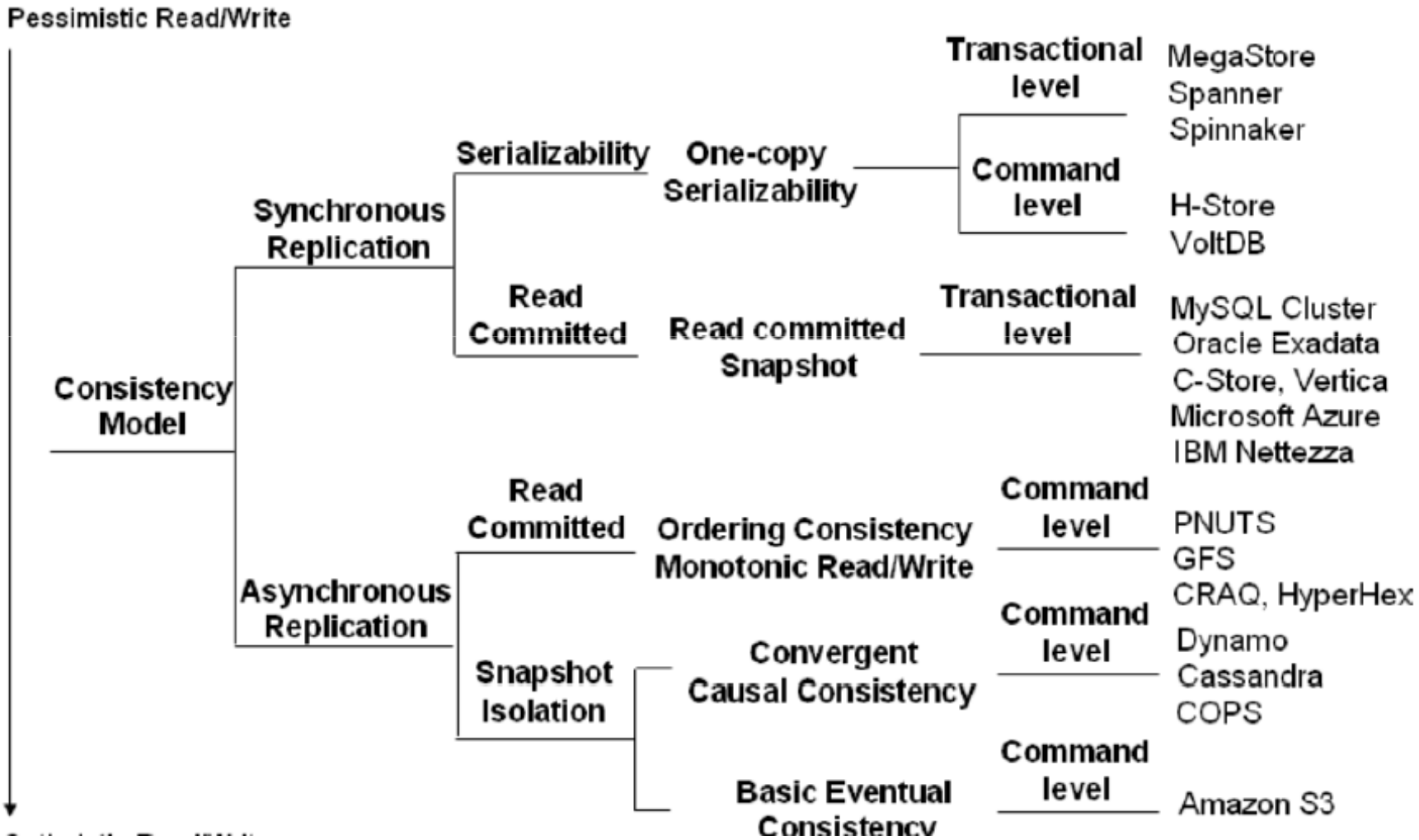
Data model Taxonomy



consistency

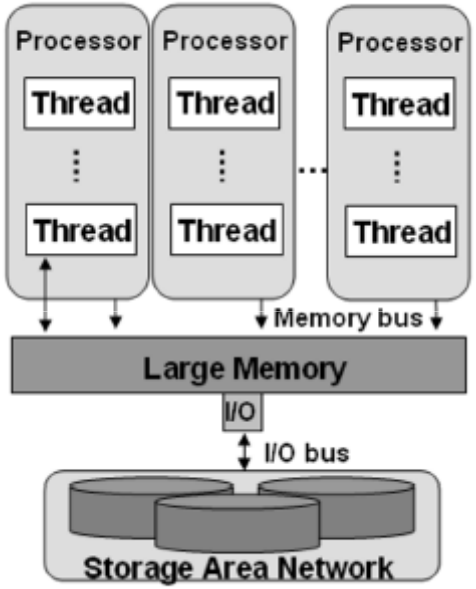
- Serializablility
- Snapshot isolation
  - Transaction sees a consistent committed snapshot of the database
  - Optimistic Read and Write
- Strict consistency
- Eventual consistence

Consistency mode taxonomy



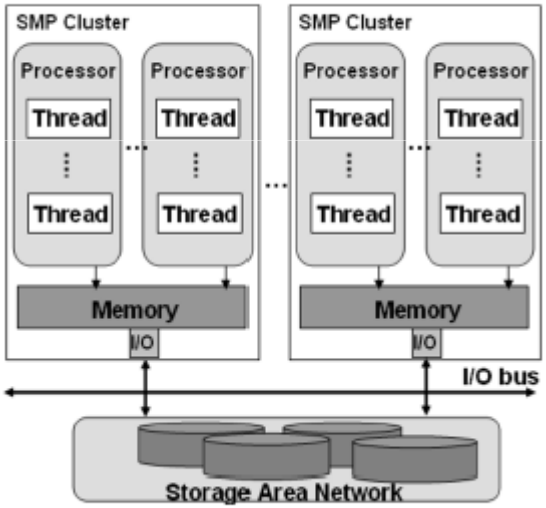
SMP on shared-memory architecture

- Processors with own cache
- Threads allocated to processors
- Coherent memory pool for sharing data
- Scale-up by adding processors
- Bounded by resource limit
  - Memory/IO bus bandwidth
  - L2 cache consistency
  - Number of cores
  - Custom-built with high price



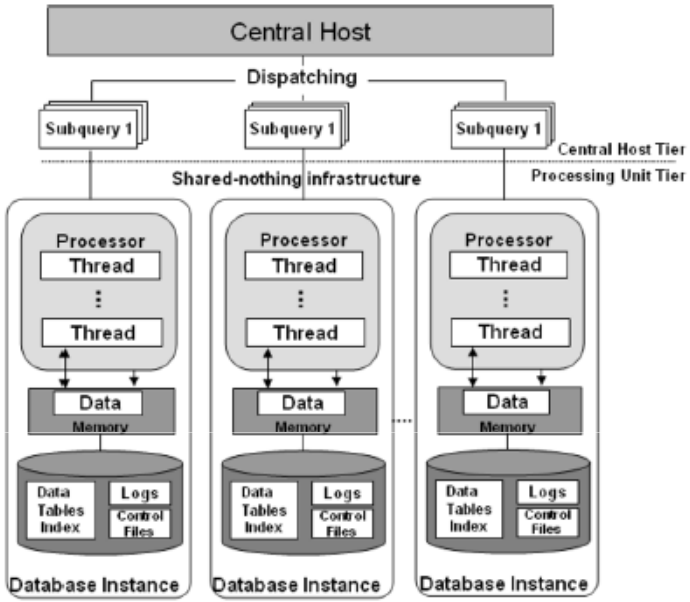
MPP on Shared-disk architecture

- Parallel SMP clusters
- Common storage by shared I/O
- Disk arrays in SAN
- Mitigate I/O bus bandwidth
  - Cache fusion protocol
    - Oracle Exadata
  - Query streaming preprocess
    - IBM Netezza

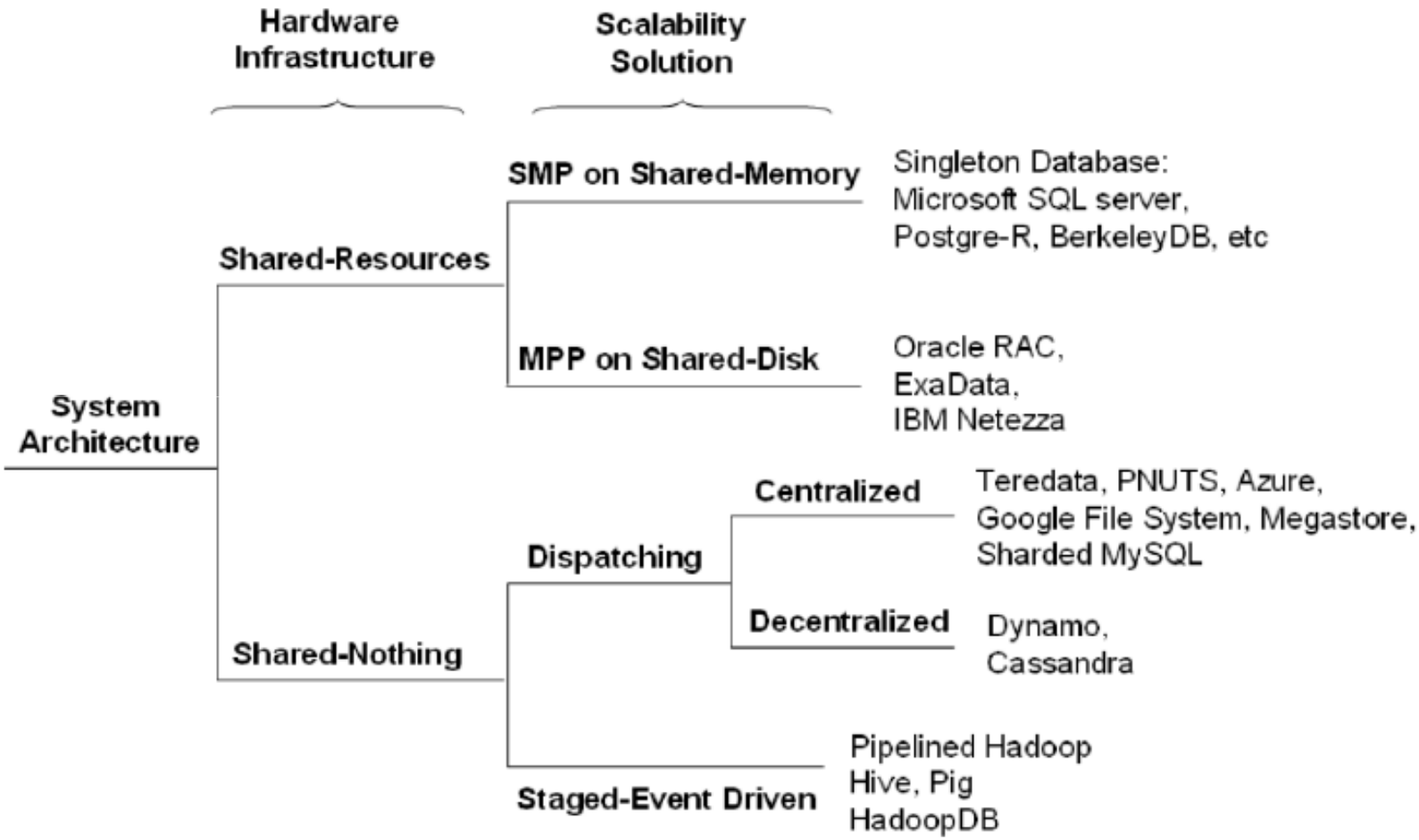


Dispatching on shared-nothing architecture

- Centeral coordinator
  - Devision
  - Dispatch
  - post-process
- Loosely coupled machines dedicated memory and disk
- Independent database instance
  - Operating on portion of the data
- Each to scale out



System architecture Taxonomy



MapReduce and Kahn Process Networks

MapReduce

- Model for processing large data sets
- Contains Map and Reduce functions
- Runs on a large cluster of machines
- A lot of MapReduce programs are executed on Google's cluster everyday

Motivation of MapReduce

- Input data is large
  - the whole web, billions of pages
- Lots of machines
  - how to use them efficiently

Programming model

- Input and output
  - each a set of key/value pairs
- Programmer specifies two functions
  - map(in-key, in-value) $\longrightarrow$ list(out-key, intermediate-value)
    - processing input key/value pair
    - process set of intermediate pairs
  - reduce(out-key, list(intermediate-value)) $\longrightarrow$ list(out-value)
    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)

Example Word counter

Input: a document with 3 pages

- Page 1: the weather is good
- Page 2: today is good
- Page 3: good weather is good

Map Output

- Worker 1 (one processor)
  - (the 1), (weather 1), (is 1), (good 1)
- Worker 2
  - (today 1), (is 1), (good 1)
- Worker 3
  - (good 1), (weather 1), (is 1), (good 1)

Reduce Input

- Worker 1
  - (the 1),
- Worker 2
  - (is 1), (is 1), (is 1)
- Worker 3
  - (weather 1), (weather 1)
- Worker 4
  - (today 1)
- Worker 5
  - (good 1), (good 1), (good 1), (good 1)

Reduce Output

- Worker 1
  - (the 1),
- Worker 2
  - (is 3)
- Worker 3
  - (weather 2)
- Worker 4
  - (today 1)
- Worker 5
  - (good 4),

Map Input

- Page 1

  the weather is good

- Page 2

  today is good

- Page 3

  good weather is good

Map Output

- Worker 1 (one processor)

  – (the 1), (weather 1), (is 1), (good 1)

- Worker 2

  – (today 1), (is 1), (good 1)

- Worker 3

  – (good 1), (weather 1), (is 1), (good 1)

Reduce Input

- Worker 1

  – (the 1),

- Worker 2

  – (is 1), (is 1), (is 1)

