TripS: Automated Multi-tiered Data Placement in a Geo-distributed Cloud Environment

Kwangsung Oh, Abhishek Chandra, and Jon Weissman
Department of Computer Science and Engineering
University of Minnesota Twin Cities

Systor 2017
Cloud Providers Publicly Available

- HP Public Cloud
- Amazon Web Services
- Google
- IBM
- Windows Azure

Private Cloud

Powered by Intel Cloud Technology
Users are around the Globe
Geo-Distributed Users, DCs and Applications

Where are the **best locations** for storing data?
Different Applications’ goals

- SLA
- Consistency Model
- Desired Cost
- Desired Fault Tolerance
- Data Access Pattern
- Users’ Locations
- And many more…
Previous Data Placement Systems

• **Volley** [Agarwal et al, NSDI ’10]
• **Spanner** [Dean et al, OSDI ’12]
• **SPANStore** [Wu et al, SOSP ’13]
• **Tuba** [Ardekani et al, OSDI ’14]

• **Focusing on data center locations**
Multiple Storage Tiers Available

Both DC locations and storage tiers should be considered for optimized data placement.

Different Characteristics
- Performance
- Pricing
- Durability
- Availability...
Challenges

• Many options for data center locations and storage tiers

• Dynamics from cloud environment
Data Center Locations Options

Colocation Data Center Statistics, Israel

Jerusalem: 1
Tel Aviv: 7

From http://www.datacentermap.com

chart by amCharts.com
Storage Services Options

- Block Storage (EBS) (SSD, HDD)
  - EBS-gp2
  - EBS-io1
  - EBS-st1
  - EBS-sc1
  - Magnetic
  - S3
  - S3-IA
  - S3-RRS
  - Glacier

- Object Storage
- File Storage (EFS)
- ElastiCache

Many storage tiers
Challenges

✓ Many options for data center locations and storage tiers

• **Dynamics** from cloud environment
Dynamics from

• **Infrastructure**
  • Cloud service providers do **not guarantee consistent performance**
  • E.g., transient DCs (or network) failure, burst access pattern, overloaded node and so on

• **Applications**
  • User locations and access patterns **keep changing**
  • E.g., users are travelling world widely, changes in data popularity
Goal

• Finding optimized data placement
  • Exploiting both DC locations and multiple storage tiers
  • Helping applications handle dynamics
Roadmap

✅ Motivations & Goals

• **TripS (Storage Switch System)**
• Handling dynamics
• Experimental Evaluations
TripS

- Light-weight **data placements decision system**; considering both **DC locations** and **storage tiers**
- Helping applications to **handle dynamics**
System Model

• Geo-distributed storage system (GDSS)
  • Running on multiple DCs (across different cloud providers)
  • Exploiting multiple storage tiers
System Model

- Applications are running on GDSS
  - Connecting any GDSS server (possibly the closest server)
  - Using Get/Put API exposed by GDSS
Locale

- \{\text{DC location, storage tier}\} \text{ tuple}
- E.g., 9 locales are available
Data Placement Problem

- **Determining set of locales** to store data
  - Satisfying all applications’ goals

- {US East, SSD}
- {US East, HDD}
- {US East, Object}

- {EU West, SSD}
- {EU West, HDD}
- {EU West, Object}

- {Asia SE, SSD}
- {Asia SE, HDD}
- {Asia SE, Object}
TripS Inputs

- **Application desired goals**
  - SLA
  - Consistency model
  - Degree of fault tolerance
  - Locale count (LC)

- **Cost information**
  - Storage and Network cost

- **Latency information**
  - Storage and network (between DCs)

- **Workload information**
  - Number of Requests (Get and Put)
  - Average data size

