Transactions in Cloud Computing

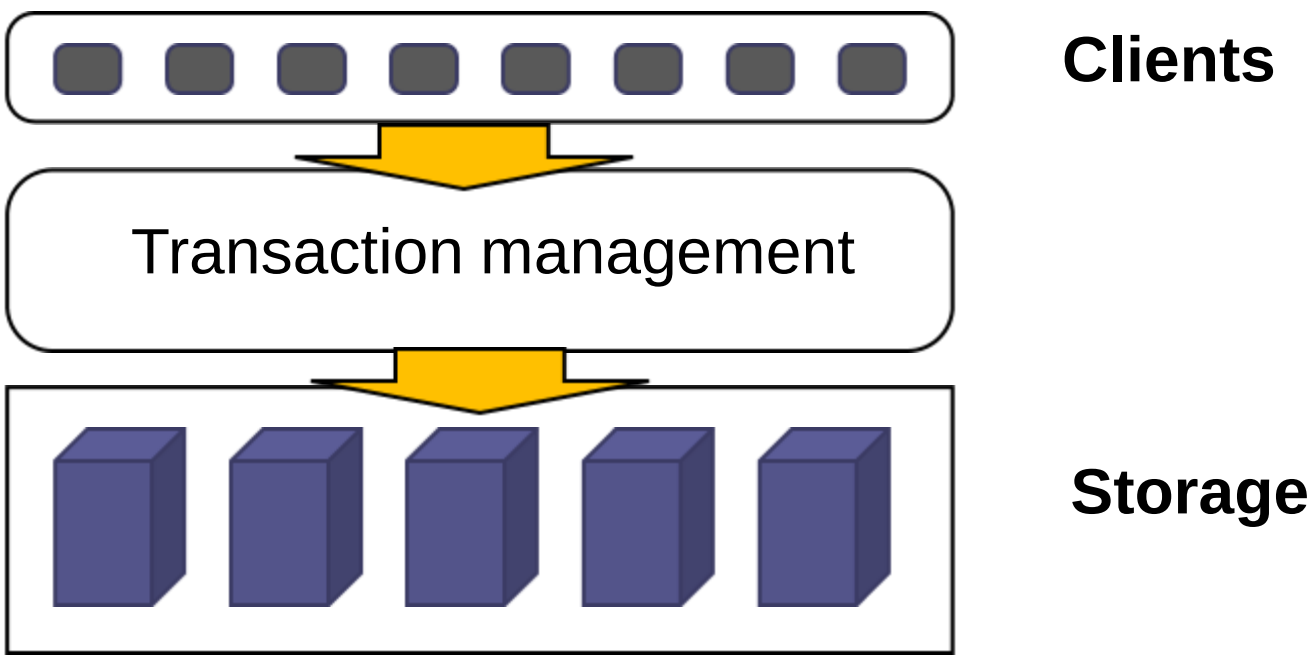
Presented by Fatemeh Shirazi
University of Saarland

Outline

- Introduction and basics
- Different transaction models
- Consistency Rationing
- Conclusion

Introduction

- Sequence of information exchange and related work (such as database updating)



Key points in transaction

- Consistency
- Scalability
- Cost
- Availability

ACID

- A set of properties that guaranteeing database reliable transactions
- Atomicity
- Consistency
- Isolation
- Durability

Outline

- Introduction and basics
- Different transaction models
- Consistency Rationing
- Conclusion

Amazon

- AWS MySQL
  - Follows a traditional (non-cloud-enabled) model
- AWS MySQL/R
- AWS RDS
  - Difference: RDS is Pre-packaged, users do not need to worry about managing the deployment, software patches, software upgrades and backups

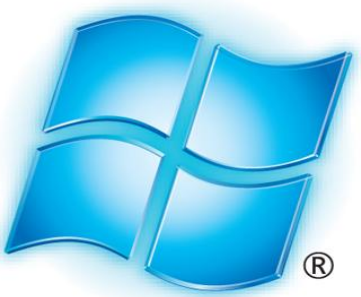
Amazon

- AWS SimpleDB
  - Amazons own database service
  - Only supports a low level of consistency called *eventual consistency*
- AWS S3
  - Supports *eventual consistency*
  - To improve performance caching is carried out in the application servers