- Worker 3

  – (weather 1), (weather 1)

- Worker 4

  – (today 1)

- Worker 5

  – (good 1), (good 1), (good 1), (good 1)

Reduce Output

- Worker 1

  – (the 1),

- Worker 2

  – (is 3)

- Worker 3

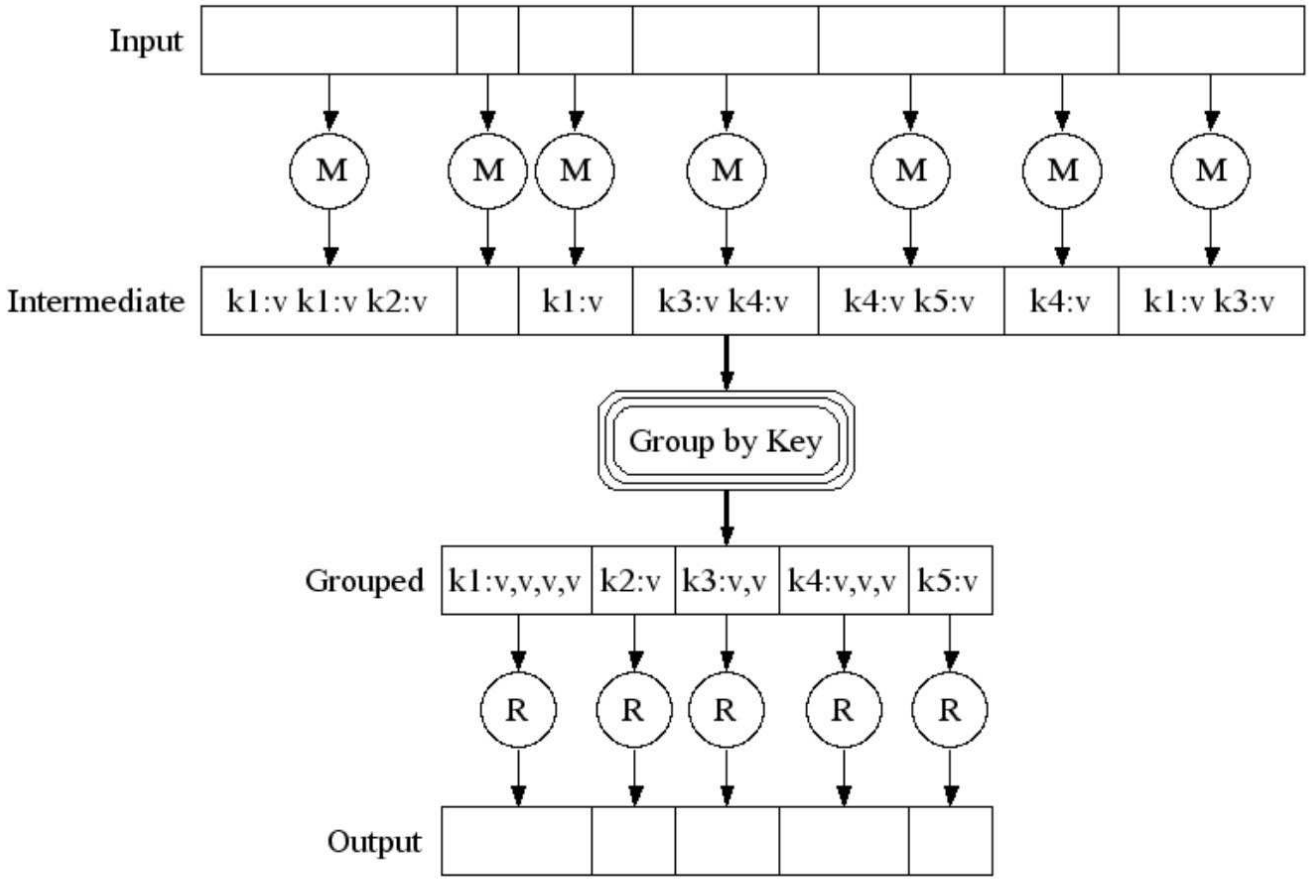
  – (weather 2)

- Worker 4

  – (today 1)

- Worker 5

  – (good 4),


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

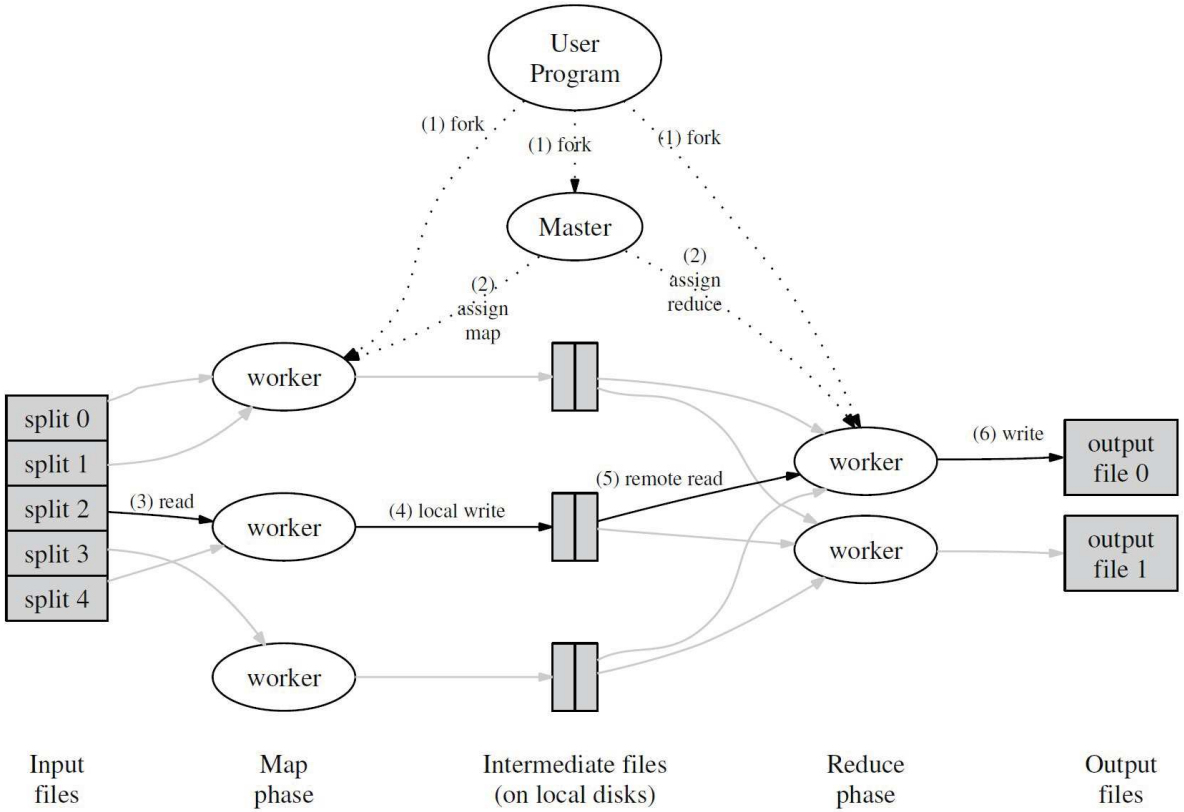
Other Examples

- Inverted Index
  - Input: list of <docId, document>
  - Output: list of <Word, list(docId, frequency)>
  - Map: parse each document and outputs a sequence of <word, docId>
  - Reduce: accept all pairs for a given work, sorts docId according its frequency, and generate the output
- Count of URL access frequency
  - Map function

    process logs of web page requests and outputs <URL, 1>
  - Reduce function

    add all values for the same URL and outputs a <URL, total count>

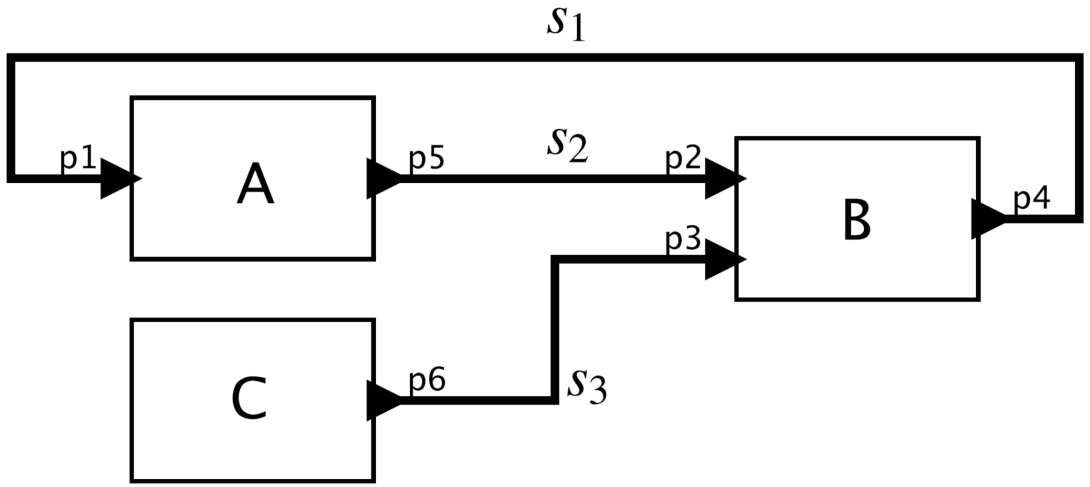
Execution Overview



Kahn Process Networks

- A set of computational components, called *actors*.
- Each representing a sequential procedure (process unit).
  - receiving messages from other actors
  - performing local operations
  - sending messages to other actors
- Messages are communicated asynchronously with unbounded buffers.
- A procedure can always send a message. It does not need to wait for the recipient to be ready to receive.
- Messages are delivered reliably and in order.
- When a procedure attempts to receive a message, that attempt blocks the procedure until a message is available.

Example 1



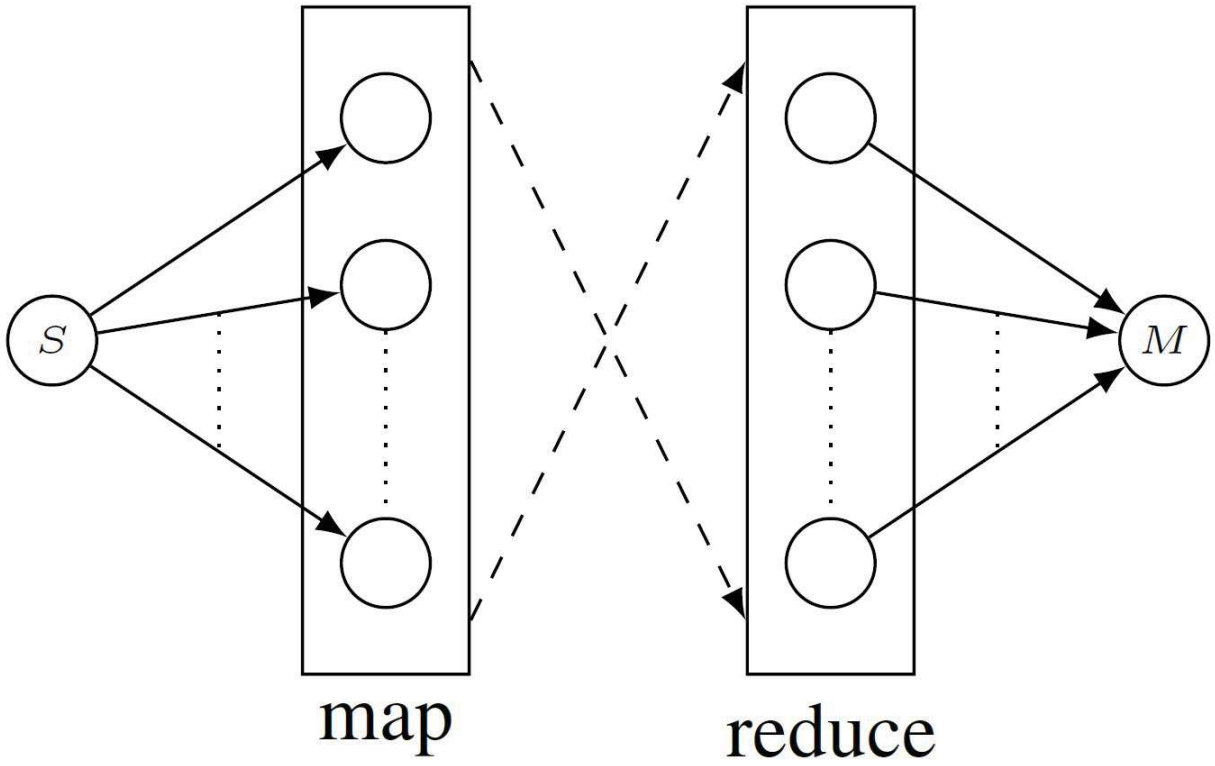
Example of a process network

- Kahn process networks
  - least fixed point semantics, deterministic
  - unbounded buffer
  - grid databases: one request/one response
- No bunbounded buffer

Desired Properties of concurrent, parallel, distributed, event-driven systems

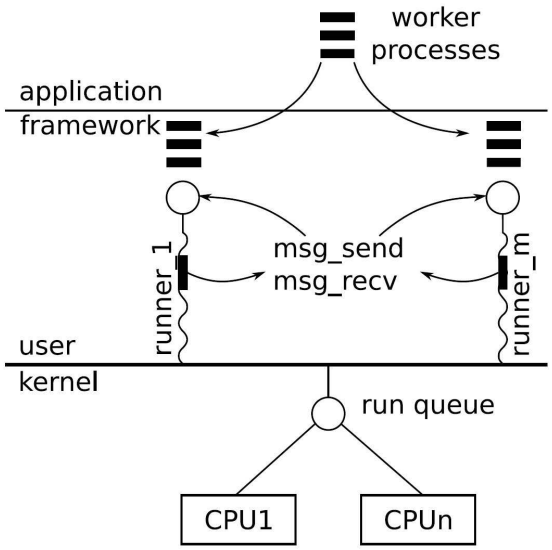
- asynchronism
- determinism
- dead lock free
- scalability

MapReduce vs Kahn Process Network



Implementation of Kahn process networks

- KPN scheduling



- Message transport
- Deadlock detection and resolution

Experimental

Word count Count the number of occurrences of each word in a given text document and outputs them sorted in the order of decreasing frequency.
Experimental

- File size: 10, 50, and 100 (MBs)
- Machine: a single server with 8 CPUs

Three solutions

- Phoenix

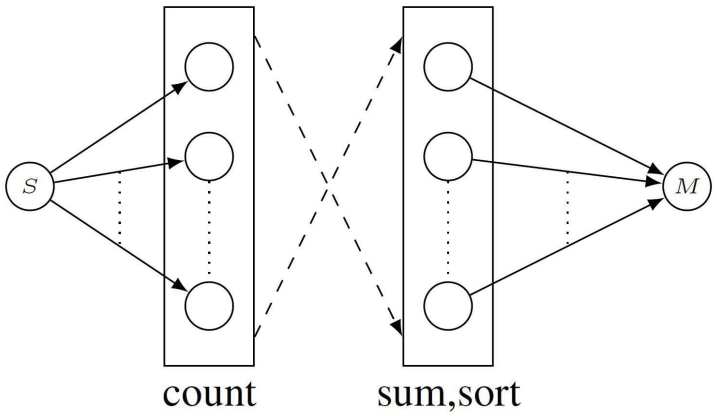
  A MapReduce implementation designed specifically for multi-core machines.

