DOLLY: Virtualization-Driven Database Provisioning for the Cloud

Emmanuel Cecchet
Joint work with Rahul Singh, Upendra Sharma and Prashant Shenoy
THE CLOUD

- Virtualization
- Pay as you go
- Elasticity
PROVISIONING IN THE CLOUD

- Based on request volume and resource usage
- Reactions based on thresholds
- Works for stateless tiers
WHY IS IT HARD TO ADD A DB REPLICA?

MySQL backup
MySQL restore
New replica
Replica ready

Replay updates

snapshot

5pm
WHY IS IT HARD TO ADD A DB REPLICA?

2pm snapshot

2pm

MySQL restore

New replica

Replica ready

Replay updates

5pm

2pm snapshot
WHY IS IT HARD TO ADD A DB REPLICA?

Queries are slow… Let’s improve this!
CREATE INDEX i1 on Table1; CREATE INDEX i2 on Table 2

MySQL backup
snapshot
MySQL restore
WHY IS IT HARD TO ADD A DB REPLICA?

- Warehouse 1GB
  - PostgreSQL backup
  - PostgreSQL snapshot
  - PostgreSQL restore
  - 24min

- Warehouse 10GB
  - PostgreSQL backup
  - PostgreSQL snapshot
  - PostgreSQL restore
  - 1H30min

- apt-get update postgresql
- echo 1 > /proc/sys/magic_options
- CREATE USER x
- GRANT PRIVILEGES TO y
WHAT ARE THE MAIN PROBLEMS?

- When to start replica spawning?
  - How to predict replica spawning time?
  - How to make replica spawning platform independent?
- When to generate new snapshots?
- How can we minimize resource usage?
  - Power/cooling in private cloud
  - $ cost in public cloud
VM Cloning: Backup/Restore in Constant Time

- Filesystem snapshot/copy is OS & DB agnostic
- Only depends on VM size

<table>
<thead>
<tr>
<th>Database</th>
<th>DB size on disk</th>
<th>DB Backup Restore</th>
<th>Dolly 4GB VM cloning</th>
<th>Dolly 16GB VM cloning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUBiS –c–i</td>
<td>1022MB</td>
<td>843s</td>
<td>281s</td>
<td>899s</td>
</tr>
<tr>
<td>RUBiS +c+bi</td>
<td>1.4GB</td>
<td>5761s</td>
<td>282s</td>
<td>900s</td>
</tr>
<tr>
<td>RUBiS +c+fi</td>
<td>1.5GB</td>
<td>6017s</td>
<td>280s</td>
<td>900s</td>
</tr>
<tr>
<td>TPC-W</td>
<td>684MB</td>
<td>288s</td>
<td>275s</td>
<td>905s</td>
</tr>
<tr>
<td>TPC-H 1GB</td>
<td>1.8GB</td>
<td>1477s</td>
<td>271s</td>
<td>918s</td>
</tr>
<tr>
<td>TPC-H 10GB</td>
<td>12GB</td>
<td>5573s</td>
<td>n/a</td>
<td>911s</td>
</tr>
</tbody>
</table>
Dolly

- Database replication in the Cloud
- Provisioning with Dolly
- Prototype & Evaluation
SPAWNING A REPLICA WITH CLONING

- Backup & Restore replace by VM cloning
SPAWNING IN A PRIVATE CLOUD

- Clone entire virtual machine for backup/restore
- Backup server is optional
SPAWNING IN A PUBLIC CLOUD

- Storage decoupled from computing resource
- Starting a new instance clones the volume
Dolly

- Database replication in the Cloud
- Provisioning with Dolly
- Prototype & Evaluation
MODELING SPAWNING TIME

- Predictable backup and restore times are required
- Replay time can be estimated from write throughput
  - $w_t$: current workload write throughput
  - $w_{max}$: replay speed of the spawning replica

\[
\text{replica spawning time} = \frac{b_i}{w_t} + \frac{r_i}{w_{max}} + \frac{w_t}{w_{max}}
\]
WHEN TO SNAPSHOT?