Others

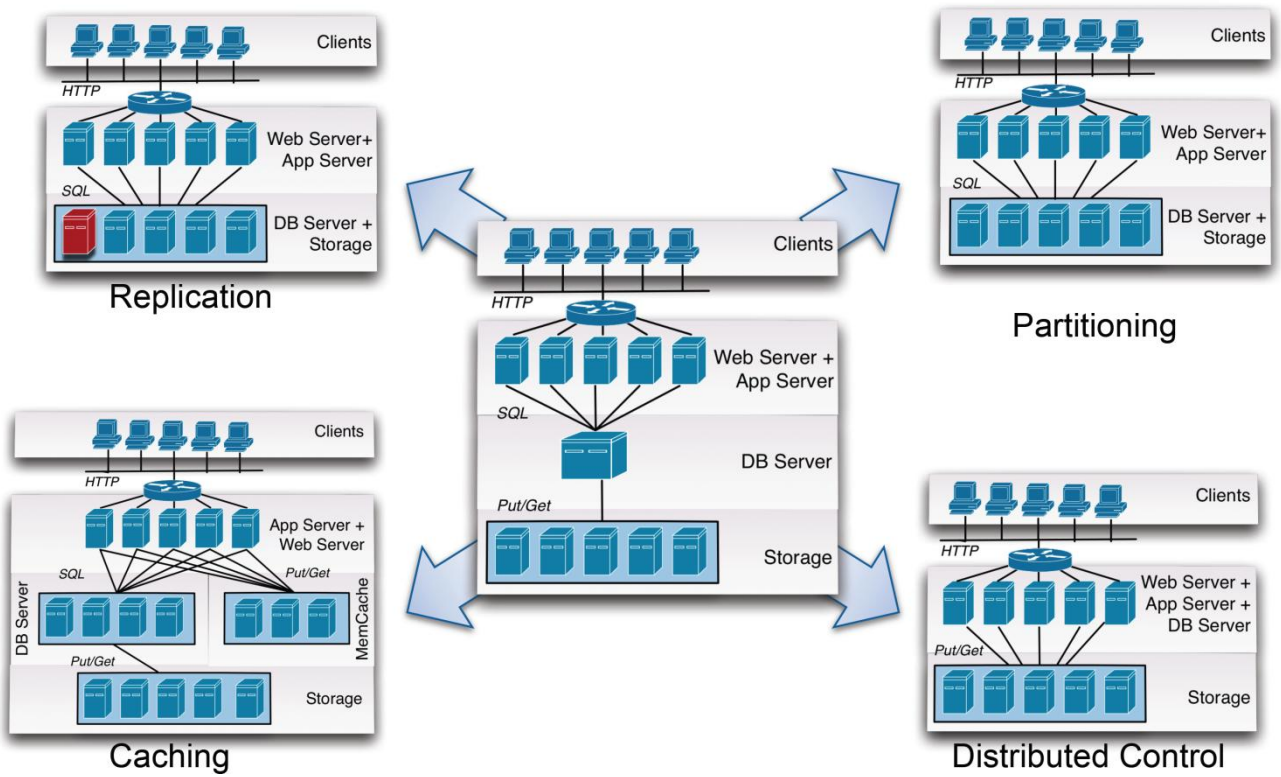
- Google
  - Google AppEngine has adopted a partitioning and replication architecture
  - Supports Memcache (caching)

- Microsoft
  - Microsoft's Azure adopts replication architecture



Cloud Architectures figure copied from [2]

- Classic : Starting point
- Partitioning
- Replication
- Distributed control
- Caching



Classic figure copied from [2]