- KPN-MR

  A Kahn process network configured for MapReduce using two instantiations connected in series

$$S \longrightarrow MR_1 \longrightarrow MR_2 \longrightarrow M$$

- KPN-FLEX



Results

Doc Size (MB)	**Phoenix** (s)	**KPN-MR** (s)	**KPN-FLEX** (s)
10	0.30	0.32	0.11
50	0.98	1.86	0.37
100	2.04	4.23	0.74

Experimental for K-means

Points/Group	**Phoenix** (s)	**KPN-MR** (s)	**KPN-FLEX** (s)
100K/100	2.39	1.83	1.51
200K/50	3.92	3.20	2.24
100K/100	5.43	5.19	4/26

NewSQL,VoltDB,& H-Store

Based on Presentation given by
Michael Stonebraker, CTO VoltDB, Inc

“Big Data”?

- Big volume
  - I have too much data
- Big velocity
  - Data is coming at me too fast
- Big variety
  - I have too many data sources

Traditional Transaction Processing

- Remember how we used to buy airplane tickets in the 1980s
  - By telephone
  - Through an intermediary (professional terminal operator)
- Commerce at the speed of the intermediary
- In 1985, 1,000 transactions per second was considered an incredible stretch goal!!!!
  - HPTS (1985)

Traditional Transaction Processing

- Workload was a mix of updates and queries
- To an ACID data base system
  - Make sure you never lose my data
  - Make sure my data is correct
- At human speed
- Bread and butter of RDBMSs (OldSQL)

How has TP Changed in 25 Years?

The internet

- Client is no longer a professional terminal operator
- Instead Aunt Martha is using the web herself

- Sends TP volume through the roof
- Serious need for scalability and performance

How has TP Changed in 25 Years?

PDAs

- Your cell phone is a transaction originator

- Sends TP volume through the roof
- Serious need for scalability and performance

Need in some traditional markets
for much higher performance!

And TP is Now a Much Broader Problem (New TP)

The internet enables a green field of new TP applications

- Massively multiplayer games (state of the game, leaderboards, selling virtual goods are all TP problems)
- Social networking (social graph is a TP problem)
- Real time ad placement
- Real time couponing

- And TP volumes are ginormous!!
- Serious need for speed and scalability!

And TP is Now a Much Broader Problem

Electronic commerce is here

- Wall Street electronic trading
- Real-time fraud detection
- Micro transactions (through your PDA)

- And TP volumes are ginormous!!
- Serious need for speed and scalability!

Add in High Velocity Ingest

- Real time click stream analysis
- Most anything upstream from Hadoop
- Or your data warehouse
- Real time risk assessment on Wall Street

- And TP volumes are ginormous!!
- Serious need for speed and scalability!

In all cases…..

- Workload is a mix of updates and queries
- Coming at you like a firehose
- Still an ACID problem
  - Don't lose my data
  - Make sure it is correct
- Tends to break traditional solutions
  - Scalability problems (volume)
  - Response time problems (latency)

Reality Check -- Size

- TP data base size grows at the rate transactions increase
- 1 Tbyte is a really big TP data base
- 1 Tbyte of main memory buyable for around $50K
  - (say) 64 Gbytes per server in 16 servers
- I.e. Moore's law has eclipsed TP data base size
- If your data doesn't fit in main memory now, then wait a couple of years and it will.....

To Go a Lot Faster You Have to……

- Focus on overhead
  - Better B-trees affects only 4% of the path length
- Get rid of ALL major sources of overhead
  - Main memory deployment – gets rid of buffer pool
    - Leaving other 75% of overhead intact
    - i.e. win is 25%

Solution Choices

- OldSQL
  - Legacy RDBMS vendors
- NoSQL
  - Give up SQL and ACID for performance
- NewSQL
  - Preserve SQL and ACID
  - Get performance from a new architecture

OldSQL

Traditional SQL vendors (the “elephants”)

- Code lines dating from the 1980’s
- “bloatware”
- Mediocre performance on New TP

NoSQL

- Give up SQL
- Give up ACID

Who Needs ACID?

- Funds transfer
  - Or anybody moving something from X to Y
- Anybody with integrity constraints
  - Back out if fails
  - Anybody for whom "usually ships in 24 hours" is not an acceptable outcome
- Anybody with a multi-record state
  - E.g. move and shoot

Who needs ACID in replication

- Anybody with non-commutative updates
  - For example, + and * don't commute
- Anybody with integrity constraints
  - Can't sell the last item twice….
- Eventual consistency means "creates garbage"

NoSQL Summary

- Appropriate for non-transactional systems
- Appropriate for single record transactions that are commutative
- Not a good fit for New TP
- Use the right tool for the job

NewSQL

- SQL
- ACID
- Performance and scalability through modern innovative software architecture

NewSQL

- Needs something other than traditional record level locking (1st big source of overhead)
  - timestamp order
  - MVCC
  - Your good idea goes here

NewSQL

- Needs a solution to buffer pool overhead (2nd big source of overhead)
  - Main memory (at least for data that is not cold)
  - Some other way to reduce buffer pool cost

NewSQL

- Needs a solution to latching for shared data structures (3rd big source of overhead)
  - Some innovative use of B-trees
  - Single-threading
  - Your good idea goes here

NewSQL

- Needs a solution to write-ahead logging (4th big source of overhead)
  - Obvious answer is built-in replication and failover
  - New TP views this as a requirement anyway
- Some details
  - On-line failover?
  - On-line failback?
  - LAN network partitioning?
  - WAN network partitioning?

Summary

Old TP

New TP

OldSQL for New OLTP		▪ Too slow ▪ Does not scale
NoSQL for New OLTP		▪ Lacks consistency guarantees ▪ Low-level interface
NewSQL for New OLTP		▪ Fast, scalable and consistent ▪ Supports SQL

A NewSQL Example: H-Store/VoltDB

- Main-memory Linux SQL DBMS
  - Anti-caching
- Multi-node and sharded
- Stored procedure interface
  - No locking
- Pure ACID
- Single threaded
- Built-in HA and durability
  - No log (in the traditional sense)

Anti-caching

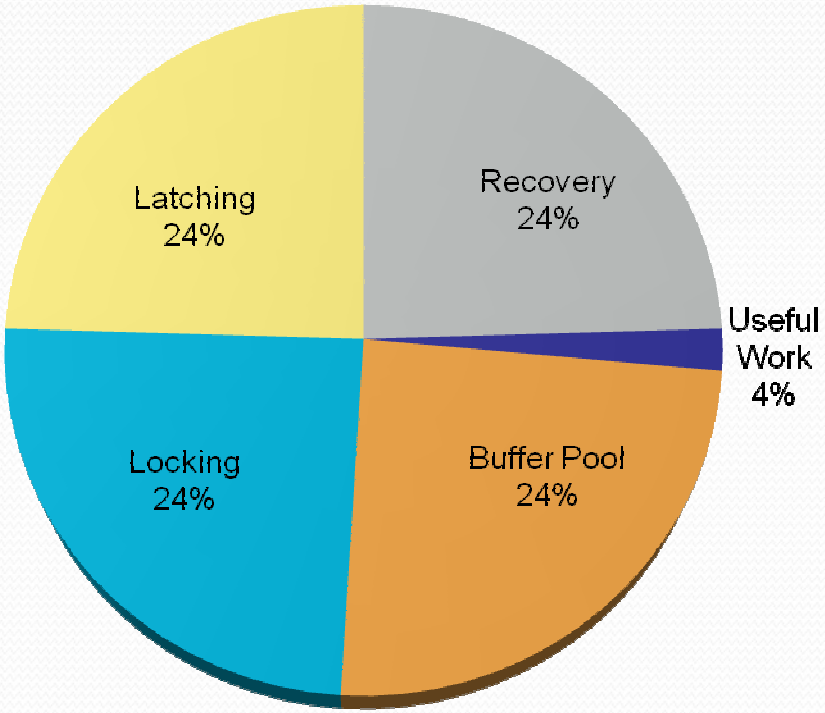
- Main memory format for data
- When memory fills
  - Gather col tuples and write to an archive
- When a transaction has a “miss”
  - Abort it
  - Find all the absent data
  - Reschedule transaction when all needed data in memory

What About Multicore?

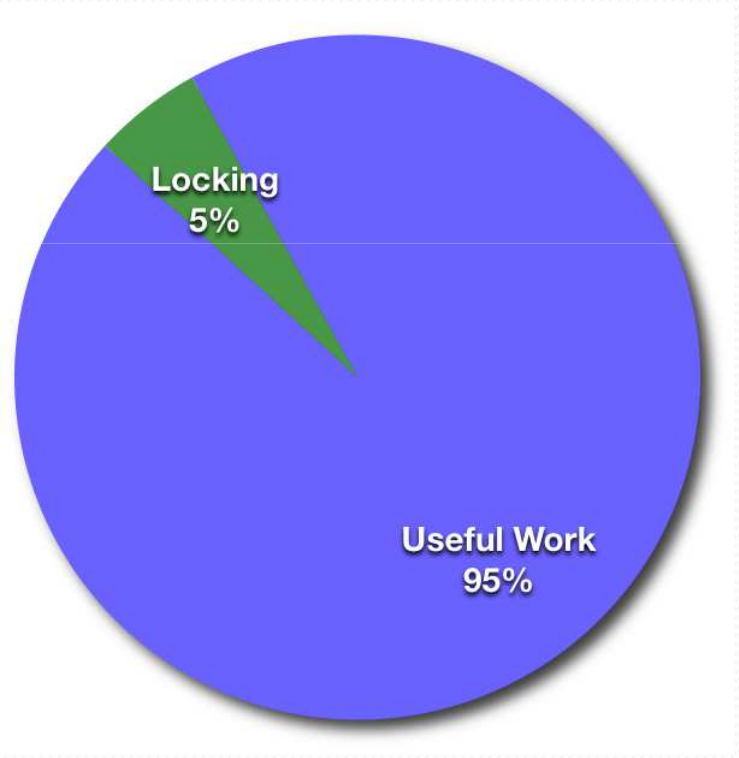
- For A K-core CPU, divide memory into K (non overlapping) buckets
- i.e. convert multi-core to K single cores

Where all the time goes… revisited

Before



VoltDB



Current VoltDB Status

- Runs a subset of SQL (which is getting larger)
- On VoltDB clusters (in memory on commodity gear)
- With LAN and WAN replication
- 70X a popular OldSQL DBMS on TPC-C
- 5-7X Cassandra on VoltDB K-V layer
- Scales to 384 cores (biggest iron we could get our hands on)
- Clearly note this is an open source system!

