What is new in the cloud?
- A Database Perspective –

Donald Kossmann
Systems Group, ETH Zurich
http://systems.ethz.ch
Acknowledgments
Reference

  – [http://www.pubzone.org dblp journals dbsk KossmannK10](http://www.pubzone.org dblp journals dbsk KossmannK10)
  – (Published by Springer Open Access.)

• (Summary of several talks and papers of the DB community on cloud computing.)
Agenda

• Promises of Cloud Computing

• Benchmarking the State-of-the Art  [SIGMOD 2010]
  – Amazon, Google, Microsoft

• Towards a Silver Bullet  [ICDE 2010]
  – Asking the right questions
The Cloud 15 Years Ago

Browser

http

Internet

Browser

http

Service 1

DB 1

Service 2

DB 2
The Cloud Today

Client Machines

- Browser
- Adobe Air
- Adobe Flex
- Mobile
- Games
- ...

Internet

Servers of utility provider

- Service 1
- Service 2
- Service 3

Internal & External Data

http, udp, ...

REST (http)
Promises of Cloud Computing

• **Cost**
  – reduce cost
    • utilization, commoditization, optimization
    • prevention of failures, having failures
  – „pay as you go“ (cap-ex vs. op-ex)
  – easy to test!

• **Time to market**
  – avoid unnecessary steps
    • HW provisioning, purchasing, testing
  – difficult to test!
    • can only test scalability / elasticity
# Cow vs. Supermarket

<table>
<thead>
<tr>
<th>Cow</th>
<th>Supermarket: Bottle of Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big upfront investment</td>
<td>Small continuous expense</td>
</tr>
<tr>
<td>Produces more than you need</td>
<td>Buy what you need</td>
</tr>
<tr>
<td>Uses up resources</td>
<td>Less resources needed</td>
</tr>
<tr>
<td>Maintenance needed</td>
<td>No maintenance</td>
</tr>
<tr>
<td>Unpleasant waste product</td>
<td>Waste is not my problem</td>
</tr>
</tbody>
</table>

[Abadi 2009], [Kraska 2009]
Questions to ask

• Questions you ask your supermarket
  – How expensive is one litre of milk?
  – Is the price always the same?
  – What if I have a big party? Can you deliver?

• Questions you do not ask
  – How fast is the supermarket?
  – Good enough is good enough!
## What to optimize?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Traditional</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost [$]</td>
<td>fixed</td>
<td>optimize</td>
</tr>
<tr>
<td>Performance [tps, secs]</td>
<td>optimize</td>
<td>fixed</td>
</tr>
<tr>
<td>Scale-out [#cores]</td>
<td>optimize</td>
<td>fixed</td>
</tr>
<tr>
<td>Predictability [σ($)]</td>
<td>-</td>
<td>fixed</td>
</tr>
<tr>
<td>Consistency [%]</td>
<td>fixed</td>
<td>???</td>
</tr>
<tr>
<td>Flexibility [#variants]</td>
<td>-</td>
<td>optimize</td>
</tr>
</tbody>
</table>

*Put $ on the y-axis of your graphs!!!*  

[Florescu & Kossmann, SIGMOD Record 2009]
Agenda

• Promises of Cloud Computing

• **Benchmarking the State-of-the Art** [SIGMOD 2010]
  – Amazon, Google, Microsoft

• Towards a Silver Bullet [ICDE 2010]
  • Asking the right questions
What is out there?

• Amazon
  – IaaS: S3 and EC2
  – PaaS: SQS, SimpleDB, RDS

• Google App Engine
  – PaaS: Integrated DB, AppServer, and Cache

• Microsoft Azure

• Gazillions of start-ups (e.g., 28msec 😊)

• Research prototypes (e.g., Crescando 😊)
What is out there?