- Time to spawn from a live replica
  \[ s = (b_i + r_i) \frac{w_{\text{max}}}{w_{\text{max}} - w_t} \]

- Time to spawn from an existing snapshot
  \[ s = (r_i + \text{replay}_i) \frac{w_{\text{max}}}{w_{\text{max}} - w_t} \]

- Faster to take a new snapshot \( j \) to spawn a new replica than using old snapshot \( i \) if:
  \( \text{backup}_j + \text{restore}_j < \text{restore}_i + \text{replay}_i \)
DOLLY OVERVIEW

- Input
  - capacity prediction
  - write prediction

- Output
  - schedule of snapshots
  - schedule of replica spawning
  - admission control if needed

---

- Predictors
- Capacity Provisioning
  - HA adjuster
  - Spawning options
  - Write throttling
- Snapshot scheduler
- Scheduler
- Paused pool cleaner
- Admission Control
- Management API
- Monitoring
- Free pool Manager
- Start/stop clone/snapshot
- Write throttling/ read throttling
- Reclaim
PROVISIONING REPLICA

- Dolly does not provide predictors
- Dolly can work with any predictor (see [Eurosys09])
CLOUD COST FUNCTIONS

- Adapt the provisioning decisions to the cloud platform specifics
- Cost can be $ on public cloud or time on private cloud

<table>
<thead>
<tr>
<th>Cost function name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause_cost(VM, t)</td>
<td>cost of pausing VM at time t</td>
</tr>
<tr>
<td>spawn_cost(s, t, d)</td>
<td>cost to spawn a replica from snapshot s at time t to meet deadline d</td>
</tr>
<tr>
<td>spawn_cost(VM, t, d)</td>
<td>cost to spawn a replica from a paused VM at time t to meet deadline d</td>
</tr>
<tr>
<td>running_cost(VM, t1, t2)</td>
<td>cost to run a VM from time t1 to time t2</td>
</tr>
<tr>
<td>pause_resume_cost(VM, t1, t2)</td>
<td>cost to pause a VM at time t1 and resume it at time t2</td>
</tr>
<tr>
<td>backupPaused_cost(VM)</td>
<td>cost to backup a paused VM</td>
</tr>
<tr>
<td>backupLive_cost(VM, t)</td>
<td>cost to backup an active VM at time t</td>
</tr>
</tbody>
</table>
PROVISIONING REPLICAS

- Parse capacity provisioning predictions
- Decrease capacity by pausing VMs
- Increasing capacity
  - Check if we can reuse a paused VM
  - Check if we can spawn from an existing snapshot
  - Choose cheapest options according to \text{spawn\_cost} function
  - Perform admission control if all replicas cannot be provisioned in time
SNAPSHOT SCHEDULING

- How to snapshot?
  - Clone a paused VM
  - Pause an active VM to clone it

- When to snapshot?
  - At time $j$ when $backup_j + restore_j < restore_i + replay_i$
  - If new snapshot is scheduled, re-run capacity provisioning

- Prediction window must have minimum size

\[ pw \geq b_{backup_{i+1}} + r_{backup_{i+1}} + replay_{backup_{i+1 \text{ switch}}} \]
Dolly

- Database replication in the Cloud
- Provisioning with Dolly
- Prototype & Evaluation
IMPLEMENTATION

- C-JDBC/Sequoia replication middleware
- OpenNebula Cloud management middleware
- Cost functions
  - private cloud: minimize resource utilization time
  - Amazon EC2: minimize cost
IMPLEMENTATION – COST FUNCTIONS