Performance of H-Store

	Simplified TPC-C with stored procedures	
Partitions	16	64
Number of concurrent clients	Not available	not available
Only local partition accesses	39,000	140,000
30% remote partition accesses	7,000	18,000

RubatoDB: A NewSQL System for OLTP and Big Data Applications

Li-Yan Yuan

RubatoDB: A Highly Scalable Staged Grid Database System for OLTP and Big Data Applications,
Proceedings of ACM CIKM 2014,
Li-Yan Yuan, Lengdong Wu, Jiahuai You, Chi Yan

Contents

- NewSQL systems
- Challenges
- Staged Architecture
- Formula Protocol for Concurrency
- RubatoDB and its Implementation Techniques
- Performance Evaluation
- Future work

NewSQL Systems

- Classical SQL
  - ACID
  - Scalability ?
- NoSQL (Not only SQL)
  - Scalability
  - Performance
  - BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency)
    - Weaker consistency for scalability and availability
- NewSQL
  - ACID
  - Scalability
  - Performance

Sample NewSQL systems

- H-store
  - distributed, main memory executor nodes
  - a pre-defined store procedure as a transaction
  - TPC-C(simplified): 160K/64 partition/16nodes(sigmod2012)
  - no standard transaction statements
  - not very impressive but considered the most promising
- VoldDB
  - distributed in memory operational database, shared nothing
  - ACID-complaint DBMS using stored procedures
  - very fast
  - no standard transaction statements

Challenges

- Scalability vs ACID
  - CAP Theorem
    - Consistency, Availability, Partition Tolerance
- System architecture
  - A collection of commodity servers, shared nothing
  - SMP, MPP: IBM Netezze, Oracle Exadata
  - H-store: distributed main memory OLTP database system
  - Main memory DBMS
- Concurrency control protocol
  - Distributed data vs. global consistency
  - No satisfactory solutions, not yet
- Performance

Objectives of RubatoDB

- Full support of SQL
- Scalability
  - A collection of commodity servers
    - shared nothing
- ACID properties
  - Standard transaction processing
    - Not stored procedures
  - TPC-C benchmark test
- Performance
  - Comparable to NoSQL systems

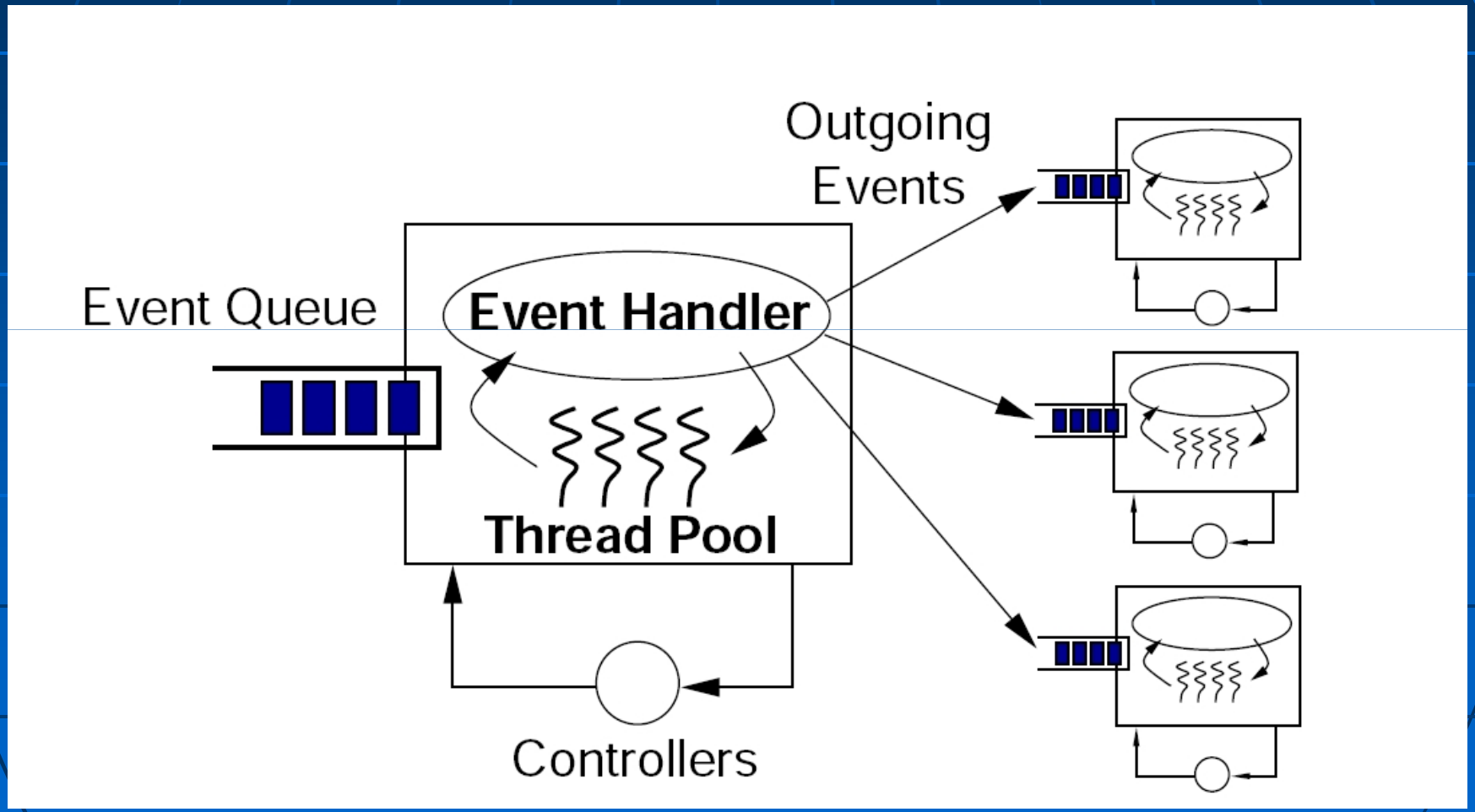
Distinguished Features of RubatoDB

- Staged architecture
  - The system runs as a relay team
  - Each player (stage) performs a very specific task
- Formula Protocol for Concurrency (FPC)
  - A novel implementation of the multi-version tamp-stamp protocol
    - Formula caching
    - Dynamic timestamps

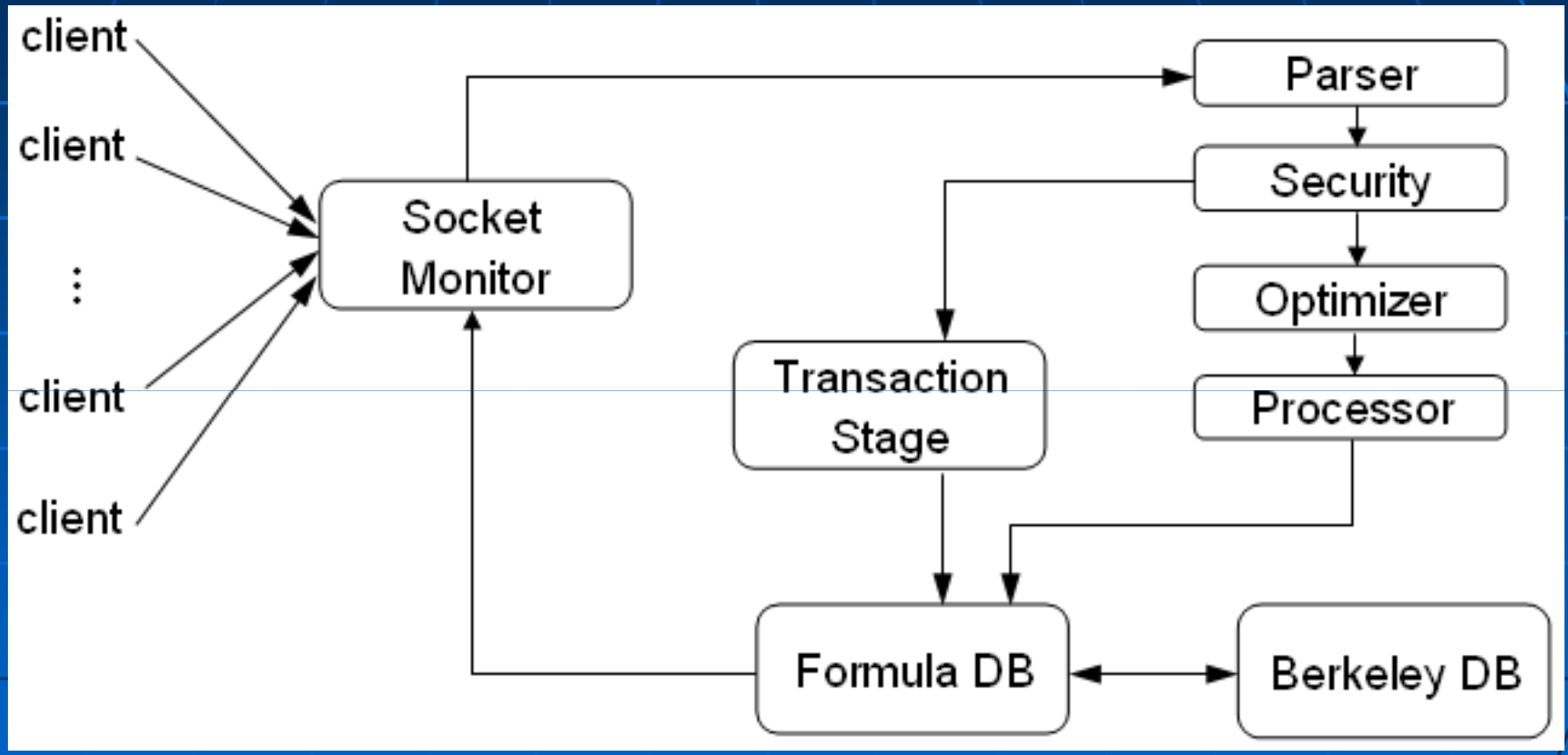
Staged architecture

- The system is a network of stages
  - Each stage is an independent software module performing a very specific task
    - Share nothing with others
    - Access only local storage devices
- Communication between any two stages is by means of operation queues
  - Block-read from the input queue
  - Non-block write to the output queues
- Evaluation of requests to the system
  - Relay-style
  - Passing a request from one stage to the next until it is completed
  - Single thread

Stage architecture



Stage architecture



Staged architecture

- Sample systems using Staged architecture
  - MapReduce
  - Kahn process network
  - web-servers
  - staged database management systems
- No solutions to concurrency problem
  - NewSQL === Scalability + ACID

Formula Protocol for Concurrency

- Concurrency control problem
  - NoSQL
    - Why BASE, not ACID ?
  - H-store/VoltDB
    - Why stored procedures ?
      - Not a standard SQL approach
        - Start transaction; S1, S2; …/ Sn; Commit;
      - No proper concurrency control protocol
    - Why performance problem
      - Very fast, like VoltDB, if all tables accessed by a transaction, in the form of a stored procedure access only one partition
      - System pauses if otherwise

Formula Protocol for Concurrency

- Difficulty of distributed concurrency
  - Distributed data vs. global consistency
  - Two-phase locking
    - Communication cost over the network
  - Multi-version timestamps
    - Memory cost
    - Clearance time
    - Fixed timestamps
      - Unnecessary block
  - Others?

Formula Protocol for Concurrency

A novel implementation of multi-version timestamps protocol with the following two distinct features

- Update formula Caching
  - Store update formulas, not multiple version
- Dynamic timestamps ordering
  - Modify the timestamps if needed

Formula Protocol for Concurrency

- Why formulas, not just multi-version?
  - Smaller sizes
    - None key updates
    - Numerical values
    - String values
    - Page vs formula
  - Commutative updates
    - Incremental updates
  - Dynamic time ordering
    - Clear the stored formulas as early as possible
    - Avoid unnecessary waiting and/or blocking
- Performance will tell the results

Demo Example

	T_{10}	T_{20}	T_{30}	A	B	C
t_1				90	100	80
t_2	W(B=B×1.1)			90	100	80
t_3			W(B=B+10)	90	100	80
t_4			W(C=C+10)	90	100	80
t_5			commit	90	110	90
t_6		R(B=121)				
t_7	commit			90	121	90
t_8		commit		90	121	90

The original timestamps order: T10, T20, T30

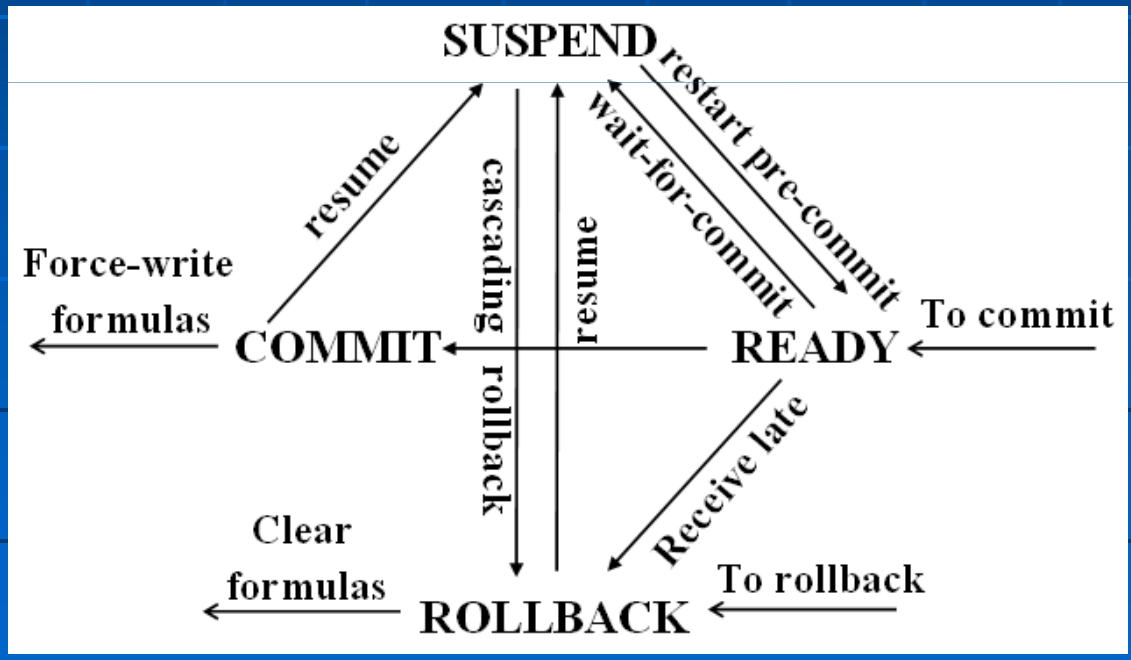
The dynamic one: T30, T10, T30

Formula Protocol for Concurrency

- Each data item x
  - Lst(x, Ni): largest TS that read x on Node Ni
  - List(x, Ni): the list of update formulas
- Read/Write operation by T1
  - R1: read proper values
    - Record read_by(Tu, x, T1)
  - W1: if TS(T1) < lrt(x,Ni)
    - rollback T1
  - W2: if lrt(x,Ni)=0 or TS(T1) = lrt(x,Ni)
    - Add the update formula into list(x,Ni)
  - W3: if TS(T1) > lrt(x,Ni) > 0
    - Add the update formula into list(x,Ni)
    - There exists T2 that read x before, thus
      - Record read_b4(T2,x, T1)

