Provenance for the Cloud

Kiran-Kumar Muniswamy-Reddy,
Peter Macko, and Margo Seltzer
Cloud Stores

- Becoming increasingly important
  - Backups
  - Host shared scientific data
  - Store web application data
  - Serve web pages

- However, not designed to store provenance
What is Provenance?

- Meta-data describing the history of an object
  - What objects does this object depend on?
  - What applications modified/generated this object?
What is Provenance? (2)

Climate Prediction Data

$\text{WG}_1$

$\text{WG}_n$

depends on $\text{WG}_n$

depends on $\text{WG}_n$

depends on $P$

Provenance forms a DAG

Aggregate $P$

Report
Why cloud provenance?

- Provides information regarding structure of data and applications
  - Validate Data sets
  - Identify how data spread through the system
  - Search [Shah-Usenix’07]
  - Generate data on-demand [Adams-HotCloud’09]
Goal

- Provenance is vital
- How do we store provenance given today’s cloud offerings?
Outline

- Introduction
- **Design Issues**
- Protocols
- Evaluation
- Conclusions
Setting

- Provenance-Aware Storage system (PASS) tracks and collects provenance
  - Observes system calls that applications make and infers relationships between objects
  - Designed for local file system/NFS backend
- Modified it to use AWS services as backend
Properties

- Provenance-data coupling
- Multi-object ordering
- Data-independent persistence
- Efficient query
Provenance-data coupling

- Provenance accurately describes the data object
- Data must be what is described by provenance
  - Can mislead users if violated
  - Detection, if not coupling

![Diagram showing the relationship between P_2 and the report generated by P_2]
Multi-object Ordering

- The provenance and data of an ancestor object must be recorded in the provenance system
  - No dangling references
Multi-object Ordering(2)

QC+Aggr

WG$_1$

AGgregate P

WG$_n$

Activist$_1$

Activist$_2$

Report
Data Independent Persistence

- Cannot always delete provenance when object is deleted
  - Can disconnect the provenance DAG
Data Independent Persistence (2)

Deleting $\text{WG}_1$’s provenance with $\text{WG}_1$ will disconnect DAG
Efficient Query

- Provenance must be accessible to users who want to verify properties of their data or simply be aware of its lineage
  - If provenance is not readily accessible, the provenance is of questionable value.
Efficient Query (2)

Query: find all descendants of Activist₁
Design Decisions

- Protocols, not system
- Use CloudDB
- Limited guarantees
Outline

- Introduction
- Design Issues
- Protocols
- Evaluation
- Conclusions
P1: Standalone Cloud Store

- Stores both provenance and data on cloud object store
  - Provenance as a separate object

- Amazon S3 and Azure Blob
  - Object identified by URI
  - SOAP or REST interface
  - Operations: PUT, GET, COPY, DELETE
  - Cost: data storage + bandwidth + num ops
  - S3 - Eventual consistency
P1: Standalone Cloud Store
P1: Standalone S3

PASS

PUT:Prov
PUT:Data

S3
## Properties

<table>
<thead>
<tr>
<th></th>
<th>Data Coupling</th>
<th>Causal Ordering</th>
<th>Persistence</th>
<th>Efficient Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>
P2: Cloud Store + Cloud Database

- Store data in cloud blob store
- Store provenance in cloud database
- Amazon SimpleDB, Azure Table
  - Semi-Structured Data model: items described by attribute-value pairs
  - Operations: PutAttributes, GetAttributes, DeleteAttributes
  - Query: SELECT/LINQ
  - name/value size: 1KB or 64KB
  - Cost: bandwidth + storage + num ops + machine hrs
P2: Cloud Store + Cloud Database

Client Application

PASS

Cloud Store

Data

Prov

CloudDB
P2: S3 + SimpleDB
## Properties

<table>
<thead>
<tr>
<th></th>
<th>Data Coupling</th>
<th>Causal Ordering</th>
<th>Persistence</th>
<th>Efficient Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>P2</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
P3: Cloud Store + Cloud DB + Messaging Service

- P2 + use messaging service as a log
- Amazon Simple Queuing Service (SQS), Azure Queue
  - Distributed Messaging System
  - Queues are identified by URL
  - Operations: SendMessage, ReceiveMessage, DeleteMessage
  - Limits: 8KB message size
P3: Store + Database + Queue service
Protocol 3: S3 + SimpleDB + SQS

PASS

PUT: Temp copy

S3

COPY

Commitd

SndMsg+

SQS

RecvMsg+

SimpleDB

PutAttrs+
Protocol 3: S3 + SimpleDB + SQS
## Properties

<table>
<thead>
<tr>
<th></th>
<th>Data Coupling</th>
<th>Causal Ordering</th>
<th>Persistence</th>
<th>Efficient Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>P2</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>P3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Only ensure eventual data coupling
Outline

- Introduction
- Design Issues
- Protocols
- Evaluation
- Conclusions
Evaluation

- Results AWS specific
- Baseline S3fs
- Workloads
  - Microbenchmarks
  - Application benchmarks
  - Query benchmarks
  - Cost overheads
MicroBenchmark Results

Elapsed Time (s)

S3fs  44%
P1    79%
P2    2%
P3    Fastest

Provenance for the Cloud - FAST*10
Application Benchmarks

Not much difference between protocol performance
Query Results

- Recall
  - P1 uses S3
  - P2, P3 use SimpleDB
- SimpleDB was much faster
  - Speedup depends on the query
## Cost Overheads

<table>
<thead>
<tr>
<th></th>
<th>Nightly</th>
<th>Blast</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3fs</td>
<td>$1.05</td>
<td>$0.37</td>
<td>$0.27</td>
</tr>
<tr>
<td>P1</td>
<td>$1.05</td>
<td>$0.39</td>
<td>$0.29</td>
</tr>
<tr>
<td>P2</td>
<td>$1.05</td>
<td>$0.38</td>
<td>$0.29</td>
</tr>
<tr>
<td>P3</td>
<td>$1.06</td>
<td>$0.40</td>
<td>$0.30</td>
</tr>
</tbody>
</table>
Evaluation Summary

- Obtaining statistical significance hard
  - Too many uncontrollable factors: WAN latency, service load, software version
  - Services seem to be getting better
Outline

- Introduction
- Design Issues
- Protocols
- Evaluation
- Conclusions
Conclusions

- We have shown how to store provenance in today’s cloud offerings
- Performance results show that we can use the most robust protocol
- Future work: Native cloud provenance
  - Architecture
  - Trusted provenance
  - Graph processing and provenance mining
Thanks!

- Questions?

  kiran@eecs.harvard.edu
  www.eecs.harvard.edu/~kiran