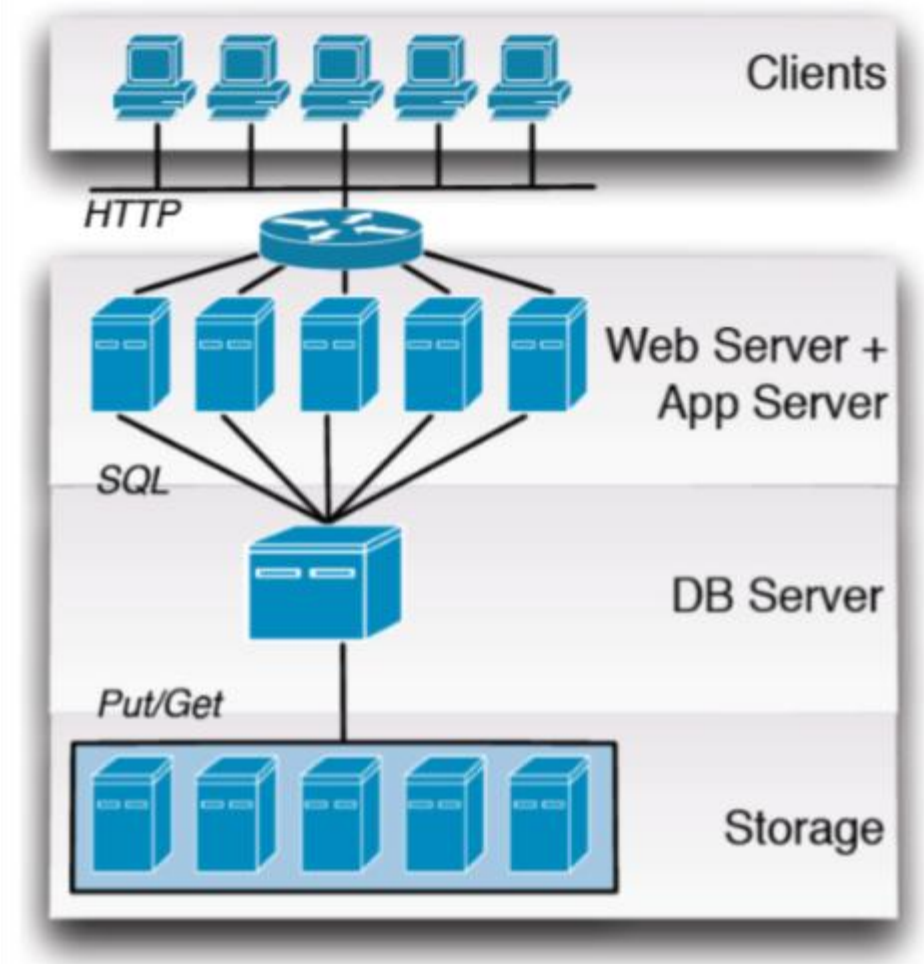
- Used as starting point

- Advantages:

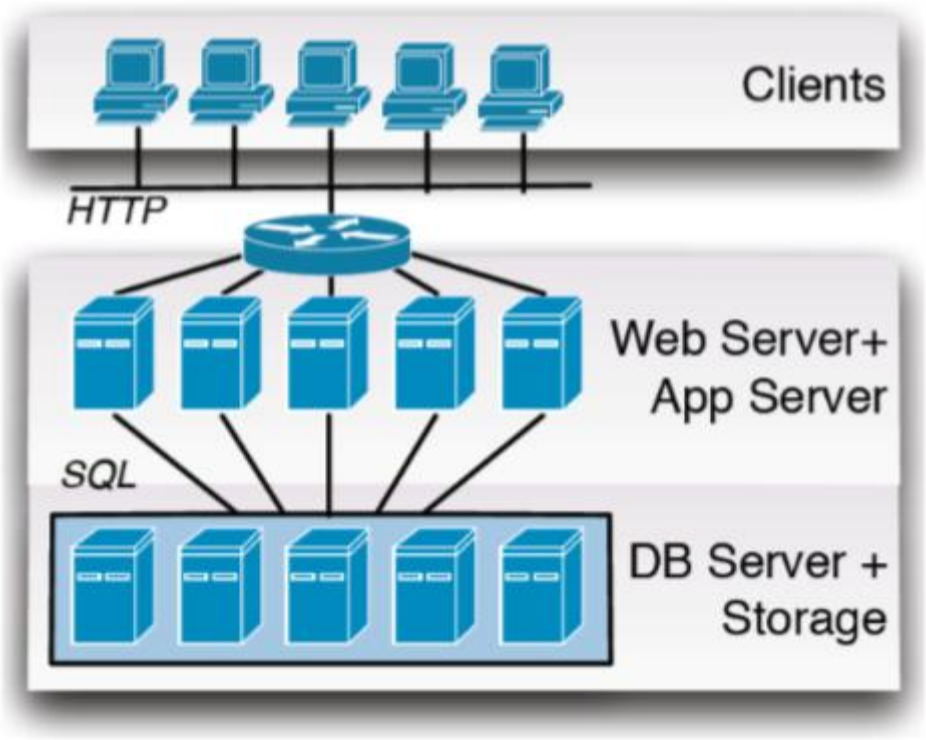
1. It allows to use "best-of-breed" components at all layers.