Formula Protocol for Concurrency

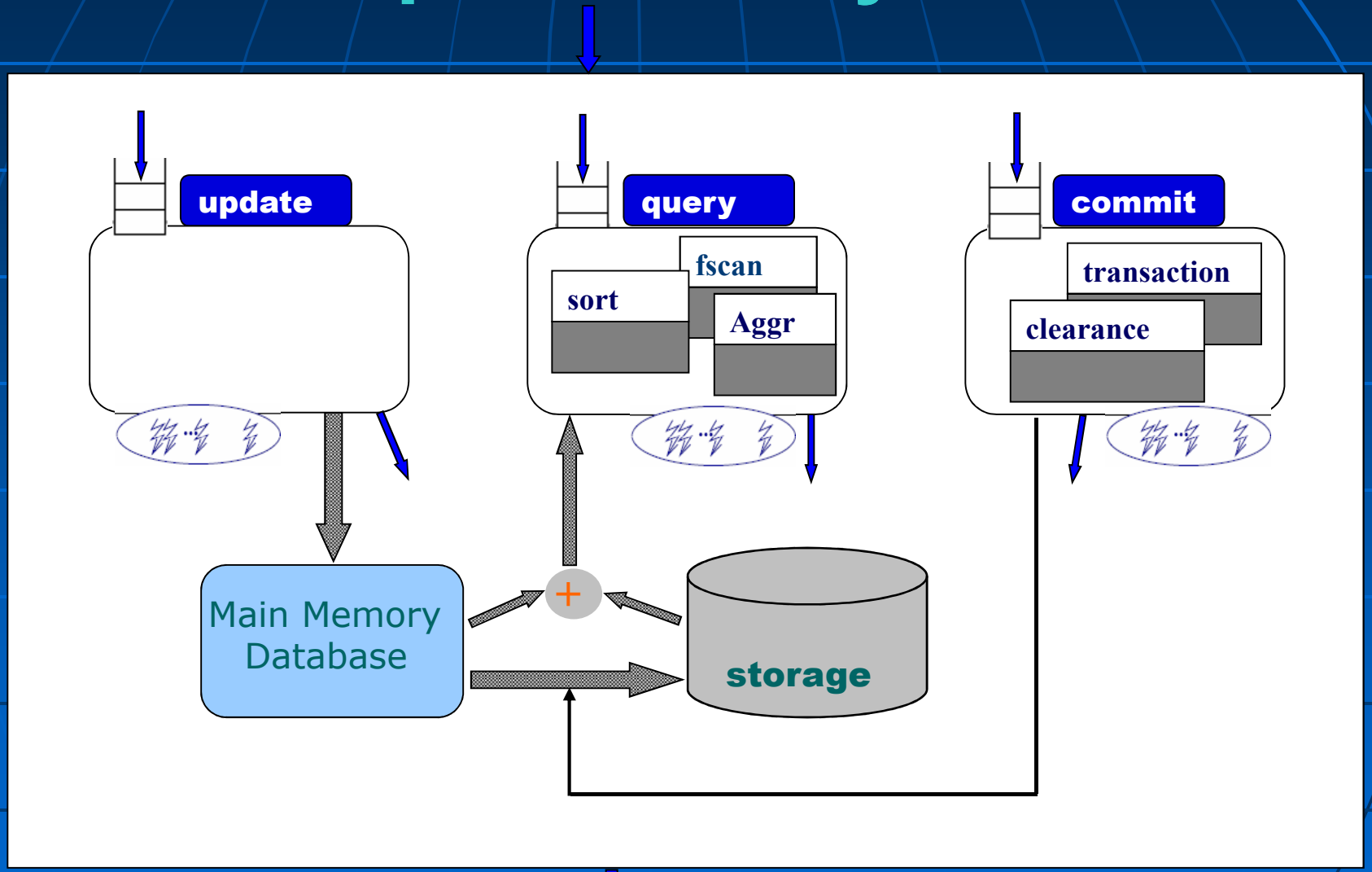
- Two phase commit protocol
  - Pre-commit
  - Commit
  - Rollback/cascading rollback



Implementation of FPC

- Formula DB,
  - One layer on the top of Berkeley DB
    - FDB->pre_commit(Ti)
    - FDB->commit(Ti)
    - FDB->rollback(Ti)
    - FDB->update(Ti,N,F,W)
    - FDB->get(Tid,Key,Data,Flags)
    - FDB->put(Tid,Key,Data,Flags)
    - FDB->del(Tid,Key,Flags)
    - FDB_Curssor(…)

Table Update/Query Module



Development of Rubato DB

- The standard SQL
  - the conform test of SQL'92: 98%
- Interfaces
  - JDBC
  - ODBC
  - Dot Net
  - Many others

Hybrid of row and column storages

- Traditional row database
  - Efficient for read-only operations
- Column storage
  - Efficient for frequent update operations
    - Reducing IO-wait
- Hybrid approach
  - Using the column storage for frequently updated columns

Lazy loading

- Basic idea
  - At any time, the system processes the request with lest conflict potential,
  - The system will not process any requests if the chances of conflict are high
    - Rather waiting than being rolled back

Soft instruction set

- A set of (software) instructions
  - Specify all the basic operations
  - Each instruction specifies
    - Operation id
    - Stage id,
    - Data source, and
    - All other parameters
- Instructions are only packets flowing through stages
  - except large volumn of data items

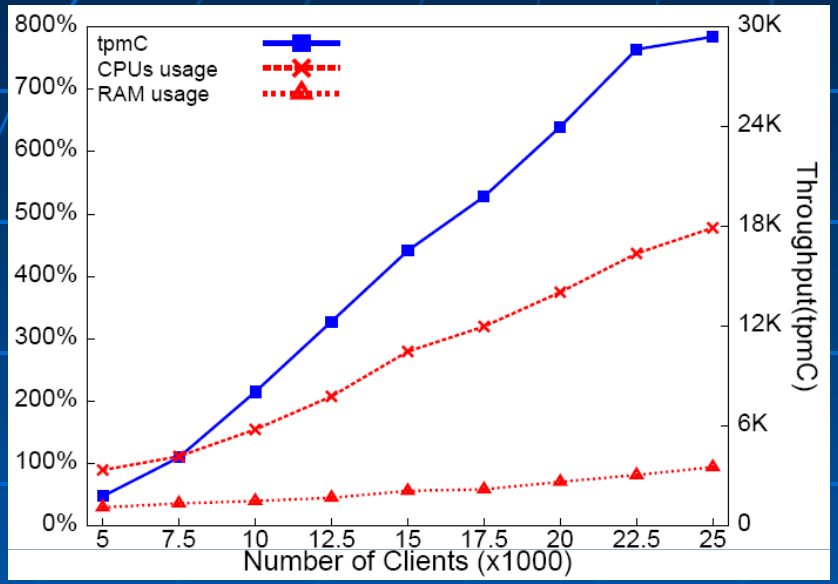
Experiments and Performance

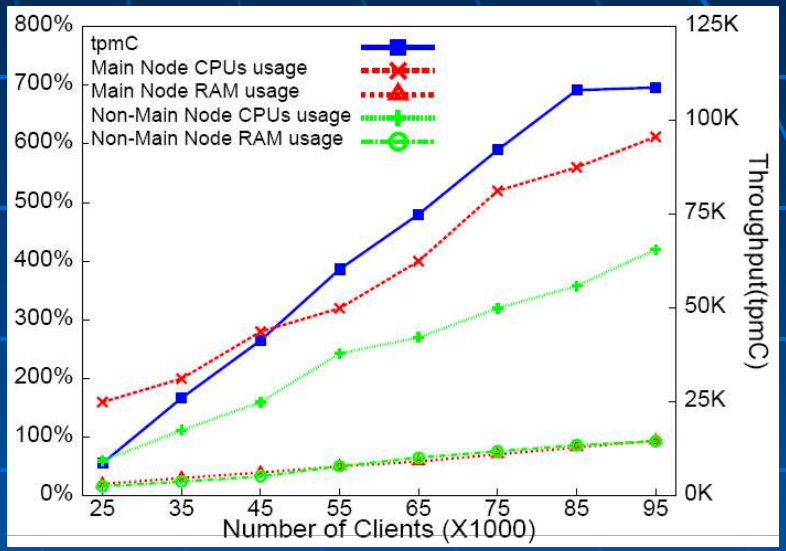
- Benchmark used
  - TPC-C for OLTP applications
  - YCSB for big data applications
  - BASIC and the test
    - A new consistency level between ACID and BASE
      - Basic Available, Scalability, Instant Consistency
- Objectives
  - Scalability
  - Performance for OLTP
  - Comparable to other NoSQL systems
  - Cost for BASIC

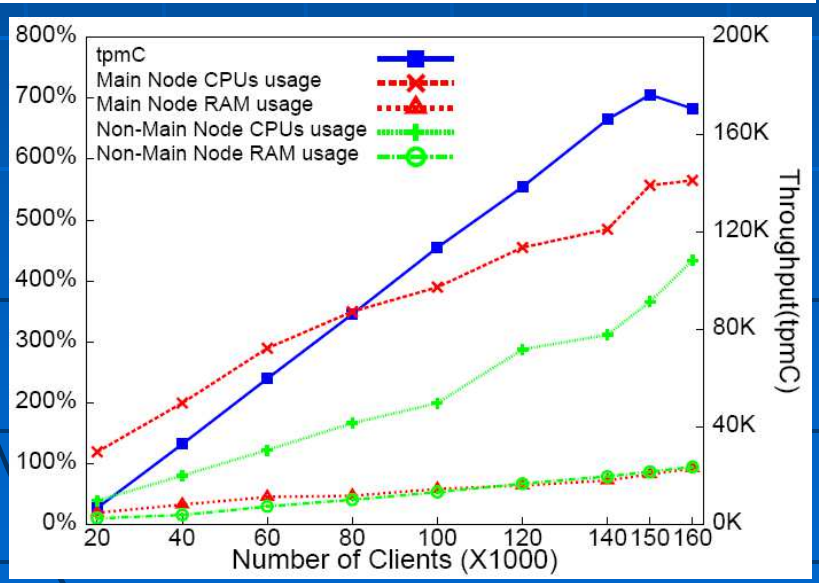
Experiment Set up

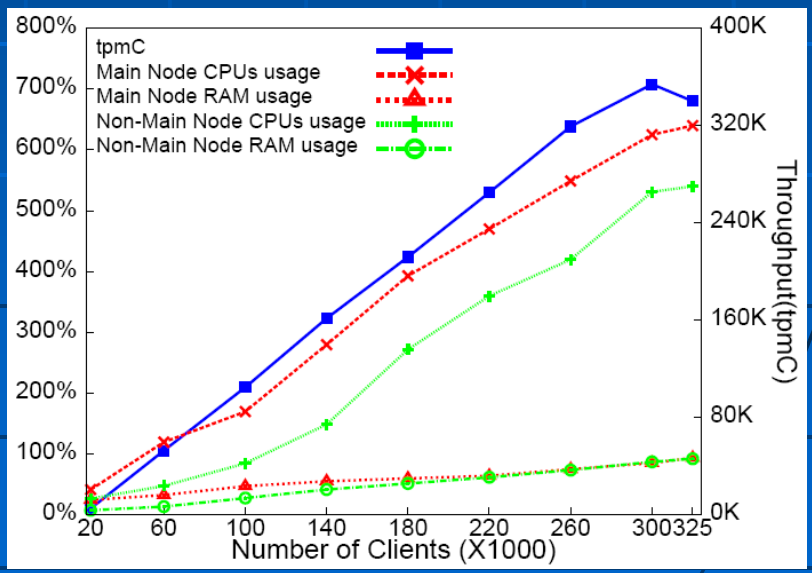
- Standard benchmark tests
  - TPC-C
  - YCSB
  - No stored procedures
  - All standard transaction statements
- 1-16 nodes
  - Dual quad-core Intel Xeon CPUs
  - 32GM main memory
  - SATA disks RAID0
  - Gigabit Lan
  - provided by Cybera