• Amazon
  – IaaS: S3 and EC2
  – PaaS: SQS, SimpleDB, RDS
• Google App Engine
  – PaaS: Integrated DB, AppServer, and Cache
• Microsoft Azure

• Gazillions of start-ups (e.g., 28msec 😞)
• Research prototypes (e.g., Crescando 😞)
Scope of this study

• **Workloads:** **Focus on OLTP**
  – OLAP under heavy debate by others
  – streaming not addressed yet (~ OLTP)
  – testing, archiving, etc. is boring

• **Types of clouds:** **Focus on public clouds**
  – believe that applicable to any kind of cloud
    • only difference: private clouds have planned downtime
  – cloud on the chip
  – swarms: ad-hoc private clouds

• **IaaS vs. PaaS vs. SaaS:** **Focus on PaaS**
## Tested Services

<table>
<thead>
<tr>
<th>Tested Services</th>
<th>MS Azure</th>
<th>Google App Eng</th>
<th>AWS RDS</th>
<th>AWS S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Model</td>
<td>PaaS</td>
<td>PaaS</td>
<td>PaaS</td>
<td>IaaS</td>
</tr>
<tr>
<td>Architecture</td>
<td>Replication</td>
<td>Part. + Repl. (+Dist. Control)</td>
<td>Classic</td>
<td>Distr. Control</td>
</tr>
<tr>
<td>Consistency</td>
<td>SI</td>
<td>≈ SI</td>
<td>Rep. Read</td>
<td>EC</td>
</tr>
<tr>
<td>Cloud Provider</td>
<td>Microsoft</td>
<td>Google</td>
<td>Amazon</td>
<td>Flexible</td>
</tr>
<tr>
<td>Web/App Server</td>
<td>.Net Azure</td>
<td>AppEngine</td>
<td>Tomcat</td>
<td>Tomcat</td>
</tr>
<tr>
<td>Database</td>
<td>SQL Azure</td>
<td>DataStore</td>
<td>MySQL</td>
<td>--</td>
</tr>
<tr>
<td>Storage / FS</td>
<td>Simple DataStore</td>
<td>GFS</td>
<td>--</td>
<td>S3</td>
</tr>
<tr>
<td>App-Language</td>
<td>C#</td>
<td>Java/AppEngine</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>DB-Language</td>
<td>SQL</td>
<td>GQL</td>
<td>SQL</td>
<td>Low-Lev. API</td>
</tr>
</tbody>
</table>
# Tested Services

<table>
<thead>
<tr>
<th></th>
<th>AWS SimpleDB</th>
<th>AWS MySQL</th>
<th>AWS MySQL/R</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business Model</strong></td>
<td>PaaS</td>
<td>IaaS</td>
<td>IaaS</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Replication</td>
<td>Classic</td>
<td>Replication</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>EC</td>
<td>Rep. Read</td>
<td>Rep. Read</td>
</tr>
<tr>
<td><strong>Cloud Provider</strong></td>
<td>Amazon</td>
<td>Flexible</td>
<td>Flexible</td>
</tr>
<tr>
<td><strong>Web/App Server</strong></td>
<td>Tomcat</td>
<td>Tomcat</td>
<td>Tomcat</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>SimpleDB</td>
<td>MySQL</td>
<td>MySQL</td>
</tr>
<tr>
<td><strong>Storage / File System</strong></td>
<td>--</td>
<td>EBS</td>
<td>EBS</td>
</tr>
<tr>
<td><strong>App-Language</strong></td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td><strong>DB-Language</strong></td>
<td>SimpleDB Queries</td>
<td>SQL</td>
<td>SQL</td>
</tr>
</tbody>
</table>
Benchmarking Cloud Services

• **Goal:** Test if cloud promises are fulfilled

• **Benchmarking Approach**  [DBTest09, SIGMOD10]
  – TPC-W Benchmark, Ordering Mix
  – Adapted for testing elasticity and cost ($)  
  – Vary load (EB): clicks per second

• **Benchmark metrics**
  – Cost: $ / WI
  – Cost Predictability: $\sigma(\$/WI)$
  – Throughput: WIPS
## Cost [m$/WI]

<table>
<thead>
<tr>
<th></th>
<th>EBs</th>
<th>Fully Utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>MySQL</td>
<td>0.635</td>
<td>0.072</td>
</tr>
<tr>
<td>MySQL/R</td>
<td>2.334</td>
<td>0.238</td>
</tr>
<tr>
<td>RDS</td>
<td>1.211</td>
<td>0.126</td>
</tr>
<tr>
<td>SimpleDB</td>
<td>0.384</td>
<td>0.073</td>
</tr>
<tr>
<td>S3</td>
<td>1.304</td>
<td>0.206</td>
</tr>
<tr>
<td>Google AE</td>
<td>0.002</td>
<td>0.028</td>
</tr>
<tr>
<td>Google AE/C</td>
<td>0.002</td>
<td>0.018</td>
</tr>
<tr>
<td>Azure</td>
<td>0.775</td>
<td>0.084</td>
</tr>
</tbody>
</table>
Cost predictability [m$/WI]