2. Allows scalability and elasticity at the storage and web/application server layers.

- e.g., AWS MySQL , AWS RDS



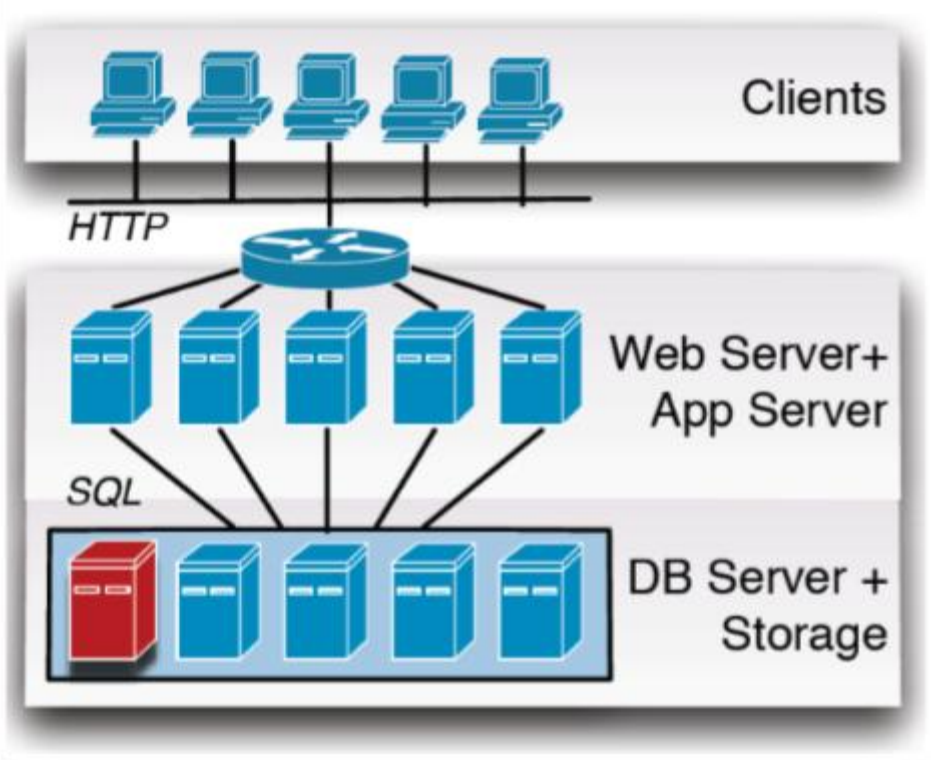
Partitioning figure copied from [2]



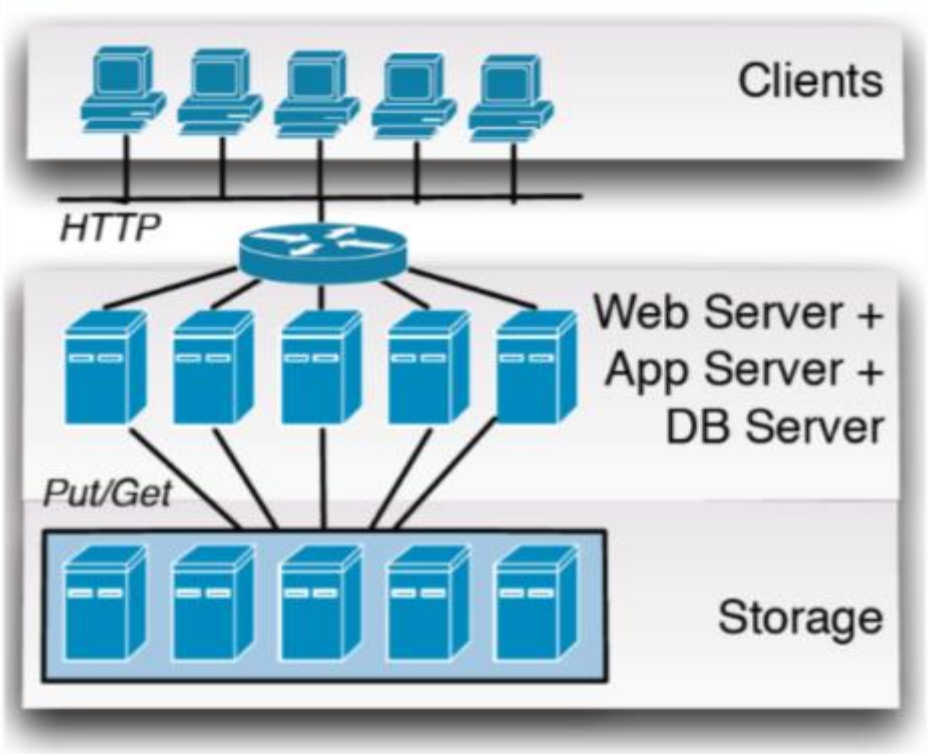
- The database is logically partitioned and each partition is controlled by a separate database server.
- e.g., combined with the replication model : AWS SimpleDB and Google AppEngine

