Cheap and Available State Machine Replication

Rong Shi, Yang Wang
The Ohio State University
Protect data from failures

• Data is important
  Durability: data is never lost
  Availability: data can be accessed at any time

• Failures
  Power loss, DRAM bit errors, disk corruption, software bugs, ...
Protect data from failures

• Data is important
  Durability: data is never lost
  Availability: data can be accessed at any time

• Failures
  Power loss, DRAM bit errors, disk corruption, software bugs, ...

How to ensure data durability and availability despite failures?
Protect data from failures

• Data is important
  Durability: data is never lost
  Availability: data can be accessed at any time

• Failures
  Power loss, DRAM bit errors, disk corruption, software bugs, ...

How to ensure data durability and availability despite failures?

Replication
Basic idea of replication
Basic idea of replication
Basic idea of replication

Redundancy => fault tolerance
Basic idea of replication

Redundancy => fault tolerance

Data replication => availability and durability
Stronger replication requires more replicas

• **Primary-backup:** \( f+1 \) replicas \( \Rightarrow \) \( f \) crash failures
  
  Data: GFS [Ghemawat SOSP’03], HDFS [Shvachko MSST’10], ...

• **Paxos:** \( 2f+1 \) replicas \( \Rightarrow \) \( f \) crash failures and timing errors (e.g. long message delay)
  
  Lock service: Boxwood [MacCormick OSDI’04], Chubby [Burrows OSDI’06], ...
  
  Data: SMART [Lorch Eurosys’06], ...
  
  Metadata: MS Azure [Calder SOSP’11], ...
  
  Data + metadata: Megastore [Baker CIDR’11], Spanner [Corbett OSDI’12], ...

• **BFT:** \( 3f+1 \) replicas \( \Rightarrow \) \( f \) arbitrary failures
  
  Data + metadata: FARSITE [Adya OSDI’02], UpRight [Clement SOSP’09], ...
Stronger replication requires more replicas

- Are we willing to pay a higher cost for stronger guarantees?
Existing work made other tradeoffs

• On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosys’11], ...

[Diagram showing client connected to active replicas and backup replica]
Existing work made other tradeoffs

• On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosys’11], ...
Existing work made other tradeoffs

- On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosys’11], ...
Existing work made other tradeoffs

• On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosys’11], ...
Existing work made other tradeoffs

- On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosys’11], …

The system is unavailable when activating a backup replica (transfer data)
Existing work made other tradeoffs

• On-demand instantiation for asynchronous replication protocols
  Cheap Paxos [Lamport DSN’04], ZZ [Wood Eurosyst’s11], ...

Activate minimum subset of replicas
Activate a backup replica
The system is unavailable when activating a backup replica (transfer data)
Existing work made other tradeoffs

• Separating agreement from execution ([Yin SOSP’03])
  Separating a replica into an agreement node and an execution node
  In BFT, # execution nodes can be smaller than # agreement nodes
  Not effective for applications that are heavy in agreement or using Paxos
Existing work made other tradeoffs

• Separating agreement from execution ([Yin SOSP’03])
  Separating a replica into an agreement node and an execution node
  In BFT, # execution nodes can be smaller than # agreement nodes
  Not effective for applications that are heavy in agreement or using Paxos

• Separating metadata from data (Gnothi [Wang ATC’12])
  Full replication of metadata and partial replication of data
  Only effective for block storage

• ......
Is it possible to reduce replication cost without hurting availability and correctness?
Is it possible to reduce replication cost without hurting availability and correctness?

Yes for many popular protocols (e.g. Paxos, UpRight)
## Highlights

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>$2f + 1$</td>
<td>$f + 1$</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>$u + \max(u, r) + 1$</td>
<td>$u + r + 1$</td>
</tr>
</tbody>
</table>

$u$: number of omission failures  
$r$: number of commission failures
Background: State Machine Replication (SMR)
Background: State Machine Replication (SMR)
Background: State Machine Replication (SMR)
Background: State Machine Replication (SMR)

- Deterministic state machine
- Agreement Protocol
- Server Application
- Replica

Input flow:
- Input
- Input
- Input
- Input
- Input

Server program interactions:
- Server
- Replica
Background: State Machine Replication (SMR)
Our solution: on-demand instantiation + lazy recovery

- Reduce cost with on-demand instantiation
  - Activate minimum set of replicas and wakeup backup ones when active ones fail
  - Problem: system is unavailable when rebuilding backup replica
Our solution: on-demand instantiation + lazy recovery

• Reduce cost with on-demand instantiation
  Activate minimum set of replicas and wakeup backup ones when active ones fail
  Problem: system is unavailable when rebuilding backup replica

• Address availability problem by lazy recovery
  Rebuild a backup replica in the background
Our solution: on-demand instantiation + lazy recovery

• Reduce cost with on-demand instantiation
  Activate minimum set of replicas and wake up backup ones when active ones fail
  Problem: system is unavailable when rebuilding backup replica

• Address availability problem by lazy recovery
  Rebuild a backup replica in the background

