Ovid

A Software-Defined Distributed Systems Framework

Deniz Altinbuken, Robbert van Renesse
Cornell University
Ovid

Build distributed systems that are

easy to evolve

easy to reason about

easy to compose
Approach

Create a theoretical model using abstractions

Deploy the distributed system using this model
Approach

- Create a theoretical model using abstractions
- Deploy the distributed system using this model
not fault tolerant
transform data store agent to be fault tolerant

\[ \text{replicate(datastore,3)} = \]
transform data store agent to be fault tolerant

replicate(datastore,3) = datastore*3
minority of replicas can fail

crash fault tolerant
asynchronous environment
linearizable

\[
\text{replicate(datastore,3)} = R^*3 + \text{datastore}^*3
\]
minority of replicas can fail
crash fault tolerant
asynchronous environment
linearizable

replicate(datastore,3) = proxy + R*3 + datastore*3
How do we know that a transformation did not break an agent?
How do we know that a transformation did not break an agent?
Use refinement mappings to prove equivalence
Use path names to represent transformations
Supported transformations

- replication
- byzantine resistance
- batching
- sharding
- encryption/decryption
- compression
- load-balancing
- deduplication
Approach

Create a theoretical model using abstractions

Deploy the distributed system using this model
Deployment

- Create a configuration from the model
- Deploy processes on boxes depending on the configuration
Deployment

- Create a configuration from the model
- Deploy processes on boxes depending on the configuration
Deployment

Machine Configuration

client 192.168.7.56 client.cpp
proxy 192.168.7.56 proxy.cpp
R1 192.168.7.80 replica.cpp
R2 192.168.7.81 replica.cpp
R3 192.168.7.82 replica.cpp
data store1 192.168.7.80 datastore.cpp
data store2 192.168.7.81 datastore.cpp
data store3 192.168.7.82 datastore.cpp
Deployment

Machine Configuration
- **client**: 192.168.7.56 client.cpp
- **proxy**: 192.168.7.56 proxy.cpp
- **R1**: 192.168.7.80 replica.cpp
- **R2**: 192.168.7.81 replica.cpp
- **R3**: 192.168.7.82 replica.cpp
- **data store1**: 192.168.7.80 datastore.cpp
- **data store2**: 192.168.7.81 datastore.cpp
- **data store3**: 192.168.7.82 datastore.cpp
Dynamic Routing

- Create routing tables
- Route messages to destination depending on the model
Dynamic Routing

Routing Configuration
- **client** datastore proxy
- **proxy** datastore R1,R2,R3
- R1 datastore datastore1
- R2 datastore datastore2
- R3 datastore datastore3
- **datastore1** client client
- **datastore2** client client
- **datastore3** client client
Dynamic Routing

msg: [dest, payload]

client → proxy

controller

R_1 → data store_1
R_2 → data store_2
R_3 → data store_3
Dynamic Routing

msg: [dest payload]

client → proxy

R1 → data store1
R2 → data store2
R3 → data store3

controller

data store put(1,6)
Dynamic Routing

- send msg(dest,payload)
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
Dynamic Routing

• send msg(dest, payload)
• look for dest in routing table
• dest not present
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
- dest not present
- send lookup message to controller

Routing Configuration

<table>
<thead>
<tr>
<th>Type</th>
<th>Datastore</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>datastore proxy</td>
</tr>
<tr>
<td>proxy</td>
<td>datastore R1,R2,R3</td>
</tr>
<tr>
<td>R1</td>
<td>datastore datastore1</td>
</tr>
<tr>
<td>R2</td>
<td>datastore datastore2</td>
</tr>
<tr>
<td>R3</td>
<td>datastore datastore3</td>
</tr>
<tr>
<td>datastore1</td>
<td>client client</td>
</tr>
<tr>
<td>datastore2</td>
<td>client client</td>
</tr>
<tr>
<td>datastore3</td>
<td>client client</td>
</tr>
</tbody>
</table>

msg: [dest, payload]
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
Dynamic Routing

- send msg(dest, payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
- update routing table

msg: dest payload

data store

data store

data store
Dynamic Routing

msg: \[\text{dest \hspace{1mm} payload}\]

- send msg\((\text{dest}, \text{payload})\)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
- update routing table
- send msg\((\text{dest}, \text{payload})\)
Dynamic Routing

- send msg(dest, payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
- update routing table
- send msg(dest, payload)
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
- update routing table
- send msg(dest,payload)

msg: \[\text{dest \| payload}\]
Dynamic Routing

- send msg(dest,payload)
- look for dest in routing table
- dest not present
- send lookup message to controller
- get route mapping
- update routing table
- send msg(dest,payload)
Conclusion

• Ovid introduces new abstractions and a new way of modeling distributed systems.

• Ovid can create distributed systems that can be reconfigured and deployed on the fly.

• Ovid makes building, running, maintaining and evolving distributed systems an easy task.