<table>
<thead>
<tr>
<th>Service</th>
<th>mean ± σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>0.015 ± 0.077</td>
</tr>
<tr>
<td>MySQL/R</td>
<td>0.043 ± 0.284</td>
</tr>
<tr>
<td>RDS</td>
<td>0.030 ± 0.154</td>
</tr>
<tr>
<td>SimpleDB</td>
<td>0.063 ± 0.089</td>
</tr>
<tr>
<td>S3</td>
<td>0.018 ± 0.098</td>
</tr>
<tr>
<td>Google AE</td>
<td>0.029 ± 0.016</td>
</tr>
<tr>
<td>Google AE/C</td>
<td>0.021 ± 0.011</td>
</tr>
<tr>
<td>MS Azure</td>
<td>0.010 ± 0.058</td>
</tr>
</tbody>
</table>

- **Ideal:** Cost / WI is constant
  - low σ better (ideal: σ=0)
  - cost independent of load
- **Google clear winner here!**
Relative Cost Factors

MySQL

Fixed CPU

Network

SimpleDB

Variable CPU

Network

Azure

Fixed CPU

Network

S3

Storage

Network

Fixed CPU

Variable CPU

Network

Legend:

- Network
- Fixed CPU
- Variable CPU
- Storage
Throughput [WIPS]

Emulated Browsers
- SimpleDB
- GoogleAE
- S3
- Azure
- RDS
- MySQL
- Ideal

WIPS

Emulated Browsers
Overload Behavior

[Graphs showing Overload Behavior for RDS, SimpleDB, and App Engine]
Agenda

• Promises of Cloud Computing

• Benchmarking the State-of-the Art [SIGMOD 2010]
  – Amazon, Google, Microsoft

• Towards a Silver Bullet [ICDE 2010]
  – Asking the right questions
Principles of Distributed Databases

• **Partitioning** [Ceri, Pelagatti 1984]
  – Large number of possible partitioning schemes
  – Repartitioning is very expensive

• **Replication, Caching** [Bernstein et al. 1987]
  – Replicating data increases fault-tolerance and read performance
  – Replication needs a mechanism to keep replicas consistent

• **Distributed Control** [Tanenbaum 2002]
  – Client-side consistency protocols

• **Combination of approaches difficult!** [Unterbrunner+ 2010]
Reference Architecture

Client

HTTP

Web Server

FCGI, ...

App Server

XML, JSON, HTML

DB Server

SQL

records

Store

get/put

block
Open Questions

• How to map stack to IaaS?
• How to implement store layer?
• What consistency model?
• What programming model?
• Whether and how to cache?
Open Questions

• **How to map stack to IaaS?**

  • How to implement store layer?
    – [VLDB 2009a]
  
  • What consistency model?
    – [VLDB 2009b]
  
  • What programming model?
    – [VLDB 2009c]
  
  • Whether and how to cache?
    – [SIGMOD 2010]
Variant I: Partition Workload by „Tenant“

- Client
  - HTTP
  - XML, JSON, HTML
  - Workload Splitter
    - Server-A
    - Server-B
  - XML, JSON, HTML
  - Block

- Client
  - HTTP
  - XML, JSON, HTML
  - Workload Splitter
    - Server-A
    - Server-B
  - XML, JSON, HTML
  - Block

- Store
  - get/put
  - block

- App Server
  - FCGI, ...
  - XML, JSON, HTML
  - Block

- DB Server
  - SQL
  - records

- Web Server
  - HTTP
  - XML, JSON, HTML
  - Block
Partition Workload by „Tenant“

- **Principle**
  - partition data by „tenant“
  - route request to DB of that tenant

- **Advantages**
  - reuse existing database stack (RDBMS)
  - flexibility to use DAS or SAN/NAS

- **Disadvantages**
  - multi-tenant problem [*Salesforce*]
    - optimization, migration, load balancing, fix cost
  - silos: need DB federator for inter-tenant requests
  - expensive HW and SW for high availability
If a shop is successful, you need to move it!

(popularity vs. growth of product assortment)
Variant II: Partition Workload by „Request“

HTTP

XML, JSON, HTML

FCGI, ...