Scalability

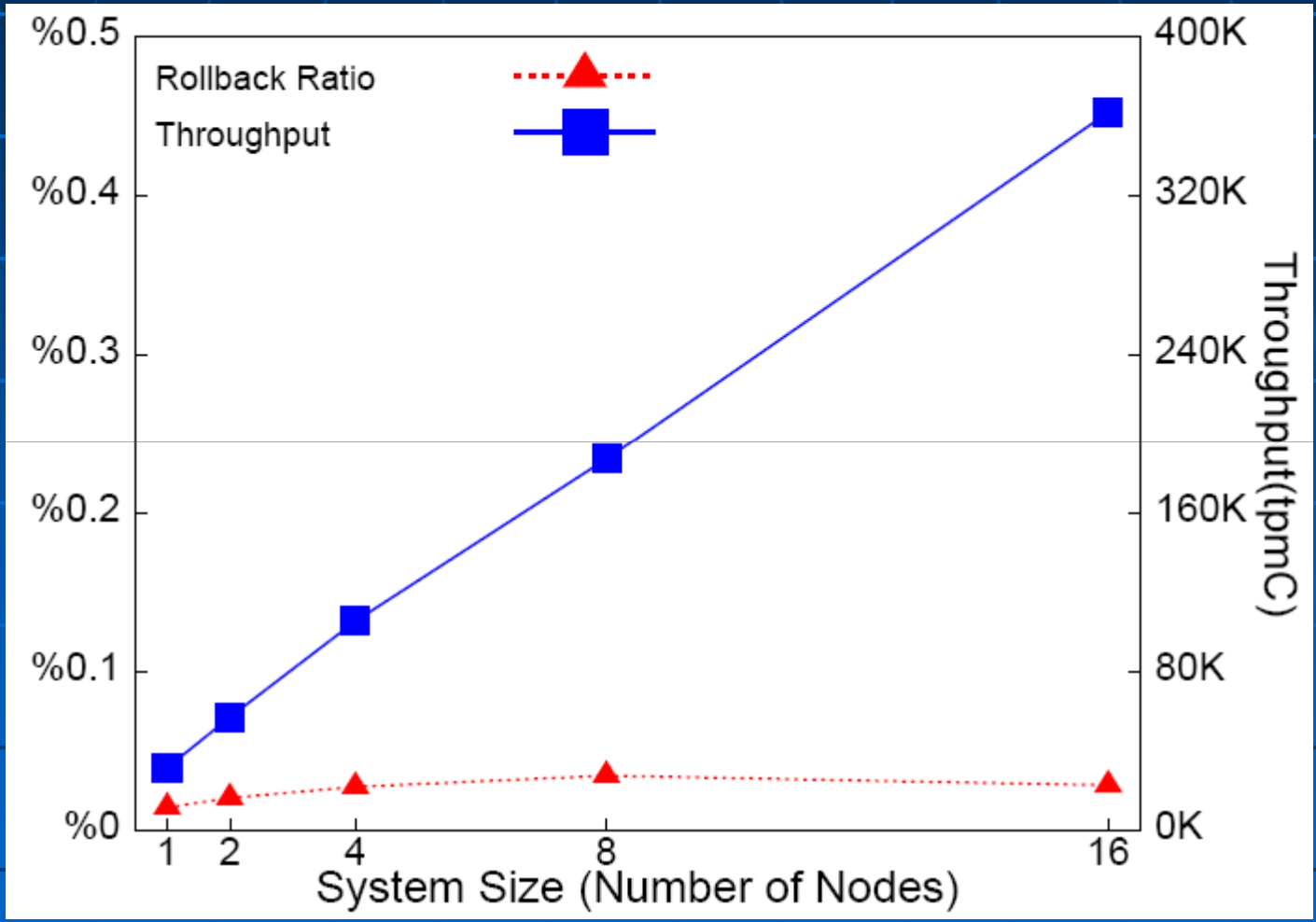








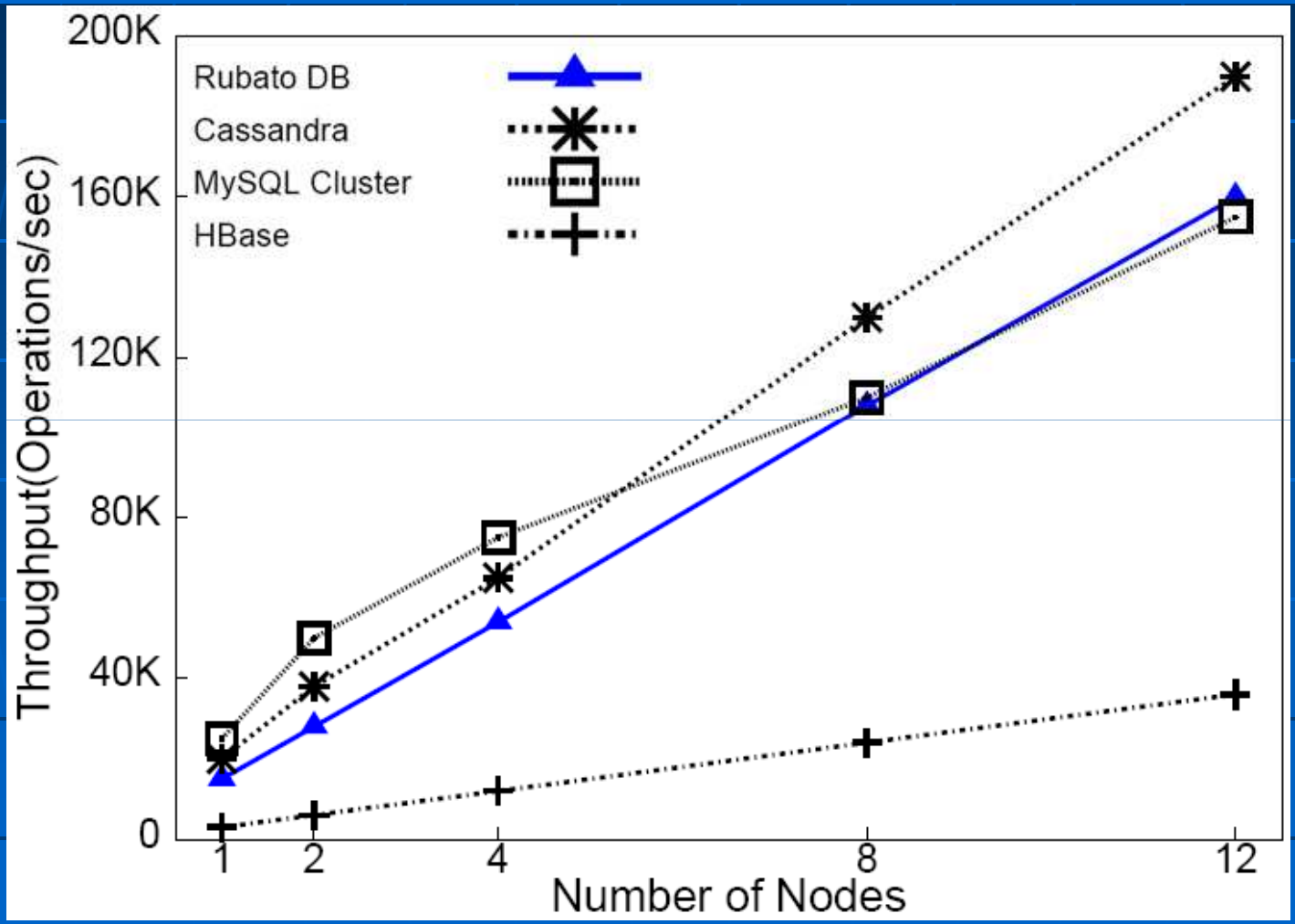
Scalability



Performance Comparisons

- RubatoDB, standard TPC-C test
  - 16 nodes
  - 325K clients, 363K tpmC, 825K transactions
  - 660K transaction if 10% guest wharehouses
- H-store, simplified TPC-C test
  - 64 partitions
  - No standard transaction statements
  - 160K transactions (all single partitions)
  - 60K transactions (10% distributed)
- One DBMS official TPC-C test
  - 260K tpmC for comparable hardware cost

YCSB for Big Data Applications



Experimental Summary

- RubatoDB is Highly scalable
  - OLTP with ACID
- Performance of RubatoDB for OLTP
  - A few times higher than S-store
- RubatoDB is comparable or better than other NoSQL systems in YCSB

Conclusions

- Staged architecture and FPC provide a satisfactory solution to a long outstanding problem
  - How to achieve Scalability as well as ACID

Future works

- FPC with data replications
  - RubatoDB does not utilize data replication
  - It may improve the performance
- How to obtain global timestamps
  - To eliminate one potential bottle-net
  - To increase availability
- How to improve the performance with remote guest accesses
  - Communication
  - ???

Distinguished Features of LogicSQL (continued)

- Hybrid of row and column storages
  - Difference between read and write
- Lazy loading
  - a novel load control implemented in the socket monitor stage
- Soft-instruction set

Data Mining and OLAP

- Introduction
  - OLTP (On Line Transaction Processing)
  - OLAP (On Line Analytic Processing)
  - Data Ming
- Association rules
- Data cube
- Data classification
- Cluster

1 Introduction

- OLTP: On Line Transaction Processing
  - Maintains a database that is an accurate model of some real-world enterprise.
  - Supports day-to-day operations.
  - Characteristics:
    * Short simple transactions
    * Relatively frequent updates
    * Transactions access only a small fraction of the database Processing
  - TPC-benchmark
  - Examples: Bank systems, university registration systems

- OLAP: On Line Analytic Processing
  - Uses information in database to guide strategic decisions.
  - Characteristics:
    - Complex queries
    - Infrequent updates
    - Transactions access a large fraction of the database
    - Data need not be up-to-date
  - Sample query:

    How many cases of tomatoes were sold in all northeast warehouses in the years 2010 and 2011?

- Data Mining is to extract knowledge from a database
  - How can information about salary, net worth, and other historical data be used to predict who will default on their mortgage?
  - Is it true that customers who purchase diapers more likely to purchase bear ?
  - How to classify all face-book users ?

- Classification of Data Mining Techniques
  - Based on data sources
  - Based on knowledge to be mined (discussed in this course)
    - **association rules**
    - **data cube**
    - **classification rules**
    - **clustering**
  - Based on techniques utilized

    data-driving, query-driving, statistics-based, mathematics-based, ...

2 Mining Association Rules

To discover important associations among items such that the presence of some items in a transaction will likely imply the presence of other items in the same transaction.

- How likely will a mortgage applicant go to bankruptcy ?
- Is it true that a customer who buy a camera is more likely to buy a memory card?
- Should we put bear next to diapers in a supermarket ?

  The answer is YES if customers who buy diapers are more likely to buy bear.

[1] Association Rules

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items,

$D = \{T_1, T_2, \ldots, T_m\}$ be a set of transactions such that $T_i \subseteq I$,

$X \subseteq I$ and $Y \subseteq I$ are sets of items

An *association rule* is an implication of the form

$$X \Rightarrow Y$$

where $X \cap Y = \emptyset$.

This a very simplified model.

[2] Examples of Association Rules

- In a grocery store,
  - The set I of all **items** are items available for sale in the store: diaper, milk, bear, etc.
  - A **transaction** T_i in D represents the set of all items purchased by one customer at one particular time.
  - An **association rule**

$$X \Rightarrow Y$$

    represents a fact that a customer who purchase X is more likely to purchase Y.

  Note that this is a simplistic view of shopping baskets, and some important information not considered

  - the quantity of each item purchased and
  - the price paid.

- In a high school
  - The set I of all **items** are activities participated by students, such as

    basket ball, hokey, light breakfast, ..., etc.
  - A **transaction** T_i in D represents the set of activities participated by an individual student.
  - An **association rule**

$$\{basket\ ball\} \Rightarrow \{light\ breakfast\}$$

    represents a fact that a student who plays basket ball is more likely to each light breakfast.

[3] Parameters of Association rules

- Challenges
  - How to define likelihood?

    probability and statistics
  - How to argue/convince the likelihood as specified by the data set?

    the sample space used to define likelihood must not be very small
- Two parameters
  - Confidence: based on the conditional probability specified in the data set
  - Support: based on the size of sample space in the data set

Parameters of Association rules

The rule $X \Rightarrow Y$

- holds in D with *confidence* c if $c\%$ of transactions in D that contain X also contain Y;

  Conditional probability:

  $$c = P(Y|X)$$

- has *support* s in D if $s\%$ of transactions in D contain $X \cup Y$.

  Size of sample space:

  $$s = P(X \cup Y).$$

An association rule is a pattern that states when X occurs, Y occurs with certain probability.

[4] Task of Mining Association Rules

- **Strong Rules:**

  Association rules with high confidence and strong supports.

- **The task of mining association rules:**

  To discover strong association rules in large databases:

[5] Goal and key features of Mining Association Rules

- **Goal:**

  Find all association rules that satisfy the user-specified minimal confidence C and minimum support S.

- **Key Features**

  - Completeness: find all such rules
  - No target item(s) on the right-hand-side
  - Mining with large volume of data on hard disk (not in memory)

Example 1 • Consider the following transactions:

T_1	beef, chicken, mild
T_2	beef, cheese
T_3	cheese, boots
T_4	beef, chicken, cheese
T_5	beef, chicken,clothes
T_6	chicken,clothes,milk
T_7	chicken, mild, clothes

- Assume that
  - confidence: C = 80 (%)
  - support: S = 30 (%)
- Association rules that satisfies the two conditions:
  - $\{clothese\} \Rightarrow \{milk, chicken\}\ with\ C = 100, S = 43\ \ (\frac{3}{7}\%)$
  - ...
  - $\{beef\} \Rightarrow \{chcken\}\ with\ C = 75, S = 43\ \ (\frac{3}{7}\%)$

[6] Algorithms for Mining Association Rules

- There are a large number of them !!!
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.

  Given a set D of transactions, and a confidence C and a support S, the set of association rules existing is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study only one: the Apriori Algorithm

[7] Apriori Algorithm

- Probably the best known and widely used algorithm
- Two steps Methods:
  - Discover the large itemsets, i.e. the sets of itemsets that have transaction support above a pre-determined minimum support s.
    (also called frequent itemsets)
  - Use the large itemsets to generate the association rules for the database.

  The second step is rather straightforward.

Iterative algorithms for finding large itemsets

Step 1: finding the set L_1 of all itemsets of size 1 with support s
Step 2: finding the set L_2 of all itemsets of size 2 with support s based on L_1
Step 3: finding the set L_3 of all itemsets of size 3 with support s based on L_2
...
Since the size of large item sets, i.e., the maximum number of items in a set that satisfies the minimum support S is very small, probably less than, say 10. For example, for a supermarket, it is probably true that an item set with 10 items that will appear in 20% of all purchases.

The key idea is the *Apriori property*: Any subset of a large item set is also a large item set.

Example 2 Consider database D below and the supporting factor 40%

TID	Item
100	A C D
200	B C E
300	A B C E
400	B E

Thus, the minimum support is = 2.

