Head in the clouds:
an overview of cloud computing and
some associated research challenges

Rodrigo Rodrigues

INESC-ID DLS – January 24, 2014
2012 – Email service @ NOVA
Google’s data center

http://www.google.com/about/datacenters/gallery/
Distinguishing features when compared to traditional distributed systems

(and an overview of associated research problems)
(#1) Trust relationships

5 reasons enterprises are frightened of the cloud

1. Losing control

2. Security

3. Data Protection

Closely tied in with security, enterprises are concerned about data protection. Many governments place strict data protection requirements on large companies and standards audit schemes such as ISO-9001 place additional restrictions on firms.
Excalibur: Trusted computing abstractions for the cloud

Joint work with Nuno Santos, Krishna Gummadi, and Stefan Saroiu

• Customer specifies policies over which configurations handle data

• Trust model:
  – Software administrator → adversarial role
  – Cloud provider → trusted to deploy defenses (but no control over all administrators)
  – Platform developer → mitigation role

• Leverage commodity TPMs to enforce policies
Policy-sealed data

- TPM attestation not designed for cloud
  - inefficient and exposes low-level info
- New abstraction: policy-sealed data
  - Cryptographically seal (⭐️) data to policy
  - Can only be unsealed (⭐️) by nodes obeying policy
Building Policy-Sealed Data

• New data center component: Monitor
  – Attested by clients to bootstrap trust
  – Attests remaining system nodes

• New distributed cryptographic protocols
  – Leverage attribute-based encryption

• Example VM rental service:
#2) The data center as the new computer

- New instruction set for “big data” computations
  - First “instruction”: MapReduce

http://www.google.com/about/datacenters/gallery/
MapReduce overview

- Observation: data sets evolve slowly
  - MapReduce not designed to handle small update
Incremental MapReduce

Joint work with Pramod Bhatotia, Alexander Wieder, Umut A. Acar, Rafael Pasquini

- Percolate changes through dependence graph
- Inspired by PL research

Read input
Map tasks
Reduce tasks
Write output
Controlling Reduce Granularity

• Leverage combiners: pre-processing of Reduce
  – Meant to be co-located with Map task, save bw
• Use them to break up Reduce work
Experimental Results

- Significant speedups for incremental runs
- Modest overhead for first run
(#3) Geo-replication: Making systems fast when possible, consistent if necessary

Joint work with Cheng Li, Daniel Porto, Allen Clement, Nuno Preguiça and Johannes Gehrke

NOVA LINCS, MPI for Software Systems, Cornell Univ.
Higher latency ⇒ Fewer clicks

Microsoft Bing: 2-sec slowdown reduced number of users that perform any clicks on the page by 4.4%.

[source: E. Schurman and J. Brutlag, “Performance Related Changes and their User Impact”. Talk at Velocity ‘09]
[same source for next set of graphs]
Higher latency ⇒ Less engagement

Bing: 2-sec slowdown added more than 3 seconds in the time to first click.
Google observed similar trends

Similar impact in number of daily searches per user at Google.
Users get increasingly annoyed over time

- Google experiment ran for 6 weeks, split results into first and second 3-week period
  - Saw a degradation in the number of searches
Difficult for user behavior to recover

- After 6 weeks, delays were removed
  - Changes in user behavior persisted
Higher latency $\Rightarrow$ Lower revenue

Bing: 2-sec slowdown reduced revenue per user by 4.3%.
Consequence: Geo-replication
Flip-side of geo-replication

- Geographically disperse replicas are expensive to synchronize
- Leads to tension between latency and consistency
Handling geo-replication: current practice and limitations

• Amazon: Eventual consistency
  – Assumes a seamless merge strategy and may allow undesirable behaviors
  – Folklore: Eventual consistency is no consistency

• Facebook + PNUTS: single master, read-only mirror replicas
  – Works well when there is a single updater
  – Not the case, e.g., in social networking services
An observation and a challenge

• Eventual consistency works most of the time, but need some strongly consistent operations
• Must let both weakly and strongly consistent operations co-exist [LazyReplication:PODC90, Walter:SOSP11]
  — But which level of consistency to use for an operation?