• Challenge
  How to ensure the system is able to function correctly even when some nodes have only partial state?
Key observation: when agreement and execution are separated, they each presents a unique property that enables lazy recovery
Instant activation for agreement nodes

• Agreement: decide the next request to execute

• Observation: agreement protocol is memoryless
  A node does not need to know prior requests when agreeing on the next one
Instant activation for agreement nodes

• Agreement: decide the next request to execute

• Observation: agreement protocol is memoryless
  A node does not need to know prior requests when agreeing on the next one

Activate minimum number of agreement nodes that can reach agreement $N_{active}^A = N_{normal}^A$
Instant activation for agreement nodes

• Agreement: decide the next request to execute

• Observation: agreement protocol is memoryless
  A node does not need to know prior requests when agreeing on the next one

Activate minimum number of agreement nodes that can reach agreement $N_{active}^A = N_{normal}^A$
Instant activation for agreement nodes

- Agreement: decide the next request to execute

- Observation: agreement protocol is memoryless
  A node does not need to know prior requests when agreeing on the next one

 Activate minimum number of agreement nodes that can reach agreement $N_{active}^A = N_{normal}^A$

 Ask a blank backup node to join agreement
 Immediately
Instant activation for agreement nodes

- **Agreement:** decide the next request to execute

- **Observation:** agreement protocol is memoryless
  
  A node does not need to know prior requests when agreeing on the next one

Activate minimum number of agreement nodes that can reach agreement $N_{active}^A = N_{normal}^A$

- Ask a blank backup node to join agreement immediately
- Recover the backup node later in the background
Separating critical and flexible tasks for execution

• Execution: execute requests and other applications’ tasks
  Critical task (e.g. executing a request, sending replies to clients)
  Flexible task (e.g. taking a snapshot for garbage collection)

• Observation:
  Number of replicas required to execute critical tasks \( N_{critical}^E \) is sometimes fewer than that required to execute flexible tasks \( N_{flexible}^E \).
Separating critical and flexible tasks for execution

• Our strategy

\[ \text{Activate } N_{\text{active}}^E = \max (N_{\text{critical}}^E + f, N_{\text{flexible}}^E) \text{ nodes} \]
Separating critical and flexible tasks for execution

• Our strategy

\[ \text{Activate } N_{\text{active}}^E = \max (N_{\text{critical}}^E + f, N_{\text{flexible}}^E) \text{ nodes} \]

Can always perform critical tasks
Separating critical and flexible tasks for execution

• Our strategy
  
  Activate \( N_{active}^E = \max (N_{critical}^E + f \cdot N_{flexible}^E) \) nodes

  Can perform flexible tasks when there are no failures
Separating critical and flexible tasks for execution

• Our strategy
  
  \[ N_{active}^E = \max (N_{critical}^E + f, N_{flexible}^E) \]

\[ f = 1, \quad N_{critical}^E = 1, N_{flexible}^E = 2 \Rightarrow N_{active}^E = 2 \]
Separating critical and flexible tasks for execution

• Our strategy

\[ \text{Activate } N_{\text{active}}^E = \max (N_{\text{critical}}^E + f, N_{\text{flexible}}^E) \text{ nodes} \]

\[ f = 1, N_{\text{critical}}^E = 1, N_{\text{flexible}}^E = 2 \Rightarrow N_{\text{active}}^E = 2 \]

Can perform critical tasks, but not flexible tasks
Separating critical and flexible tasks for execution

• Our strategy
  
  \[ N^{E}_{\text{active}} = \max (N^{E}_{\text{critical}} + f, N^{E}_{\text{flexible}}) \] nodes

\[ f = 1, N^{E}_{\text{critical}} = 1, N^{E}_{\text{flexible}} = 2 \Rightarrow N^{E}_{\text{active}} = 2 \]

- Can perform critical tasks, but not flexible tasks
- Delay flexible tasks after recovery completes
Summary

• Activate a subset of agreement and execution nodes

• When an agreement node fails, ask a blank one to join agreement immediately

• When an execution node fails, keep processing requests with remaining execution nodes

• Recover nodes later in the background
Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>2f+1 / 2f+1</td>
<td>f+1 / f+1</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>u + max(u, r) + 1</td>
<td>u + r + 1</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>3f+1 / 2f+1</td>
<td>3f+1 / 2f+1</td>
</tr>
</tbody>
</table>

- **Paxos**
  \[ N^A_{normal} = f+1 \Rightarrow N^A_{active} = f+1 \]
  \[ N^E_{critical} = 1, N^E_{flexible} = f+1 \Rightarrow N^E_{active} = f+1 \]
Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>2f+1 / 2f+1</td>
<td>f+1 / f+1</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>u + max(u, r) + 1</td>
<td>u + r + 1</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>3f+1 / 2f+1</td>
<td>3f+1 / 2f+1</td>
</tr>
</tbody>
</table>