XML, JSON, HTML

SQL

records

get/put

block

Store

Client

Web Server

App Server

DB Server

Store (e.g., S3)

Client

Workload Splitter

XML, JSON, HTML

Server-A

Server-B

Store (e.g., S3)
Metaphor: Internet Department Store

- If a product is successful, you stock up its supply
  - Transparent and fine-grained reprovisioning
  - Cost of reprovisioning much lower!!!
Partition Workload by „Request“

• **Principle**
  – fine-grained data partitioning by page or object
  – any server can handle any request
  – implement DBMS as a library (not server)

• **Advantages**
  – avoids disadvantages of Variant I

• **Disadvantages**
  – new synchronization problem
  – whole new breed of systems
  – caching not effective
Experiments Revisited: Cost / WI (m$)

<table>
<thead>
<tr>
<th>Variant</th>
<th>Low Load</th>
<th>Peak Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant 1 (RDS)</td>
<td>1.212</td>
<td>0.005</td>
</tr>
<tr>
<td>Variant 2 (S3)</td>
<td>-</td>
<td>0.007</td>
</tr>
<tr>
<td>Variant 2 (Google)</td>
<td>0.002</td>
<td>0.028</td>
</tr>
<tr>
<td>Variant 1 (Azure)</td>
<td>0.775</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Only V2 is cheap and predictable!
High performance can be achieved with both!

Warning: Still a hypothesis. S3 is only EC.
Open Questions

- How to map traditional DB stack to IaaS?
- **How to implement the storage layer?**
- What is the right consistency model?
- What is the right programming model?
- Whether and how to make use of caching?
Storage Layer Questions

- **What is the interface?**
  - get/put or more?
  - return records or blocks?

- **How to implement this interface?**
  - data structures
  - storage media and network
  - clustering of data
  - granularity of partitioning / sharding
API of Store

• Assumption: Hardware trends will change the game
  – most data in MM or SSD / PCM

• Proposition 1: return tuples rather than blocks
  – blocks are an arte-fact of old HW (i.e., disks)

• Proposition 2: push down predicates
  – because it can be done (power depends on impl.)
  – compensates for cost of remote storage
  – (old idea; e.g., iDisks)

• Open: consistency construct of store?
  – store might have nice properties (e.g., mon. Writes)
  – store might have nice primitives (e.g., counter, set/test)
Store Implementation Variants

• **DAS**
  – local disks with physically exclusive access
  – put/get interface; no synchronization
  – only works for V1

• **Key-value stores (e.g., Dynamo)**
  – DHTs with concurrent access
  – put/get interface; no synchronization
  – works for V1 and V2; makes more sense for V2

• **A DBMS (e.g., MySQL, SQL Server)**
  – have been misused before

• **ClockScan**  
  [Unterbrunner et al. 2009]
  – massively shared scans in a distributed system
Open Questions

• How to map traditional DB stack to IaaS?
• How to implement the storage layer?
• **What is the right consistency model?**
• What is the right programming model?
• Whether and how to make use of caching?
CAP Theorem

- **Three properties of distributed systems**
  - Consistency (ACID transactions w. serializability)
  - Availability (nobody is ever blocked)
  - Resilience to network Partitioning

- **Result**
  - it is trivial to achieve 2 out of 3
  - it is impossible to have all three

- **Highly controversial discussion** [Brantner 08, Abadi 10]
  - Levels of availability (always vs. never available)
  - What is the difference between CP and CA
What have people done?

• **Client-side Consistency Models** [Tannenbaum]
  – read & write monotonicity, session consistency, ...

• **Time-line consistency** [PNUTS08]
  – all writes are consistent (except across DC)
  – read consistency: either don’t care or latest version

• **New DB transaction models**
  – Escrow, Reservation Pattern [O’Neil 86], [Gawlick 09]
  – SAGAs and compensation; e.g., in BPEL [G.-Molina, Salem]
  – SAP, Amadeus et al. [Buck-Emden], [Kemper et al. 98]

• **Define logical unit of consistency** [Azure]
  – strong consistency within LuC
  – app-defined consistency across LuCs

• **Educate Application Developers** [Helland 2009]
What have people done?

• **Client-side Consistency Models** [Tannenbaum]
  – read & write monotonicity, session consistency, ...