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Set of DCs</td>
</tr>
<tr>
<td>$D_i \cdot S$</td>
<td>Set of storage tiers in DC $i$</td>
</tr>
<tr>
<td>$C_{network}^{i,j}$</td>
<td>Network cost between DC $i$ and DC $j$</td>
</tr>
<tr>
<td>$C_{storage}^t$</td>
<td>Storage tier $t$ provisioned storage cost in DC $i$</td>
</tr>
<tr>
<td>$C_{get/put}^{i,t}$</td>
<td>Get/Put request cost for storage tier $t$ in DC $i$</td>
</tr>
<tr>
<td>$C_{ret/write}^{i,t}$</td>
<td>Data retrieval/write cost from/to storage tier $t$ in DC $i$</td>
</tr>
<tr>
<td>$SLA_{get/put}$</td>
<td>Get/Put operation SLA from each DC</td>
</tr>
<tr>
<td>$LC (&gt; 0)$</td>
<td>Locale count in the TLL that can be accessed within SLA from each DC location</td>
</tr>
<tr>
<td>$F$</td>
<td>Minimum number of DC faults handled</td>
</tr>
<tr>
<td>Consistency</td>
<td>Consistency Model</td>
</tr>
<tr>
<td>$Size_i$</td>
<td>Average object size in DC $i$</td>
</tr>
<tr>
<td>Center</td>
<td>Centralized DC location for a Global Lock (in strong Consistency)</td>
</tr>
<tr>
<td>$L_{network}^{i,j}$</td>
<td>Network latency from DC $i$ to DC $j$</td>
</tr>
<tr>
<td>$L_{get/put}^t$</td>
<td>Get/Put latency for storage tier $t$ in DC $i$</td>
</tr>
<tr>
<td>$A_{get/put}^i$</td>
<td>Number of Get/Put requests for DC $i$</td>
</tr>
</tbody>
</table>
Optimized Data Placement

- **Solving data placement problem** with given inputs as MILP (Mixed Integer Linear Problem)

- **Minimized**

**Total cost** = Get Cost + Put cost + Broadcast Cost + Storage Cost

- **Get Cost:** \[ \sum_i A_{get}^i \cdot \sum_j \sum_t T_{ijt} \cdot (\text{Size}_i \cdot (C_{ij}^{\text{network}} + C_{jt}^{\text{get}}) + C_{jt}^{\text{get,req}}) \]

- **Put Cost:** \[ \sum_i A_{put}^i \cdot \sum_j \sum_t T_{ijt} \cdot (\text{Size}_j \cdot (C_{ij}^{\text{network}} + C_{jt}^{\text{write}}) + C_{jt}^{\text{put,req}}) \]

- **Broadcast Cost:** \[ \sum_i A_{put}^i \cdot \sum_j \sum_k \sum_l T_{ijkl} \cdot (\text{Size}_j \cdot (C_{jk}^{\text{network}} + C_{lt}^{\text{write}}) + C_{lt}^{\text{put,req}}) \]

- **Storage Cost:** \[ \sum_i \sum_t P_{it} \cdot \text{Size}_i \cdot C_{it}^{\text{storage}} \]
Data Placement Example

- TripS decides to store data in 2 locales
  - {US East, HDD}, {Asia SE, Object}
Roadmap

✓ Motivations & Goals
✓ TripS (Storage Switch System)

• Handling dynamics
• Experimental evaluations
Dynamics

• Long-term dynamics
  • E.g., diurnal access pattern, user locations
  • From hour(s) to week(s)
  • Lazy re-evaluating the data placement is enough

• Short-term dynamics
  • E.g., burst access, transient failures or overload
  • From second(s) to minute(s)
  • Frequent re-evaluating the data placement is expensive!!
Target Locale List (TLL)

- **List of locales** satisfying the SLA goal
  - *Locale count* (LC) parameter = 1 (as an application’s goal)
Target Locale List (TLL)

- **List of locales** satisfying the SLA goal
  - *Locale count* (LC) parameter = 2 (as an application’s goal)
Locale Switching