- Private cloud: minimize resource utilization
- Amazon EC2: minimize cost

<table>
<thead>
<tr>
<th>Cost function name</th>
<th>Private Cloud</th>
<th>EC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause_cost(VM, t)</td>
<td>return 1/VM-&gt;machine-&gt;temp</td>
<td>return 60-((t-VM-&gt;start)%60)</td>
</tr>
</tbody>
</table>
| spawn_cost(s, t, d)         | return d-t    | comp$=(d-t)/60*hour$  
  io$=EBS_storage*$s->size +  
  EBS_io*$  
  (s->restore_io+s->replay_io)  
  return comp$+io$             |
| spawn_cost(VM, t, d)        | return d-t    | comp$=(d-t)/60*hour$  
  io$= EBS_io*$  
  (s->resume_io+s->replay_io)  
  return comp$+io$             |
| running_cost(VM,t1,t2)      | return 1      | (t2-t1)/60*hour$                                               |
| pause_resume_cost(VM, t1, t2)| if (t2-t1 >  
  VM->pause + VM->resume)  
  return 0  
  else return 2        | io$= EBS_io*$  
  (VM->pause_io+VM->resume_io)  
  comp$=(60-(VM->stop-VM->start)  
  %60)/60*hour$  
  return io$+ comp$       |
| backup_paused_cost(VM)      | return backup_time | return S3_storage*$s->size                                  |
| backup_live_cost(VM, t)     | return VM->pause +  
  backup_time + VM->resume | return pause_cost(VM, t)$+  
  S3_storage*$s->size +  
  (VM->stop_io+VM->start_io)*  
  EBS_io$                   |
TPC-W EVALUATION

- Multi-tier online bookstore benchmark
- 4GB Xen VM for the database
- Large EC2 instances from EBS volumes with CloudWatch

<table>
<thead>
<tr>
<th>Operation</th>
<th>Private Cloud</th>
<th>Public Cloud (EC2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>start VM</td>
<td>42s</td>
<td>220s</td>
</tr>
<tr>
<td>pause VM</td>
<td>26s</td>
<td>30s</td>
</tr>
<tr>
<td>resume VM</td>
<td>42s</td>
<td>30s</td>
</tr>
<tr>
<td>backup (stop/clone)</td>
<td>150s</td>
<td>320s</td>
</tr>
<tr>
<td>restore (clone/start)</td>
<td>165s</td>
<td>220s</td>
</tr>
<tr>
<td>$w_{\text{max}}$</td>
<td>149 writes/sec</td>
<td>197 writes/sec</td>
</tr>
<tr>
<td>Avg IOs per write</td>
<td>15</td>
<td>13</td>
</tr>
</tbody>
</table>
WORKLOAD DESCRIPTION

- Snapshot $s_0$ available at $t_0$
Overprovisioning with 6 replicas – 1h snapshot

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>720m</td>
<td>0</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$8.39</td>
<td>0</td>
</tr>
</tbody>
</table>
Reactive provisioning

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>410m</td>
<td>42.1</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$4.61</td>
<td>41.5</td>
</tr>
</tbody>
</table>
Reactive provisioning – 15m snapshot

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>381m42s</td>
<td>17.5</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$18.29</td>
<td>27.2</td>
</tr>
</tbody>
</table>

![Graph showing reactive provisioning over time](image-url)
Dolly – 30m Prediction Window

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>352m</td>
<td>0</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$3.73</td>
<td>0</td>
</tr>
</tbody>
</table>

Cheaper to leave instances online.
CONCLUSION

- **VM cloning**
  - Solves administration issues by blackboxing the database
  - Constant time backup/restore needed to predict replica spawning time
- **New provisioning algorithm**
  - Decouples capacity provisioning from snapshot scheduling
  - Cost functions to optimize for cloud platform specifics
Bonus Slides
Dolly – 10m Prediction Window

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>381m54s</td>
<td>0</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$7.16</td>
<td>0</td>
</tr>
</tbody>
</table>
Reactive provisioning – 1h snapshot

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>MRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private cloud</td>
<td>360m30s</td>
<td>25.8</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>$5.00</td>
<td>33.7</td>
</tr>
</tbody>
</table>
BACKUP/RESTORE TECHNIQUES

- Database native tools
  - Vendor specific or 3rd party ETL
  - Understand database semantics