Replication figure copied from [2]

- There are several database servers.
- Each database server controls a copy of the whole database (or partition of the database, if combined with partitioning).
- e.g., AWS MySQL/R



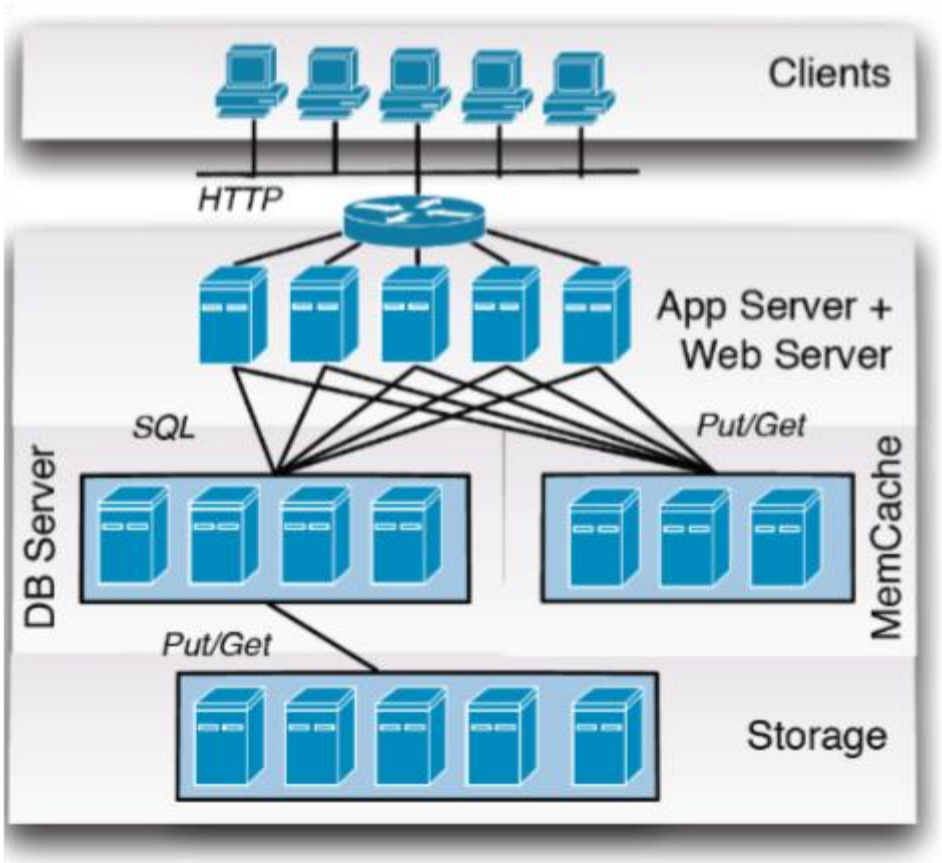
Distributed control figure copied from [2]



- The storage system is separated from the database servers
- The database servers access concurrently and autonomously the shared data from the storage system.
- e.g., AWS S3