• **Time-line consistency** [PNUTS08]

  - all writes are consistent (except across DC)
  - read consistency: either don’t care or latest version

• **New DB transaction models**
  – Escrow, Reservation Pattern [O’Neil 86], [Gawlick 09]
  – SAGAs and compensation; e.g., in BPEL [G.-Molina, Salem]
  – SAP, Amadeus et al. [Buck-Emden], [Kemper et al. 98]

• **Define logical unit of consistency** [Azure]
  – strong consistency within LuC
  – app-defined consistency across LuCs

• **Educate Application Developers** [Helland 2009]
What have we done?

• **Levels of Consistency, Cost Tradeoffs** [Brantner08]
  – can achieve read/write monotonicy + „A“ + „P“
  – the more consistency, the higher the cost

• **Economic models for consistency** [Amadeus], [Kraska09]
  – Classify the data as A, B, C
    • A data always strong consistent
    • C data always eventually consistent
    • B data handled like A or C data adaptively
  – Cost model that estimates business impact of inconst.
What have we done?

• Levels of Consistency, Cost Tradeoffs [Brantner08]
  – can achieve read/write monotonicy + „A“ + „P“

• Economic models for consistency [Amadeus], [Kraska09]
  – Classify the data as A, B, C
    • A data always strong consistent
    • C data always eventually consistent
    • B data handled like A or C data adaptively

  – Cost model that estimates business impact of inconst.

Follow tradition to push problem to app programmer!

But for a different reason!!!
## Cost per 1000 TAs [$] (TPC-W)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Total</th>
<th>Adjustable</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>0.15</td>
<td>0</td>
<td>0.15</td>
</tr>
<tr>
<td>Durability</td>
<td>1.8</td>
<td>1.1</td>
<td>0.7</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>2.1</td>
<td>1.4</td>
<td>0.7</td>
</tr>
<tr>
<td>Monotonicity+</td>
<td>2.9</td>
<td>2.6</td>
<td>0.3</td>
</tr>
<tr>
<td>Atomicity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{Brantner+08}\]
<table>
<thead>
<tr>
<th></th>
<th>Avg.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>11.3</td>
<td>12.1</td>
</tr>
<tr>
<td>Durability</td>
<td>4.0</td>
<td>5.9</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>4.0</td>
<td>6.8</td>
</tr>
<tr>
<td>Mono+Atomicity</td>
<td>2.8</td>
<td>4.6</td>
</tr>
</tbody>
</table>

[Brantner+08]
Open Questions

- How to map traditional DB stack to IaaS?
- How to implement the storage layer?
- What is the right consistency model?
- What is the right programming model?
- Whether and how to make use of caching?
Programming Model

• Properties of a programming lang. for the cloud
  – support DB-style + OO-style + CEP-style
  – avoid keeping state at app servers for V2

• Many languages will work in the cloud
  – SQL, XQuery, Ruby, .Net /LINQ, …;
  – J2EE will not work

• Open (research) questions
  – do OLAP on the OLTP data: My guess is yes!
  – rewrite your apps: My guess is yes!
Caching

- Many Variants Possible
  - this is just one
  - V1 caching mandatory
  - V2 caching prohibitive

- TPC-W Experiments
  - marginal improvements for Google AppEngine

- No low hanging fruit
Agenda

• Promises of Cloud Computing

• Benchmarking the State-of-the Art  [SIGMOD 2010]
  – Amazon, Google, Microsoft

• Towards a Silver Bullet  [ICDE 2010]
  – Asking the right questions
Case Study: Bets Made by 28msec

- How to map traditional DB stack to IaaS?
  - implemented both architectures (V1 + V2)
  - V1 only in a single server variant for low end

- How to implement the storage layer?
  - EBS for V1; KVS for V2

- What is the right consistency model?
  - ACID for V1; configurable for V2

- What is the right data + programming model?
  - XML & XQuery

- Whether and how to make use of caching?
  - No! (Only for code / precompiled query plans)
Conclusion & Future Work

- Hopefully started a new benchmark war
  - Tested elasticity, cost and cost predictability
  - Vendors interested in cost experiments
  - Academia interested in scalability experiments
  - More bits available: http://www.pubzone.org

- Find silver bullet for data management in cloud
  - Develop a reference architecture
  - Combine partitioning + replication + distributed control
  - ETH Systems Group: http://www.systems.ethz.ch