- Avoiding SLA violation
- Tradeoff cost for performance
Roadmap

✓ Motivations & Goals
✓ TripS (Storage Switch System)
✓ Handling dynamics

• Experimental evaluation
Evaluation

- Running on Wiera [Oh et al, HPDC ’16] as GDSS
- 8 Amazon DCs and 3 storage tiers
- Evaluation illustrates
  - TripS finds optimized data placement
  - TripS helps applications handle dynamics (e.g., network delays or transient failures)
TripS Finds Optimized Data Placement

- Two synthetic workloads
  - Latency sensitive Web applications
  - Data analytic applications
- Compare with emulated SPANStore [Wu et al, SOSP ’13]
  - Only one storage tier (S3 or EBS) on TripS

<table>
<thead>
<tr>
<th>Workload</th>
<th>Average Data Size</th>
<th># Get / Put Request</th>
<th>Get / Put SLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload 1</td>
<td>8 KB (small data)</td>
<td>10,000 / 1,000</td>
<td>200 ms / 350 ms (latency sensitive)</td>
</tr>
<tr>
<td>Workload 2</td>
<td>100 MB (big data)</td>
<td>1,000 / 100</td>
<td>500 ms / 800 ms (bandwidth sensitive)</td>
</tr>
</tbody>
</table>
Optimized Data Placement for Both Workload

- **Only 1 storage tier for TripS**
  - EBS-st1
  - Emulated SPANstore
  - Workload 1

- **Any storage tiers combination for TripS**
  - S3
  - EBS-st1
  - Emulated SPANstore
  - TripS
  - Workload 2
Handling Short-term Dynamics

- 5 DCs on North America region
- Workload
  - YCSB Workload B
    - 95% Read, 5% Write
  - Average data size: 8 KB
  - 80 ms (Get) / 200ms (Put)
- Varying LC parameter
Transient Network Delays with $\text{LC} = 1$

- SLA violation!!
- Dynamic but no SLA violation

Chart showing latency over time with markers indicating SLA violations.
Transient Network Delays with $\text{LC} = 2$

- SLA violation more than 30 seconds!!
- No more dynamics
- Switch Locale!
- No Period violation
## Tradeoff Cost for Performance by LC

- As LC increases, total cost also increases
- Tradeoff cost for performance

<table>
<thead>
<tr>
<th>LC parameter</th>
<th>Data placement</th>
<th>Storage</th>
<th>Network</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{US East, EBS-st1}, {US East 2, EBS-st1}, {US West 2, EBS-st1}</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>{US East, EBS-st1}, {US East 2, \textbf{EBS-gp2}}, {US West 2, EBS-st1}</td>
<td>140.7%</td>
<td>100%</td>
<td>105.3%</td>
</tr>
<tr>
<td>3</td>
<td>{US East, \textbf{EBS-gp2}}, {US East 2, EBS-gp2}, {US West, EBS-st1}</td>
<td>188.1%</td>
<td>100%</td>
<td>111.5%</td>
</tr>
<tr>
<td>4</td>
<td>{US East, EBS-gp2}, {US East 2, EBS-gp2}, {US West, EBS-st1}, {CA central, EBS-gp2}</td>
<td>269.6%</td>
<td>166.7%</td>
<td>180.1%</td>
</tr>
</tbody>
</table>
Real Application Scenario - Retwis

- Twitter like Web application
- Using TripS-enabled Wiera instead of Redis
Satisfying SLA Goals

Put SLA: 200 ms
Get SLA: 80 ms

Latency (ms)

Get | Put

US East | US East 2 | US West | US West 2 | CA Central | EU West | Asia SE | Asia NE

1K users: 125 Users per each location
Conclusion

• TripS finds optimized data placement with a consideration both **DC locations** and **storage tiers** with **minimized cost**

• TripS helps applications **handle dynamics** especially **short-term dynamics** with Target Locale List (TLL)
Thank You!