An Evaluation of Alternative Architectures for Transaction Processing in the Cloud

Donald Kossmann, Tim Kraska*, Simon Loesing

* now at UC Berkeley
Systems Group @ ETH Zurich

- Joint initiative of 4 professors, 6 PostDocs, ≈30 PhD students
- Mission: Redefining the „systems stack“ according to new app requirements and technology trends
  - Increasing scale, concurrency/parallelism at all levels, heterogeneity (HW, SW & data), diversity (ideas from other fields),...
- Research Approach
  - Build real systems (Barrelfish, Cloudy, Crescando...)
  - Collaboration within group (from HW to app)
  - Working closely with industry (e.g., ECC)
  - Rethinking and creating new ideas (e.g., merge DB & app server)
Cloud Computing Today

- The Cloud Computing era promises
  - Scalability
  - Fault-tolerance
  - Pay-as-you-go

- All big players and more and more startups offer cloud products

- Wide range of services offered
  - Infrastructure services (e.g., data storage, virtual machines...)
  - Platform services (e.g., Web-application hosting, map-reduce...)

- Resulting in a jungle of services
  - Divergent properties and guarantees
  - Different internal architectures
  - Often not compatible
  - ...
Lost in the Cloud?
Motivation

- We are trying to bring light into the dust!

Questions:

- What are the building blocks to fulfill these promises?
- Do current offerings really fulfill the cloud promises?
- How do offerings compare in terms of scalability and cost?
A Peek Ahead...

Throughput

WIPS

Emulated Browsers
Traditional Database Architecture

- The Database-Tier limits application scalability!

Related Work: Distributed Databases

- Database is distributed across multiple physical locations

- Common techniques:
  - Partitioning
    - Large number of possible partitioning schemes [1]
    - Repartitioning is very expensive
  - Replication
    - Replicating data increases fault-tolerance and read performance
    - Replication needs a mechanism to keep replicas consistent [2]

Cloud Architectures

Replication

Partitioning

Caching

Distributed Control
### Tested Services

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

9th June 2010

Systems Group | Department of Computer Science | ETH Zurich
## 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>Business Model</td>
<td>IaaS</td>
<td>IaaS</td>
<td>IaaS</td>
</tr>
<tr>
<td>Architecture</td>
<td>Partitioning + Replication</td>
<td>Classic</td>
<td>Replication</td>
</tr>
<tr>
<td>Cloud Provider</td>
<td>Amazon</td>
<td>Flexible</td>
<td>Flexible</td>
</tr>
<tr>
<td>Web/App Server</td>
<td>Tomcat</td>
<td>Tomcat</td>
<td>Tomcat</td>
</tr>
<tr>
<td>Database</td>
<td>SimpleDB</td>
<td>MySQL</td>
<td>MySQL</td>
</tr>
<tr>
<td>Storage / File System</td>
<td>--</td>
<td>EBS</td>
<td>EBS</td>
</tr>
<tr>
<td>App-Language</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>DB-Language</td>
<td>SimpleDB Queries</td>
<td>SQL</td>
<td>SQL</td>
</tr>
</tbody>
</table>
Trade-Offs you have to make

- **Consistency**
  - eventual consistency vs. ACID

- **Query language support**
  - key/value vs. SQL light vs. full SQL

- **Scaling**
  - automatic vs. manual
  - (web/app-tier and db-tier)

- **Patches/Maintenance**
  - automatic vs. manual
  - (web/app-tier and db-tier)

- **Language flexibility**
  - pre-defined vs. fully flexible

- **Machine access**
  - none vs. partial vs. full

- **Price-plans**
  - free quotas, fixed costs,…

- **Data Models**
  - key/value, semi-structured, relational

- …
Benchmarking Cloud Services

- **Goal**: Do the services fulfill their promises?
- **Benchmarking approach**
  - TPC-W Benchmark, Ordering Mix
  - Adapted to fit the cloud requirements [DBTest 2009 paper]
- **New main metrics**:  
  - **Throughput**
    - Max Web Interactions Per Second (WIPS) in response time
    - Increase load over time from 1 to 9000EB (1EB every 0.4 sec)
    - 1 EB ~ 500 requests per hour, 9000EB ~ 1250 requests per second
  - **Cost**
    - Cost / WI ($): Divide costs of the running system by the current load
    - CostPerDay ($): The (projected) total cost of running the benchmark 24 hours
  - **Cost predictability**
    - s($) – Cost standard deviation from 1 EB to 9000EB
Benchmark Setup

Remote Browser Emulator (RBE)

VM
VM
VM
VM
VM
VM
VM

System Under Test (SUT)

.Net
App Engine
Data Store

MS SQL Azure

MySQL
RDS
EBS

MySQL R
Simple DB
S3

DataCenter B

Manually mgt, manually scaled
Provider mgt, manually scaled
Provider mgt, provider scaled

DataCenter A

9th June 2010
Systems Group | Department of Computer Science | ETH Zurich
A Peek Ahead...