Need to find principled ways to build systems that are **fast if possible, consistent when needed**
Outline

• Mixing strong and eventual consistency in a single system
• Transforming applications to increase coverage of eventual consistency
• Deciding when is it safe to use eventual consistency
• Implementation and evaluation
Balance strong/eventual consistency

Strong consistency  Eventual (causal) consistency

Allowed output ⇔ execute after preceding operations
Balance strong/eventual consistency

Strong consistency  Eventual (causal) consistency

R1 → R2 → R3

A1 → A2 → B1 → B2 → B3
Balance strong/eventual consistency

- Low latency of eventual consistency when possible
- Coordination for strong consistency only when necessary
A RedBlue consistent bank system
A RedBlue consistent bank system

- **Problem**: Different execution orders lead to divergent state.
- **Cause**: `accrueinterest` doesn’t commute with `deposit`.
- **Implication**: Convergence requires Red, but Red is slow.
Outline

• Mixing strong and eventual consistency in a single system

• Transforming applications to increase coverage of eventual consistency

• Deciding when is it safe to use eventual consistency

• Implementation and evaluation
Problem of replicating operations

Initial: balance = 100, interest = 0.05

Alice in EU

100

↓

120

↓

126

Bob in US

100

↓

accrueinterest(): +5

125

deposit(20): +20

accrueinterest(): +6

Side-effects vary depending on the state the operation observed
Generator/Shadow operation

• Intuitively, the execution of *accrueinterest* can be divided into:
  – A generator operation [executed **once at primary site**]
    • decides *how much interest to be accrued*
    • has **no side effects**
  – One or more shadow operations [executed **at all sites**]
    • adds *the decided interest to the balance*

• Formally:
  – Split operation \( u \) into:
    • Generator \( g_u \), with no side effects, executed against state \( S \)
    • Shadow \( h_u(S) \), executed against current state \( S_i \) at each site \( i \)
  – Correctness of shadow operations: \( S + h_u(S) = S + u \)
Bank generator/shadow operations

Original/Generator operation

```plaintext
deposit(float m){
    balance = balance + m;
}

accrueinterest(){
    float delta=balance × interest;
    balance=balance + delta;
}

withdraw(float m){
    if(balance-m>=0)
        balance=balance - m;
    else
        print “Error”
}
```

Shadow operation

```plaintext
deposit’(float m){
    balance = balance + m;
}

accrueinterest’(float delta){
    balance=balance + delta;
}

withdrawAck’(float m){
    balance=balance - m;
}

withdrawFail’(){
```
Bank generator/shadow operations

Original/Generator operation

```c
void deposit(float m){
    balance = balance + m;
}
```

Shadow operation

```c
void deposit'(float m){
    balance = balance + m;
}
```

All four shadow banking operations commute with each other!
Fast and consistent bank

**Initial**: balance = 100, interest = 0.05

**Alice in EU**
- Initial balance: 100
- Deposit: 20
- New balance: 120
- Accrue interest: 5
- New balance: 125

**Bob in US**
- Initial balance: 100
- Deposit: 20
- New balance: 120
- Accrue interest: 5
- New balance: 125

**Shadow op**
- From Bob to Alice
- From Alice to Bob
Not so fast ...

Initial: balance = 100, interest = 0.05

Alice in EU

100
+20
120
+5
125
withdraw(100): -100

Bob in US

100
+5
105
+20
125
withdraw(80): -80
Not so fast ...

- **Problem**: Different execution orders lead to a negative balance.

- **Cause**: Blue operations that potentially break invariants execute without coordination.

- **Implication**: We must label successful withdrawal (withdrawAck’) as Red.
Outline

• Mixing strong and eventual consistency in a single system

• Transforming applications to increase coverage of eventual consistency

• Deciding when is it safe to use eventual consistency

• Implementation and evaluation
Determining conditions for labeling

• **Goal #1: Eventual convergence**
  – If all replicas execute same set of operations then they reach the same state
  – Otherwise a quiescent system would return different system views depending on site

• **Theorem:** if all Red shadow operations commute with all other shadow operations, then eventual convergence is reached
Determining conditions for labeling (cont.)

• Goal #2: Invariant preservation
  – E.g., balance never goes below zero
  – Store does not sell more than stock
  – Auction only declares a single winner

• Problem is that operations can break invariants if unaware of each other
Determining conditions for labeling (cont.)

• Definition 1: valid state, one that meets operation invariants

• Definition 2: Given any two valid states S, S’, shadow operation $h_u(S)$ is invariant-preserving if applying $h_u(S)$ against state $S’$ leads to a valid state.

• Theorem: if all Red operations are invariant-preserving, then no invariants are violated
Which must be **Red** or can be **Blue**?

- Ensuring convergence
- Ensuring invariant preservation

- A shadow operation $u$
  - Commutes with all others?
    - Yes
    - No

- Invariant-preserving?
  - Yes
  - No

- **Red**
  - No
  - Yes **Blue**
Implementation
Gemini coordination system

Cross-site communication
Evaluation
Questions

• How common are Blue operations?

• Does RedBlue consistency improve user-observed latency?

• Does throughput scale with the number of sites?
Questions

- How common are Blue operations?