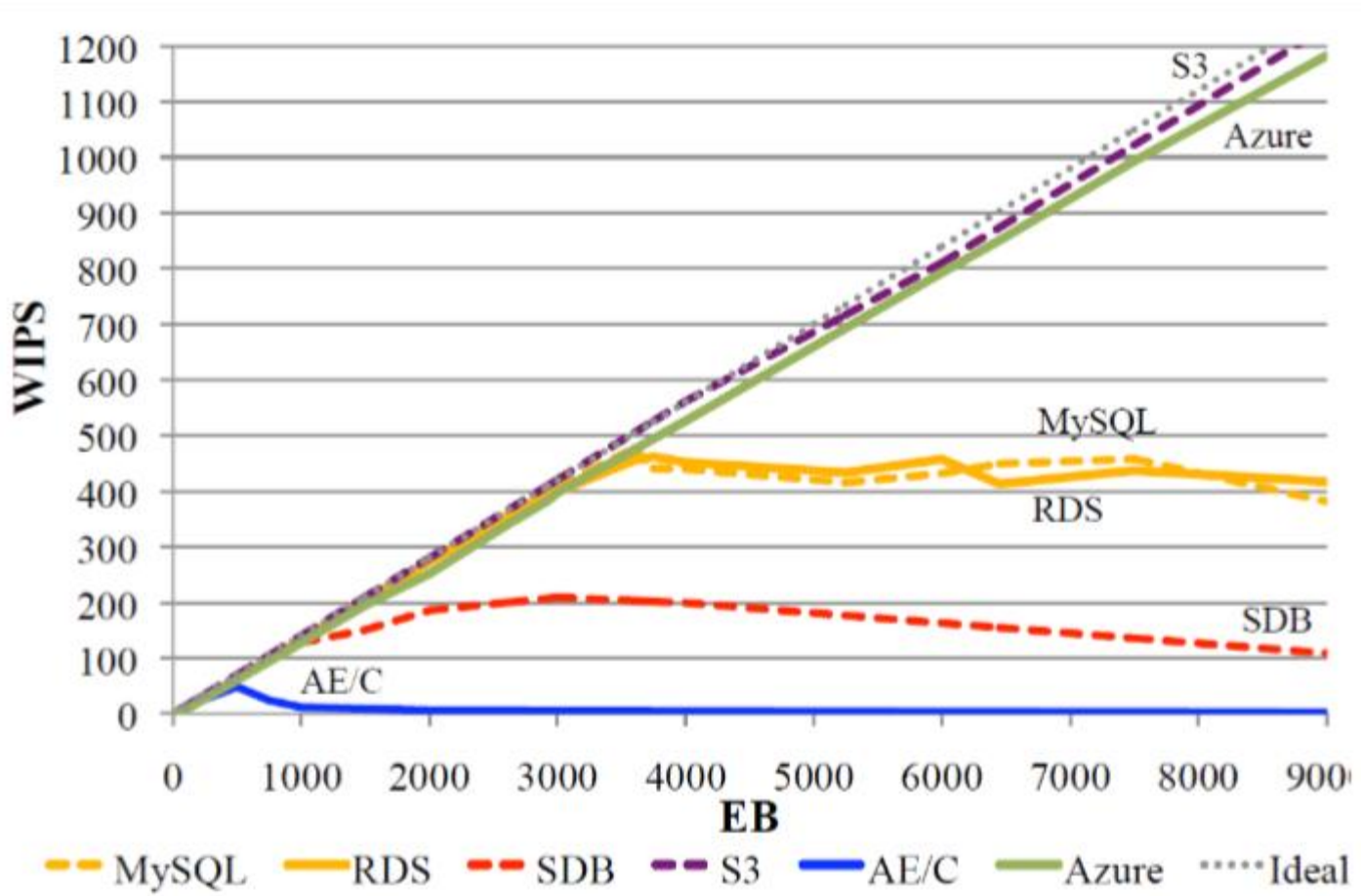
Caching figure copied from [2]

- The results of database queries are stored by dedicated cache servers.
- Correspondingly, the set of caching servers is typically referred to as *MemCache*
- e.g., Memcache, Google AppEngine supports Memcache



Comparison of Architectures [WIPS]

copied from [2]



- WIPS(EB): The throughput of valid requests per second on the number of EBs

Cost per WIPS [m$], Vary EB table copied from [2]

	1	10	100	500	1000
MySQL	0.635	0.072	***0.020***	***0.006***	***0.006***
MySQL/R	2.334	0.238	0.034	0.008	***0.006***
RDS	1.211	0.126	0.032	0.008	***0.006***
SimpleDB	0.384	0.073	0.042	0.039	0.037
S3	1.304	0.206	0.042	0.019	0.011
Google AE	0.002	0.028	0.033	0.042	0.176
Google AE/C	***0.002***	***0.018***	0.026	0.028	0.134
Azure	0.775	0.084	0.023	***0.006***	***0.006***

Consistency levels

- Snapshot Isolation SI
- Serializability S
- Repeatable read
- .
- .
- .
- Eventual Consistency EI
- Session Consistency SC