Throughput

![Graph showing throughput over emulated browsers and WIPS](image)
Throughput [WIPS] – Strongly Consistent Services

The diagram illustrates the throughput in WIPS (Work In Progress) for various cloud services as a function of the number of emulated browsers. The services compared are Azure, MySQL, RDS, and GAE/C.

- **Azure** shows a strong linear increase with the number of emulated browsers.
- **MySQL** and **RDS** demonstrate less linear behavior, with RDS slightly outperforming MySQL.
- **GAE/C** remains consistent and near the baseline, indicating lower throughput.

The x-axis represents the number of emulated browsers, while the y-axis represents the WIPS.
Throughput [WIPS] – Weakly Consistent Services

Library guarantees atomicity and commutative updates, but no isolation (e.g., integrity constraints might be violated). Here fully consistent according to the TPC-W requirements.

No atomicity, possibility of lost updates (i.e., product stock updates)
Throughput [WIPS]

![Graph showing throughput comparison between systems]

- **S3**
- **Azure**
- **MySQL**
- **RDS**
- **GAE/C**

Emulated Browsers

WIPS
Scalability Azure (first few runs)
Scalability Azure (final results)
Throughput [WIPS]

Emulated Browsers

WIPS

S3
Azure
MySQL
RDS
SDB
GAE/C
Overload Behavior

RDS

SimpleDB

App Engine

- WIPS Issued
- WIPS in RT
- Ideal
## 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>0.064</td>
</tr>
</tbody>
</table>
Cost predictability [m$/WI]

<table>
<thead>
<tr>
<th></th>
<th>mean ± s</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>Azure</td>
<td>0.010 ± 0.058</td>
</tr>
</tbody>
</table>

- In an optimal scenario the Cost / WI is constant and independent of the load
- s: the lower the better
- Google’s price model fits best the pay-as-you-go paradigm!
Relative Cost Factors

MySQL

Azure

SimpleDB

S3

Network

Fixed CPU

Variable CPU

Storage
Questions/Comments we got…

- You are using Google AppEngine not correctly! Google’s TRX model is like getting a lock on the whole DB (i.e., entity group)
- Why do you not partition the data for GAE and SimpleDB? It would make them scale
- Why does Azure scale so well? Will it continue to scale?
- This is not a fair comparison (because of consistency, functionality, purposes of the architecture,…!)
- Why stop at 9000 Ebs?
Questions/Comments we got… (ctd)

- Which platform was the easiest to program?
- The data size is too small. Why not use a bigger data size?
- MS Azure is not better than SimpleDB if you increase the data size.
- TPC-W is not the right choice. For a different workload it would look different.
- Can you include product X in your study?
- Is the source code available?
Personal remarks on SQL Azure...

- Really easy-to-use platform
- SQL Azure gives the comfort of a traditional DB and the fault-tolerance of a cloud service
- However, strong limitations in the data size and scalability and the pricing model makes it best suited for medium-sized deployments (i.e., for small applications to expensive, for really large applications not scalable)
- What we would like to see: Automatic scaling of VMs, Java support, low-level access,...
Conclusion & Future Work

- Have we started a new benchmark war?
  - We have measured elasticity, cost and cost predictability
  - Vendors implement different architectures having significant effects on performance
  - Discussion available on http://www.pubzone.org

- Now we want to find the silver bullet for data management in the cloud
  - Develop a reference architecture
  - Combine techniques of partitioning + replication + distributed control
  - http://www.systems.ethz.ch/research/projects/projectslist#Cloudy
AMP Lab: The Next Generation
Algorithms, Machines and People

- **Observation**: We can extract only a small fraction of the value from the wealth of data available.
- **Mission**: Enable many people to collaborate to collect, generate, clean, make sense of and utilize lots of data.
- **Approach**: A holistic view of the stack from data visualization down to cluster & multi-core support.
- A five year plan; will dovetail with RADLab completion

For more information – catch me later or send mail: franklin@cs.berkeley.edu / kraska@cs.erkely.edu