Step 1: C_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

$\Rightarrow$ L_1

Itemset	Sup.
{A}	2
{B}	3
{C}	3
{E}	3

Step 2: C_2

Itemset	Sup.
{A B}	1
{A C}	2
{A E}	1
{B C}	2
{B E}	3
{C E}	2

$\Rightarrow$ L_2

Itemset	Sup.
{A C}	2
{B C}	2
{B E}	3
{C E}	2

Step 3: C_3

Itemset	Sup.
{B C E}	2

$\Rightarrow$ L_3

Itemset	Sup.
{B C E}	2

Note that $C_{k+1} = \{ (I \mid I \in L_k * L_k) \wedge (I' \in L_k \quad \text{for all} \quad I' \subset I)\}$

Generating rules from large itemsets

- Large item sets $\neq$ association rules
- One more step is needed to generate association rules
- For each large itemset X

  For each proper nonempty subset A of A

  - Let B = X - A
  - $A \Rightarrow B$ is an association rule if it satisfies the confidence C, i.e., if

    This can be easily checked because

    * X, A, B are all large itemsets whose supports are known
    * the confidence of $A \Rightarrow B$ is specified by

$$\frac{support(A \cup B)}{support(A)}$$

[8] Properties of Apriori Algorithm

It seems very expensive ???

- Let K be the size of the largest itemset
- It takes at most K passes over the data set (K scans)
- item In practice, K is bounded by 10.
- The algorithm is indeed very fast
- Scale up to large data sets
- Under some conditions, all rules can be found in linear time.

[9] Interestingness of Association Rules

Given confidence c and supportness s, we have found all the association rules that satisfies these two conditions.

Let the following be such a rule discovered by the Apriori Algorithm.

$$A \Rightarrow B$$

Does the rule really represent a fact that A is associated with B ?

Example 3 Consider a hight school of 5000 students:

- 3000 play basketball (60%)
- 3750 eat cereal (75%)
- 2000 play basketball and eat cereal (40%)

and the rule

$$\{basketball\} \Rightarrow \{eat\_cereal\}$$

with 40% support and 67% confidence.
The rule is **very strong** because the confident is $\frac{2000}{3000} = 67\%$ while the support is 40%.

- By common sense, a student who plays basketball is likely to eat something more than cereal.
- By the association rule, a student who plays basketball is more like to each cereal.
- **Is something wrong here**
  - with the common sense, or
  - with the association rule ?

The rule is misleading for 75% of students eat cereal while only 67% of those who play basketball eat cereal.

The confidence of the rule, that is, the conditional probability $P(eat\_cereal|basketball)$ is less than $P(eat\_ceral)$.

Not all the discovered strong rules are interesting.

Interesting:

A rule $A \Rightarrow B$ is interesting only if its confidence exceeds a certain measure.

- For some suitable constant d,

$$\frac{P(A \cap B)}{P(A)} - P(B) > d$$

Notions of interestingness and/or usefulness of discovered rules.

[10] Summary of Mining Association Rules

- The given database consists of a set I of items, and a set D of transactions.
- We shall also be given, by the experts in the respective application areas, the following three parameters
  - confidence C,
  - supportness S, and
  - interestingness D.
- Discover all association rules that satisfy the three conditions specified by the aforementioned parameters.
- Apriori Algorithm is widely used to discover all such association rules

[11] Efficiency of mining association rules

- database scan reduction
- sampling: mining with adjustable accuracy
- incremental updating of discovered association rules
- parallel data mining

3 Data Cube

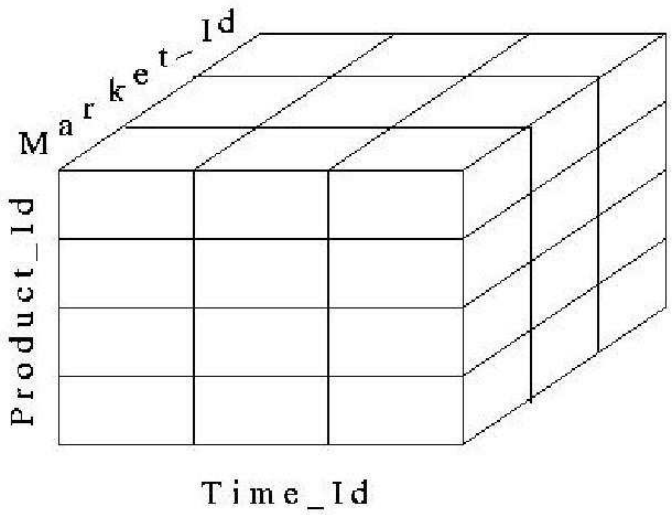
OLAP and Fact Tables

- Many OLAP applications are based on a fact table
- For example, a supermarket application might be based on a table

  Sales (Market_Id, Product_Id, Time_Id, Sales_Amt)
- The table can be viewed as multidimensional
  - Market_Id, Product_Id, Time_Id are the dimensions that represent specific supermarkets, products, and time intervals
  - Sales_Amt is a function of the other three

A Data Cube

- Fact tables can be viewed as an N-dimensional data cube



ues for Sales_Amt

Dimension Tables

- The dimensions of the fact table are further described with dimension tables
- Fact table:

  Sales (Market_id, Product_Id, Time_Id, Sales_Amt)
- Dimension Tables:
  - Market (Market_Id, City, State, Region)
  - Product (Product_Id, Name, Category, Price)
  - Time (Time_Id, Week, Month, Quarter)

Star Schema

- The fact and dimension relations can be displayed in an E-R diagram, which looks like a star and is called a star schema

Aggregation

- Many OLAP queries involve aggregation of the data in the fact table
- For example, to find the total sales (over time) of each product in each market, we might use

```
SELECT    S.Market_Id, S.Product_Id, SUM (S.Sales_Amt)
FROM      Sales  S
GROUP BY  S.Market_Id, S.Product_Id
```

- The aggregation is over the entire time dimension and thus produces a two-dimensional view of the data. (Note: aggregation here is over time, not supermarkets or products.)

Aggregation over Time

- The output of the previous query

Product_Id \ Market_Id	**SUM(Sales_Amt)**	M1	M2	M3	M4
	P1	3003	1503	...	...
	P2	6003	2402	...	...
	P3	4503	3	...	...
	P4	7503	7000	...	...
	P5	...	...	...	...

Drilling Down and Rolling Up

- Some dimension tables form an aggregation hierarchy

  Market_Id $->$ City $->$ State $->$ Region

- Executing a series of queries that moves down a hierarchy (e.g., from aggregation over regions to that over states) is called drilling down
  - Requires the use of the fact table or information more specific than the requested aggregation (e.g., cities)
- Executing a series of queries that moves up the hierarchy (e.g., from states to regions) is called rolling up
  - Note: In a rollup, coarser aggregations can be computed using prior queries for finer aggregations

Drilling Down

1. Drilling down on market: from Region to State

   Sales (Market_Id, Product_Id, Time_Id, Sales_Amt)

   Market (Market_Id, City, State, Region)

2.
```
SELECT   S.Product_Id, M.Region, SUM (S.Sales_Amt)
FROM     Sales  S,  Market  M
WHERE    M.Market_Id = S.Market_Id
GROUP BY S.Product_Id,  M.Region
```

3.
```
SELECT   S.Product_Id, M.State, SUM (S.Sales_Amt)
FROM     Sales  S,  Market  M
WHERE    M.Market_Id = S.Market_Id
GROUP BY S.Product_Id,  M.State,
```

Rolling Up

1 Rolling up on market, from State to Region

If we have already created a table, State_Sales, using

```
2. SELECT   S.Product_Id,  M.State, SUM (S.Sales_Amt)
   FROM     Sales  S,   Market  M
   WHERE    M.Market_Id = S.Market_Id
   GROUP BY S.Product_Id,  M.State
```

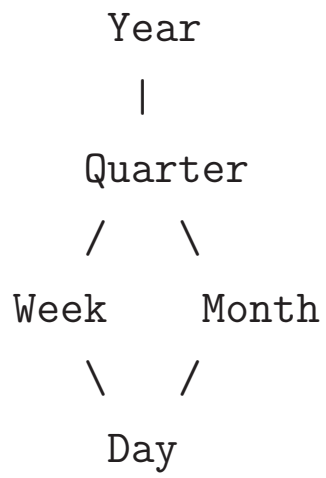
then we can roll up from there to:

```
3. SELECT        T.Product_Id,  M.Region, SUM (T.Sales_Amt)
   FROM         State_Sales  T,  Market  M
   WHERE        M.State = T.State
   GROUP BY T.Product_Id,  M.Region
```

Can reuse the results of query 1.

Time Hierarchy as a Lattice

- Not all aggregation hierarchies are liner
  - The time hierarchy is no lattice
    - Weeks are not contained in months
    - We can roll up days into weeks or months, but we can only roll up weeks into quarters



The CUBE Operator

To construct the following table, would take 4 queries (next slide)

SUM(Sales_Amt)		Market_ID			Total
		M1	M2	M3	
	P1	3003	1503	15003	19509
Product_ID	P2	6003	2402	24003	32408
	P3	4503	3	33	4539
	P4	7503	7000	9903	24406
	Total	21012	10908	48942	80862

The Four Queries

```
For the table entries, without the totals (aggregation on time)
   SELECT    S.Market_Id,  S.Product_Id,  SUM (S.Sales_Amt)
   FROM      Sales S
   GROUP BY  S.Market_Id, S.Product_Id
For the row totals (aggregation on time and markets)
   SELECT    S.Product_Id,  SUM (S.Sales_Amt)
   FROM      Sales S
   GROUP BY  S.Product_Id
For the column totals (aggregation on time and products)
  SELECT     S.Market_Id,  SUM (S.Sales)
  FROM       Sales S
  GROUP BY   S.Market_Id
For the grand total (aggregation on time, markets, and products)
  SELECT     SUM (S.Sales)
  FROM       Sales S
```

Definition of the CUBE Operator

- Doing these three queries is wasteful
  - The first does much of the work of the other two: if we could save that result and aggregate over Market_Id and Product_Id, we could compute the other queries more efficiently
- The CUBE clause is part of SQL:1999

  GROUP BY CUBE (v1, v2, , vn)

  Equivalent to a collection of GROUP BYs, one for each of the 2^n subsets of v1, v2, ..., vn

Example of CUBE Operator

The following query returns all the information needed to make the previous products/markets table:

```
SELECT   S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
FROM     Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```

ROLLUP

- ROLLUP is similar to CUBE except that instead of aggregating over all subsets of the arguments, it creates subsets moving from right to left
  - GROUP BY ROLLUP (A1,A2,,An) is a series of these aggregations:
  - GROUP BY A1 ,, An-1 ,An
  - GROUP BY A1 ,, An-1
  - ...
  - GROUP BY A1, A2
  - GROUP BY A1
  - No GROUP BY
- ROLLUP is also in SQL:1999

Example of ROLLUP Operator

```
SELECT   S.Market_Id,S.Product_Id,SUM(S.Sales_Amt)
FROM     Sales S
GROUP BY ROLLUP(S.Market_Id,S.Product_Id)

first aggregates with the finest granularity:
    GROUP BY S.Market_Id,S.Product_Id
then with the next level of granularity:
    GROUP BY S.Market_Id
then the grand total is computed with no GROUP BY clause
```

ROLLUP vs. CUBE

- The same query with CUBE:
  - first aggregates with the finest granularity:

```
GROUP BY S.Market_Id,S.Product_Id
```

  - then with the next level of granularity:

```
GROUP BY S.Market_Id
```

    and

```
GROUP BY S.Product_Id
```

  - then the grand total with no GROUP BY

- Can you write a single SQL query, without using GROUP BY CUBE, that is equivalent to the following query?

```
SELECT   S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
FROM     Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```

- Can you use SQL query facilities, without using GROUP BY CUBE, to retrieve the same result table as the following one with comparable evaluation performance?

```
SELECT   S.Market_Id, S.Product_Id, SUM(S.Sales_Amt)
FROM     Sales S
GROUP BY CUBE (S.Market_Id, S.Product_Id)
```

Materialized Views

The CUBE operator is often used to PRE-compute aggregations on all dimensions of a fact table and then save them as a materialized views to speed up future queries

Implementation Issues

- OLAP applications are characterized by a very large amount of data that is relatively static, with infrequent updates
  - Thus, various aggregations can be PRE-computed and stored in the database
  - Star joins, join indices, and bitmap indices can be used to improve efficiency (recall the methods to compute star joins in Chapter 14)
  - Since updates are infrequent, the inefficiencies associated with updates are minimized

4 Data Classification

Data classification is a process which finds the common properties among a set of objects in a database and classifies them into different classes, according to a *classification model*.

How to construct a classification model ?

- training set: a sample database such that
  - each tuple in the training set consists of the same set of multiple attributes as in a large database, and
  - each tuple is associated with a known class identify (label).
- analyze the training data and develop an accurate description for each class
- classify test data in the large database

[1] Classification based on decision trees

A supervised machine learning method that constructs decisions trees from a set of examples.

Given

- a set of objects that are described in terms of attribute values, and
- a *training set* of objects whose class is known.

The induction task is to develop a classification rule, based on the training set, that can determine the class of any object from its value of attributes.