Strong consistency

Weak consistency

Tested Services copied from [2]

	MS Azure	Google App Eng	AWS RDS	AWS S3
Business Model	PaaS	PaaS	IaaS	IaaS
Architecture	**Replication**	**Part. + Repl. (+Dist. Control)**	**Classic**	**Distr. Control**
Consistency	**SI**	**≈ SI**	**Rep. Read**	**EC**
Cloud Provider	Microsoft	Google	Amazon	Flexible
Web/App Server	.Net Azure	AppEngine	Tomcat	Tomcat
Database	SQL Azure	DataStore	MySQL	--
Storage / FS	Simple DataStore	GFS	--	S3
App-Language	C#	Java/AppEngine	Java	Java
DB-Language	SQL	GQL	SQL	Low-Lev. API
HW Config.	Part. automatic	Automatic	Manual	Manual

Tested Services copied from [2]

	AWS SimpleDB	AWS MySQL	AWS MySQL/R
Business Model	IaaS	IaaS	IaaS
Architecture	**Partitioning + Replication**	**Classic**	**Replication**
Consistency	**EC**	**Rep. Read**	**Rep. Read**
Cloud Provider	Amazon	Flexible	Flexible
Web/App Server	Tomcat	Tomcat	Tomcat
Database	SimpleDB	MySQL	MySQL
Storage / File System	--	EBS	EBS
App-Language	Java	Java	Java
DB-Language	SimpleDB Queries	SQL	SQL
HW Config	Partly Automatic	Manual	Manual

CAP Theorem

Good scalability and strong consistency are hard to achieve due to CAP theorem

Outline

- Introduction and basics
- Different transaction models
- Consistency Rationing
- Conclusion

Motivation

- Web Shop

- The price of the items
- Data about the products sold
- Credit card information
- Data of customer profiles
- Records on user’s preferences
- Logging information

Consistency Rationing

- The point is that not all data need to be treated at the same level of consistency
- Transaction Cost vs. Inconsistency Cost
  - Not every thing is worth Gold [4]
- Consistency guarantees on the data instead at the transaction level
- Automatically switch consistency guarantees at runtime

Consistency Rationing

- Categorizing the data into A-B-C groups
- Applying different consistency strategies for each category

Categories

- Category A : e.g., Bank data, Atomic bomb, Credit card information, Price of the items
- Category B : e.g., Tickets, Product inventory (to some threshold)
- Category C : e.g., Recommendations, Data of customer profiles, Records on user's preferences, Logging information
- The tricky one is category B

Consistency level of each category

- Serializability for A , always up-to-date data
- Session consistency for C , still eventual consistent
- B is switching in between depending on some policy