- **Paxos**
  \[
  N_{normal}^A = f + 1 \implies N_{active}^A = f + 1
  \]
  \[
  N_{critical}^E = 1, N_{flexible}^E = f + 1 \implies N_{active}^E = f + 1
  \]
## Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>2f+1 / 2f+1</td>
<td>f+1 / f+1</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>u + max(u, r) + 1</td>
<td>u + r + 1</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>3f+1 / 2f+1</td>
<td>3f+1 / 2f+1</td>
</tr>
</tbody>
</table>

- **UpRight Execution**

\[
N_{critical}^E = r + 1, \quad N_{flexible}^E = \max(u, r) + 1 \quad \Rightarrow \quad N_{active}^E = u + r + 1
\]
Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>2f+1 / 2f+1</td>
<td>f+1 / f+1</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>u + max(u, r) + 1</td>
<td>u + r + 1</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>3f+1 / 2f+1</td>
<td>3f+1 / 2f+1</td>
</tr>
</tbody>
</table>

• Zyzzyva [Kotla SOSP’07]

$$N_{normal}^A = 3f + 1 \text{ (Speculation)} \Rightarrow N_{active}^A = 3f + 1$$

$$N_{critical}^E = f + 1, N_{flexible}^E = f + 1 \Rightarrow N_{active}^E = 2f + 1$$

Our approach is not always effective
Case studies

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Original</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paxos</td>
<td>2f+1 / 2f+1</td>
<td>f+1 / f+1</td>
</tr>
<tr>
<td>UpRight Execution</td>
<td>u + max(u, r) + 1</td>
<td>u + r + 1</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>3f+1 / 2f+1</td>
<td>3f+1 / 2f+1</td>
</tr>
</tbody>
</table>

- Zyzzyva [Kotla SOSP’07]

\[
N^A_{normal} = 3f + 1 \text{ (Speculation)} \Rightarrow N^A_{active} = 3f + 1 \\
N^E_{critical} = f + 1, N^E_{flexible} = f + 1 \Rightarrow N^E_{active} = 2f + 1
\]

Our approach is not always effective
Adaptive recovery

• Challenge: how to finish recovery in a timely manner?

• Why necessary?
  
  Delayed recovery results in higher probability of data loss
  
  Long delayed flexible tasks (e.g. garbage collection) will block the system

• Our solution
  
  An adaptive approach to meet the deadline specified by the administrator
Evaluation

• Build ThriftyPaxos from scratch in Java

• Questions
  • What is the performance of ThriftyPaxos when there are no failures?
    Compare to standard Paxos
  • What is the availability of ThriftyPaxos when failures occur?
    Compare to standard Paxos and Cheap Paxos
  • Can adaptive recovery meet the deadline with different configurations?
    Use various deadlines and state sizes
Evaluation setup

• Machines
  Dell R220 with 8 cores, 16GB RAM and two hard drivers

• Evaluate replicated H2 and RemoteHashMap
  H2: database system, ran TPC-C over H2
  RemoteHashMap: benchmark application built by us

• Methodology
  To evaluate availability, kill agreement and execution nodes to emulate failures
ThriftyPaxos achieves higher throughput

ThriftyPaxos achieves 73%~88% more write throughput
Maintaining availability during failure recovery

\[ f=1, v=512b, \text{snapshot}=5G \]
Maintaining availability during failure recovery

Kill a non-leader replica

Kill the leader

Thrifty Paxos

Standard Paxos

f=1, v=512b, snapshot=5G
Maintaining availability during failure recovery

Kill a non-leader replica

Kill the leader

ThiftyPaxos

Standard Paxos

f=1, v=512b, snapshot=5G
Maintaining availability during failure recovery

f=1, v=512b, snapshot=5G
Maintaining availability during failure recovery

Kill a non-leader replica  Recover backup replicas  Kill the leader

Thrifty Paxos

Standard Paxos

Time (seconds)

Throughput (requests/s)

f=1, v=512b, snapshot=5G
Maintaining availability during failure recovery

Kill a non-leader replica

Kill the leader

Thrifty Paxos

Standard Paxos

f=1, v=512b, snapshot=5G
Maintaining availability during failure recovery

Kill a non-leader replica

Kill the leader

f=1, v=512b, snapshot=5G
Related work

• On-demand instantiation
  Cheap Paxos[Lamport DSN’04], ZZ [Wood Eurosys’11]

• Accurate failure detection
  Falcon[Leners SOSP’11], ...

• Higher read throughput
  ZooKeeper[Hunt ATC’10], Gaios[Bolosky NSDI’11], ...

• Lower latency
  Fast Paxos[Lamport DC’06], Speculative Paxos[Ports NSDI’15],
  Zyzzyva [Kotla SOSP’07], ...

• Multi-leader load balance
  Mencius [Mao OSDI’08], EPaxos [Moraru SOSP’13], ...
Conclusion

• Strong replication does not have to be expensive

• No need to invent new protocols
  • Examine conditions for correctness and availability in existing protocols
  • Combine on-demand instantiation and lazy recovery

https://github.com/vdr007/ThriftyPaxos