- Filesystem copy
  - Low-level data copy
  - Need to know what to copy

- VM cloning
  - Copies database content + configuration + OS
  - Unused space can be compressed
## DATABASE SIZES

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>DB size</th>
<th>Snapshot size</th>
<th>VM size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RUBiS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyISAM no constraint</td>
<td>836MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyISAM w/ constraints</td>
<td>1.1GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MyISAM w/ constraint &amp; index</td>
<td>1.2GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InnoDB no constraint</td>
<td>1022MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InnoDB w/ constraints</td>
<td>1.4GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InnoDB w/ constraint &amp; index</td>
<td>1.5GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TPC-W</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostgreSQL binary dump</td>
<td>684MB</td>
<td>210MB</td>
<td>2.1GB</td>
</tr>
<tr>
<td>PostgreSQL sql dump</td>
<td></td>
<td>314MB</td>
<td></td>
</tr>
<tr>
<td><strong>TPC-H scale 1(GB)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostgreSQL binary dump</td>
<td></td>
<td>307MB</td>
<td>1.1GB (OS) + 2.1GB (data)</td>
</tr>
<tr>
<td>PostgreSQL sql dump</td>
<td>1.8GB</td>
<td>1.2GB</td>
<td></td>
</tr>
<tr>
<td><strong>TPC-H scale 10(GB)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostgreSQL binary dump</td>
<td></td>
<td>12GB</td>
<td>2.0GB</td>
</tr>
<tr>
<td>PostgreSQL sql dump</td>
<td></td>
<td>7.3GB</td>
<td>16GB</td>
</tr>
</tbody>
</table>
BACKUP/RESTORE PERFORMANCE (1/3)

- Performance depends on database content
BACKUP/RESTORE PERFORMANCE (2/3)

- File copy is the most effective for small databases
VM cloning most effective on large databases
## BACKUP/RESTORE SUMMARY

<table>
<thead>
<tr>
<th>Feature</th>
<th>DB Backup/Restore</th>
<th>Filesystem Copy</th>
<th>VM Cloning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database specific knowledge</td>
<td>Medium</td>
<td>Very high</td>
<td>None</td>
</tr>
<tr>
<td>Performance</td>
<td>Slow</td>
<td>Fastest</td>
<td>Fast</td>
</tr>
<tr>
<td>Snapshot size</td>
<td>Small</td>
<td>DB size</td>
<td>VM size</td>
</tr>
<tr>
<td>Spawning time predictability</td>
<td>Hard</td>
<td>Moderate</td>
<td>Easy</td>
</tr>
<tr>
<td>Database installation</td>
<td>Moderate</td>
<td>Moderate</td>
<td>None</td>
</tr>
<tr>
<td>Database configuration</td>
<td>Hard</td>
<td>Hard</td>
<td>None</td>
</tr>
<tr>
<td>Missing data in transfer</td>
<td>Possible</td>
<td>Unlikely</td>
<td>No</td>
</tr>
<tr>
<td>Spawning atomicity</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Resynchronization limitations</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
DOLLY MAIN ALGORITHM

- Capacity provisioning depends on available snapshots
- Snapshots scheduled according to capacity demand
- Decouple capacity provisioning from snapshot scheduling

```java
if (predictor.capacity_changes ||
    predictor.write_workload_changes) {
    do {
        schedule = capacity_provisioning(predictions)
        snapshot_schedule = snapshot_scheduling(predictions)
    } while (snapshot_schedule schedules new snapshots)
    scheduler.schedule(snapshot_schedule)
    scheduler.schedule(capacity_schedule)
}
if (time since last operation > threshold) {
    paused_pool_cleaner.release_old_paused_vms();
    paused_pool_cleaner.delete_old_snapshots();
}
```
RELEASING RESOURCES

- Paused VMs
  - VM never re-used if cost to resume > cost to spawn from last snapshot

- Snapshots
  - Old snapshots can be released based on cost to keep them around

- Free server pool
  - Can reclaim servers with paused VMs when pool is empty