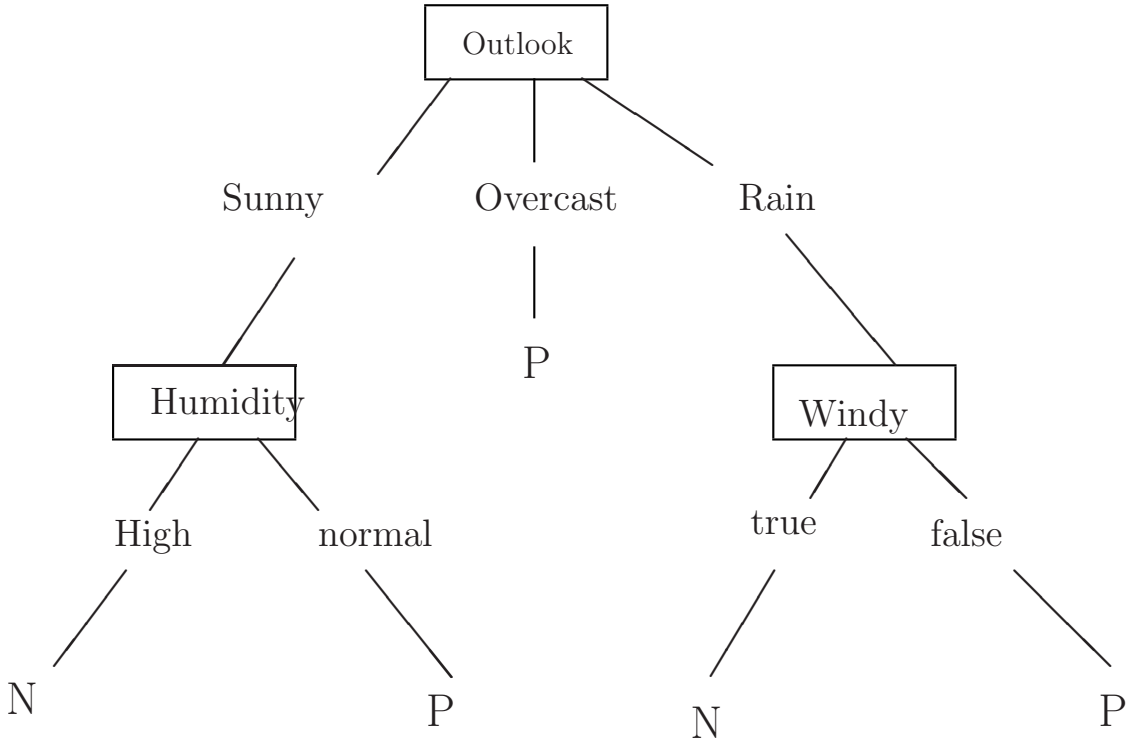
A decision tree is defined/constructed as such a classification rule.

Formally, a decision tree is either

- a leaf node that contains a class name, or
- a non-leaf node (or, decision node) that contains an attribute test with a branch to another decision tree for each possible value of the attribute.

Example 4 Consider the following training set:

No	Outlook	Temperature	Humidity	Windy	Class
1	sunny	hot	high	false	N
2	sunny	hot	high	false	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N



Now consider a Saturday morning with the following attribute values:

Outlook: overcast, Temperature: cool Humidity: normal Windy: false

What is the class of this Saturday?

- How many decision trees are there for a given training set ?

  Finite but very large.

- The simpler, the better ?

- How to find the simplest one ?

  Generate all decision trees and then choose the one you like.

- Accuracy?

- ID3 Algorithm

ID3 Algorithm

An iterative approach

- randomly select a subset of training set: window
- generate a decision tree from the window
  - if the tree correctly classify all the objects in the training set, then output the tree
  - otherwise, expand the window by adding incorrectly classified objects into the window and continue the process

Challenges

- How to form a decision tree from an arbitrary collection of objects ?

  Not difficult: select an attribute for the root test that provides a non-trivial partition of the given set of objects.

- How to choose the root attribute such that the decision tree is simple ?

  It is difficult but interesting

Information-based approach

Let C contain p objects of class P and n objects of class N. We assume

- Any correct decision tree for C will classify objects in the same proportion as their representation in C.

  Thus, an arbitrary object will belong to P with probability $\frac{p}{p+n}$ and N $\frac{n}{p+n}$.

- A decision tree can be regarded as a course of a message P or N, with the expected information needed to generate this message given by

$$I(p,n) = -\frac{p}{p+n}log_2\frac{p}{p+n} - \frac{n}{p+n}log_2\frac{n}{p+n}$$

If attribute A with values $\{A_1, A_2, \ldots, A_v\}$ is used for the root, it will partition C into $\{C_1, C_2, \ldots, C_v\}$ where C_i contains objects that have value A_i. The the expected information required for the subtree C_i is $I(p_i, n_i)$.
Therefore, the information required for the tree with A as the root is then

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i)$$

The information gained by branching on A is therefore

$$gain(A) = I(p,n) - E(A)$$

How to choose the root attribute ?

Choose the attribute to branch on which gains the most information.

Example 5 (Example 4 continued)

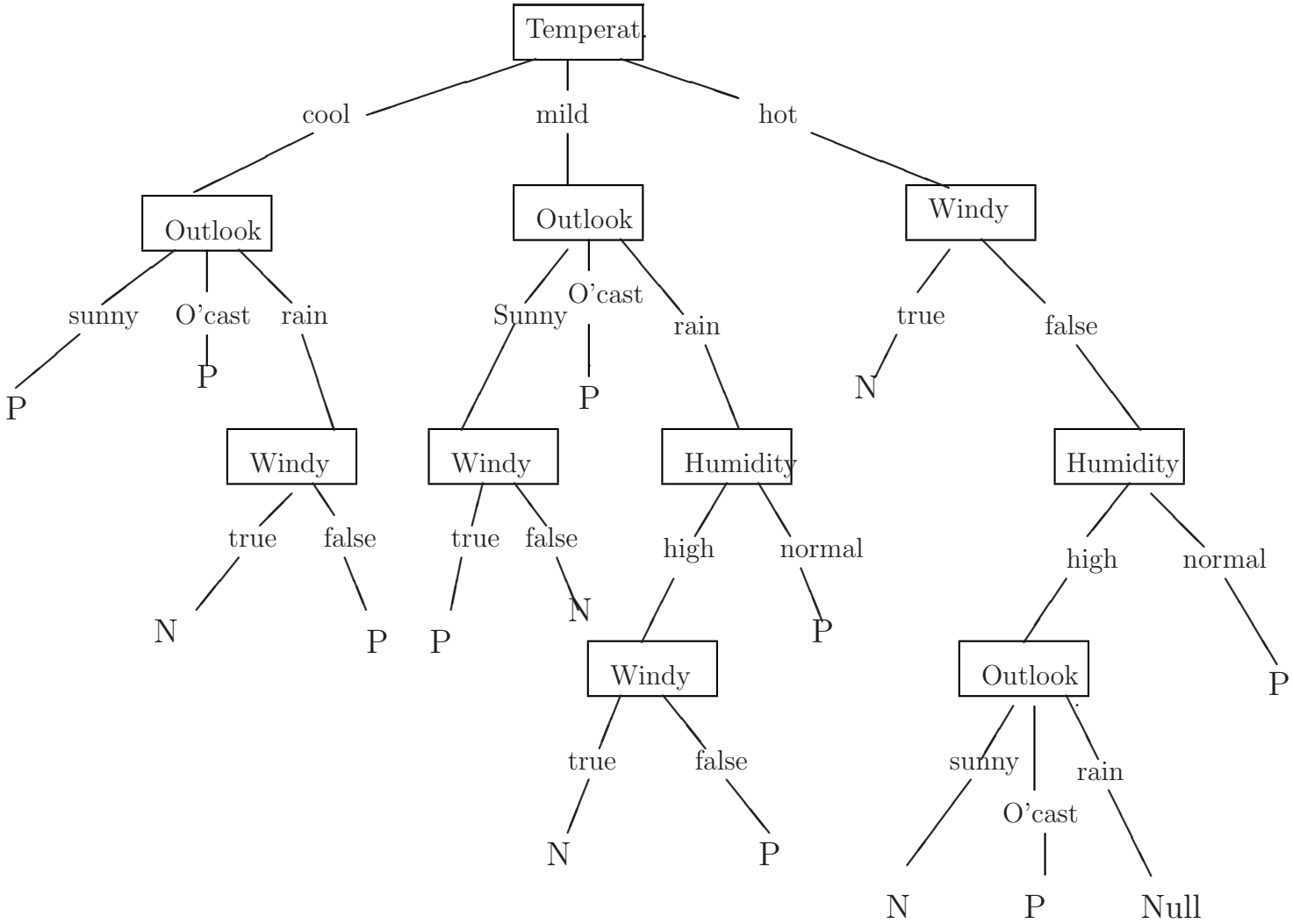
Of the 14 objects, 9 are in P and 5 in N, so the information required for classification is

$$I(p,n) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.940$$

The following table shows the expected information required and gain for each attribute to be the root.

Attribute	**Information required**	**Information gain**
Outlook	0.694	0.246
Temperature	0.911	0.029
Humidity	0.789	0.151
Windy	0.892	0.048

Obviously, by the information gain, Overlook is the best choice for the root.



A complex decision tree

Accuracy of decision trees

The accuracy of an algorithm for decision trees is usually determined

- use only part of the given set of objects as a training set, and
- check the resulting decision on the remainder.

Some experiments

- 1.4 million chess positions described in terms of 49 binary-valued attributes gave rise to 715 distinct objects divided 65%:35% between the classes. A decision tree contains about 150 nodes.

  The decision tree based on a training set that contains randomly selected 20% of these 715 objects correctly classified 84% of unseen objects.

- Another similar example: the decision tree based on a training set of 20% objects correctly classified 98% unseen objects.

[2] Other methods for data classification

- Statistical approaches
  - linear regression
  - liner discriminant analysis
- rough sets approach
- interval classifier
- neural network approach

[3] Methods for performance improvement

? Scaling up

5 Clustering Analysis

Clustering analysis, also called *unsupervised classification*, is a process of grouping physical or abstract objects into classes of similar objects.
It has been studied in

- statistics
- machine learning
- data mining

  identify clusters, i.e., densely populated regions in a large, multidimensional database, according to some distance measurement.

K-means Clustering Algorithm

Summary of Data Mining

- OLAP (On Line Analytic Processing)
  - Data Cube and Rollup
- Association Rules
  - $X \Rightarrow Y$
  - Confidence, Supportness, Interestness
  - Apriori algorithm
- Data Classification
  - Decision tree
  - ID3 Algorithm
- Clustering
  - K-means clustering method

Example Given the fact table below:

sales(market_id, item_id, sale)

and the following SQL statement:

```
SELECT market_id, item_id, SUM(sale)
FROM   sales
GROUP BY ROLLUP (market_id, item_id)
```

Write one (single) SQL statement without using any GROUP BY CUBE/ROLLUP clause that returns the same result table as the above SQL statement.

Solutions

```
SELECT market_id, item_id, SUM(sale)
FROM   sales
GROUP BY market_id, item_id
UNION
SELECT market_id, NULL, SUM(sale)
FROM   sales
GROUP BY market_id
UNION
SELECT NULL, NULL, SUM(sale)
FROM   sales
```

Example Given the fact table below:

sales(market, product, manufacturer, sale)

and the following SQL statement:

```
SELECT market, manufacturer, SUM(sale)
FROM   sales
GROUP BY CUBE (market, manufacturer)
```

Write a sequence of SQL statements without using any GROUP BY CUBE clause such that (1) the last statement in the sequence returns the same result set as the given Cube statement; and (2) it is as efficient to be evaluated as the the given Cube statement (ignoring the overhead).

```
CREATE TEMP TABLE tmp AS
SELECT market, manufacturer, SUM(sale) sum_sale
FROM   sales
GROUP  market, manufacturer;

SELECT market, manufacturer, sum_sale
FROM   temp
UNION
SELECT market, NULL, SUM(sum_sale)
FROM   temp
GROUP BY market
UNION
SELECT NULL, manufacturer, SUM(sum_sale)
FROM   temp
GROUP BY manufacture
UNION
SELECT NULL, NULL, SUM(sum_sale)
FROM   temp;
```

Example Consider a database with items $I = \{jeans, t\text{-}shirts\}$.
Assume that

- half of all the transactions in a clothe shop purchase jeans,
- one third of all transactions in the shop purchase t-shirts, and
- half of the transactions that purchase jeans also purchase t-shirts.

Write down all the non-trivial association rules you can deduce from the above information, given support and confidence of each rule.
Note $A \Rightarrow B$ is non-trivial if B must be non-empty.

Rule	Confidence	Support
$\emptyset \Rightarrow \{$ jeans $\}$		
$\emptyset \Rightarrow \{$ t-shirts $\}$		
$\{$ jeans $\} \Rightarrow \{$ t-shirts $\}$		
$\{$ t-shirts $\} \Rightarrow \{$ jeans $\}$		
$\emptyset \Rightarrow \{$t-shirts jeans $\}$		

Solution: The following table lists all the association rules $A \Rightarrow B$, where A and B are distinct subsets of $\{$ jeans, t-shirts $\}$ and B is non-empty, together with support and confidence of each rule.

Rule	Confidence	Support
$\emptyset \Rightarrow \{$ jeans $\}$	50%	50%
$\emptyset \Rightarrow \{$ t-shirts $\}$	33%	33%
$\{$ jeans $\} \Rightarrow \{$ t-shirts $\}$	50%	25%
$\{$ t-shirts $\} \Rightarrow \{$ jeans $\}$	75%	25%
$\emptyset \Rightarrow \{$t-shirts jeans $\}$	25%	25%

Example Construct a decision tree for the training set given below:

c_id	**Married**	**PreDefault**	**Income**	**class**
c1	yes	no	50	no
c2	yes	no	100	no
c3	no	yes	135	yes
c4	yes	no	125	no
c5	yes	no	50	no
c6	no	no	30	no
c7	yes	yes	10	no
c8	yes	no	10	yes
c9	yes	no	75	no
c10	yes	yes	45	no
c11	yes	no	60	yes
c12	no	yes	125	yes
c13	yes	yes	20	no
c14	no	no	15	no
c15	no	no	60	no
c16	yes	no	15	yes
c17	yes	no	35	no
c18	no	yes	160	yes
c19	yes	no	40	no
c20	yes	no	30	no

Solution: One such decision tree is given below.