Techniques to adapt B's consistency

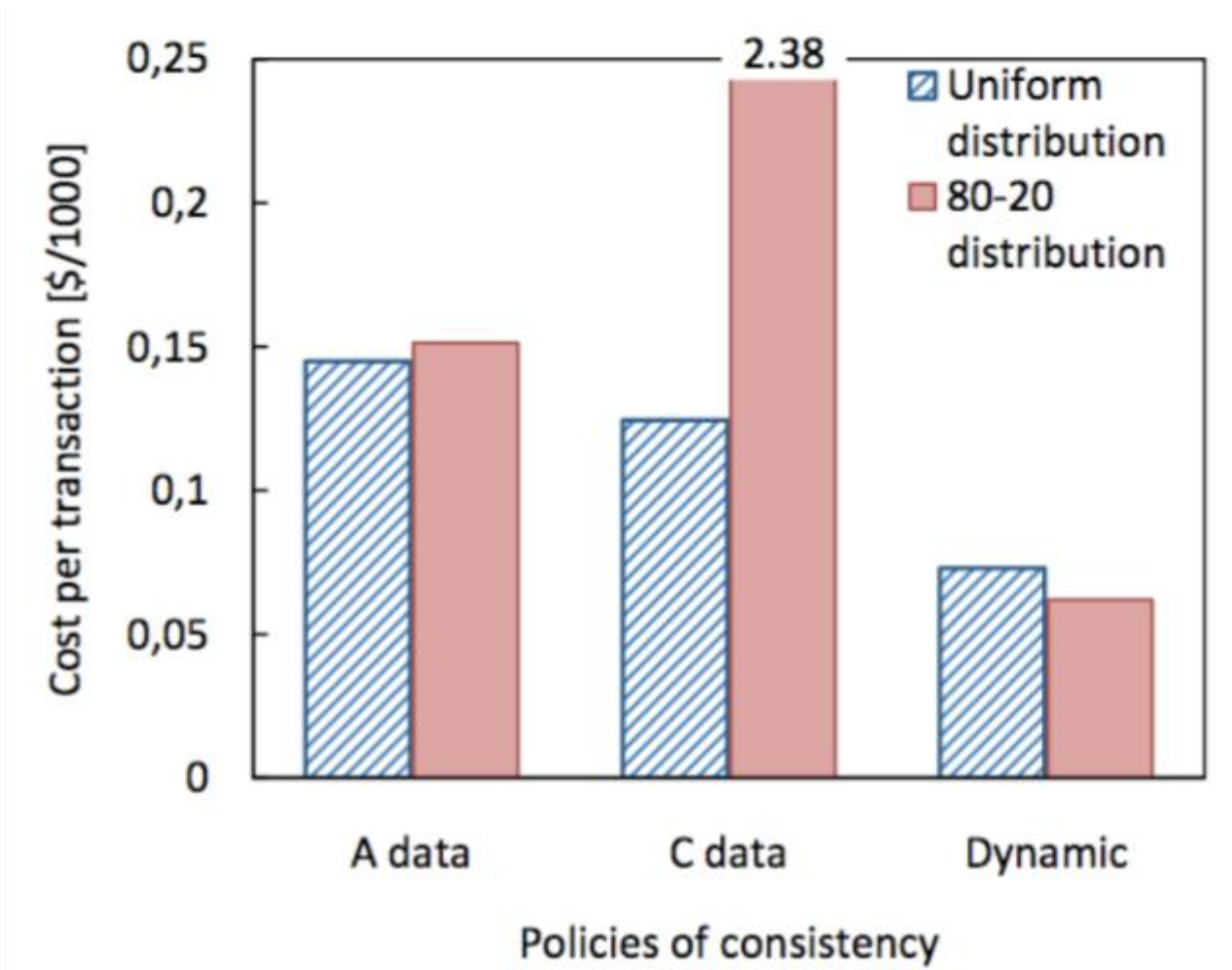
- General Policy : Based on conflict probability
- Time Policy
  - Time policy : Based on time stamp
- Numeric Policies e.g. Product inventory

  - Fixed threshold policy : Based on a certain threshold
  - Demarcation policy : Considers wider range of values
  - Dynamic policy : Extends the conflict model (general policy)on numeric types

Techniques factors copied from [4]

	Characteristics	Use Cases	Policies
General	Non-uniform conflict rates	Collaborative editing	General policy
Value Constraint	• Updates are commutative • A value constraint/ limit exists	• Web shop • Ticket reservation	• Fixed threshold policy • Demarcation policy • Dynamic policy
Time-Based	Consistency does not matter much until a certain moment in time	Auction systems	Time-based policy
Value-Based	Consistency requirements depend on the data value	Plane-ticket reservation (business vs. economy class)	Value-based policy

Cost per trx [$/1000] copied from [3]



Outline

- Introduction and basics
- Different transaction models
- Consistency Rationing
- Conclusion

Conclusion

- On Transaction in Cloud Computing, and the properties for a reliable transaction (ACID)
- Demonstrating different transaction models
- Vendors can implement different architectures having significant effects on performance[2]
- Introducing a new method to improve the consistency management, namely Consistency rationing.
- Rationing the consistency provides big performance and cost benefits[4]

References

[1]"An evaluation of alternative architectures for transaction processing in the cloud", D. Kossmann, T. Kraska, and S. Loesing, In SIGMOD '1.

[2]"An evaluation of alternative architectures for transaction processing in the cloud", Slides of authors

[3] "Consistency rationing in the cloud: pay only when it matters", T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, In VLDB '0. "

[4]"Consistency rationing in the cloud: pay only when it matters", Sildes of authors

[5]"Transaction Consistency in the Cloud: Old Paradigms Revisited",J.E.Armendariz-Ingo,M.I.Ruiz-Fuertes,2010.

[6]"Eventually Consistent", Werner Vogels, Communications of the ACM, Vol.52, No.1, Page: 40-44,January 2009.

Cost per trx [$/1000], Vary penalty cost **copied from [3]**