- Does RedBlue consistency improve user-observed latency?

- Does throughput scale with the number of sites?
Case studies

• Applications:
  – Two e-commerce benchmarks: TPC-W, RUBiS
  – One social networking app: Quoddy

<table>
<thead>
<tr>
<th>Apps</th>
<th># Original update txns</th>
<th># Blue/Red update ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-W</td>
<td>7</td>
<td>0/7</td>
</tr>
<tr>
<td>RUBiS</td>
<td>5</td>
<td>0/5</td>
</tr>
<tr>
<td>Quoddy</td>
<td>4</td>
<td>0/4</td>
</tr>
</tbody>
</table>
Case studies

• Applications:
  – Two e-commerce benchmarks: TPC-W, RUBiS
  – One social networking app: Quoddy

<table>
<thead>
<tr>
<th>Apps</th>
<th># Original update ops</th>
<th># Blue/Red update ops</th>
<th># Shadow ops</th>
<th># Blue/Red update ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-W</td>
<td>7</td>
<td>0/7</td>
<td>16</td>
<td>14/2</td>
</tr>
<tr>
<td>RUBiS</td>
<td>5</td>
<td>0/5</td>
<td>9</td>
<td>7/2</td>
</tr>
<tr>
<td>Quoddy</td>
<td>4</td>
<td>0/4</td>
<td>4</td>
<td>4/0</td>
</tr>
</tbody>
</table>
How common are Blue operations?

Runtime Blue/Red ratio in different applications with different workloads:

<table>
<thead>
<tr>
<th>Apps</th>
<th>workload</th>
<th>Originally</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blue (%)</td>
</tr>
<tr>
<td>TPC-W</td>
<td>Browsing mix</td>
<td>96.0</td>
</tr>
<tr>
<td></td>
<td>Shopping mix</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>Ordering mix</td>
<td>63.0</td>
</tr>
<tr>
<td>RUBiS</td>
<td>Bidding mix</td>
<td>85.0</td>
</tr>
<tr>
<td>Quoddy</td>
<td>a mix with 15% update</td>
<td>85.0</td>
</tr>
</tbody>
</table>
How common are **Blue** operations?

Runtime **Blue/Red** ratio in different applications with different workloads:

<table>
<thead>
<tr>
<th>Apps</th>
<th>workload</th>
<th>Originally</th>
<th>With shadow ops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blue (%)</td>
<td>Red (%)</td>
</tr>
<tr>
<td>TPC-W</td>
<td>Browsing mix</td>
<td>96.0</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>Shopping mix</td>
<td>85.0</td>
<td>15.0</td>
</tr>
<tr>
<td></td>
<td>Ordering mix</td>
<td>63.0</td>
<td>37.0</td>
</tr>
<tr>
<td>RUBiS</td>
<td>Bidding mix</td>
<td>85.0</td>
<td>15.0</td>
</tr>
<tr>
<td>Quoddy</td>
<td>a mix with 15% update</td>
<td>85.0</td>
<td>15.0</td>
</tr>
</tbody>
</table>

The vast majority of operations are **Blue**.
Questions

• How common are Blue operations?

• Does RedBlue consistency improve user-observed latency?

• Does throughput scale with the number of sites?
Experimental setup

• Experiments with:
  – TPC-W, RUBiS and Quoddy

• Deployment in Amazon EC2
  – spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  – locating users in all five sites and directing their requests to closest server
Experimental setup

- Experiments with:
  - TPC-W, RUBiS and Quoddy

- Deployment in Amazon EC2
  - spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  - locating users in all five sites and directing their requests to closest server
Does RedBlue consistency improve user-observed latency?

Average latency for users at all five sites
Does throughput scale with the number of sites?

![Bar chart showing peak throughput for different deployments.

- 1-site Original: 400 requests/s
- 1-site Gemini: 800 requests/s
- 2-site Gemini: 1200 requests/s
- 3-site Gemini: 1600 requests/s
- 4-site Gemini: 2000 requests/s
- 5-site Gemini: 2400 requests/s

The data suggests that throughput increases with the number of sites, indicating scalability.](image-url)
Conclusion

• Cloud computing brings several new challenges in distributed systems research

• RedBlue consistency allows strong consistency and eventual consistency to coexist.

• Generator/shadow operation extends the space of fast operations.

• A precise labeling methodology allows for systems to be fast and behave as expected.

• Experimental results show our solution improves both latency and throughput.
Head in the clouds: an overview of cloud computing and some associated research challenges

Part of the slides from the talk given at:
10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)
Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguica, and Rodrigo Rodrigues.
Excalibur figures by Nuno Santos

THANK YOU!