Defining Eventually Consistent Byzantine Fault Tolerant Services

LADIS 2008

Petr Kuznetsov  Rodrigo Rodrigues  Atul Singh  Petros Maniatis  Pedro Fonseca

MPI-SWS  Rice University  Intel Research
Overview

• Highly available services
  • Sacrifice consistency for availability
• Need for Byzantine fault tolerance (BFT)
  • Non-crash faults occur
• Weakly consistent BFT for high availability
  • Specification
  • Achilles protocol
Data Centers

• Backbone of many services
• E-commerce, email, social networks
• Emerging cloud computing infrastructures like EC2
• High availability and reliability requirements
• Used by millions of people worldwide
• Downtime => loss in revenue ($), bad press
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo \cite{SOSP07}

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

• Storage backend
• Handles lots of requests, tight SLAs
• Handles failures, e.g., partitions
Amazon.com’s Dynamo \([\text{SOSP07}]\)

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions

Fault!

Dynamo

Stale state during failures
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions

Add

Fault!

Stale state during failures
Amazon.com’s Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions

Eventually get correct state
Amazon.com's Dynamo [SOSP07]

- Storage backend
- Handles lots of requests, tight SLAs
- Handles failures, e.g., partitions

Relaxed consistency for high availability

Stale state during failures

Eventually get correct state
Fault model

![Graph showing the comparison between Crash-only and Non-crash in DB2, Oracle, and MySQL. The y-axis represents the percentage, ranging from 0 to 100. The x-axis lists the database systems: DB2, Oracle, MySQL. The bars indicate the proportion of Crash-only and Non-crash faults for each database system.]
Fault model

- Currently, assume crash-fault model
Fault model

- Currently, assume crash-fault model
- Software bugs, hardware faults
  - Bug logs of major databases [SOSP07]
  - File systems lose data [OSDI06]
  - Amazon S3 down for 7 hours: bug in gossip protocol
Fault model

• Currently, assume crash-fault model
• Software bugs, hardware faults
  • Bug logs of major databases [SOSP07]
  • File systems lose data [OSDI06]
  • Amazon S3 down for 7 hours: bug in gossip protocol
• Byzantine fault model is a better candidate
• Able to mask arbitrary faults
BFT Replication
BFT Replication

- 3f+1 replicas to tolerate f faults
BFT Replication

- **3f+1** replicas to tolerate \( f \) faults
- Lot of progress in making them practical
  - ✔ Raw performance
  - ✔ Number of execution replicas
  - ✔ Ability to tolerate more than \( f \) faults

<table>
<thead>
<tr>
<th>System</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>OSDI'99</td>
</tr>
<tr>
<td>Q/U</td>
<td>SOSP'05</td>
</tr>
<tr>
<td>BAR</td>
<td>SOSP'05</td>
</tr>
<tr>
<td>HQ</td>
<td>OSDI'06</td>
</tr>
<tr>
<td>BFT2F</td>
<td>NSDI'07</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>SOSP'07</td>
</tr>
<tr>
<td>Shepherd</td>
<td>SOSP'07</td>
</tr>
<tr>
<td>A2M</td>
<td>SOSP'07</td>
</tr>
</tbody>
</table>
Can we use BFT right away?

- Unfortunately, no :(  
- Provide strict consistency  
- Require $2f+1$ replicas to be available  
- Loses availability otherwise  
  - E.g., during partitions, may lose availability  
- Mismatch with application requirements
Weakly Consistent BFT
Weakly Consistent BFT

- Key idea: relaxed consistency for availability
- Available when others block but sometimes incorrect
Weakly Consistent BFT

• Key idea: relaxed consistency for availability
  • Available when others block but sometimes incorrect

• Specification of a weakly consistent BFT service
  • Form of eventual consistency, inspired by Dynamo
  • Achilles: a weakly consistent BFT protocol
Weakly Consistent BFT

• Key idea: relaxed consistency for availability
  • Available when others block but sometimes incorrect
• Specification of a weakly consistent BFT service
  • Form of eventual consistency, inspired by Dynamo
  • Achilles: a weakly consistent BFT protocol
• Grows the class of applications that could benefit from BFT
Eventual Consistency
Intuition of eventual consistency

Client A
- a1 (add item to cart)

Client B
- b1
- b2 (read items)

Client C
- c1
- c2
- c3
Intuition of eventual consistency

Service state modeled as partial order on ops

Client A

a1 (add item to cart)

Client B

b1

Client C

b2 (read items)

c1
c2
c3
Intuition of eventual consistency

Service state modeled as partial order on ops

Committed History
Intuition of eventual consistency

Service state modeled as partial order on ops

Committed History — Tentative History
Intuition of eventual consistency

- **Client A**: a1 (add item to cart) → a2 → a3
- **Client B**: b1 → b2 (read items)
- **Client C**: c1 → c2 → c3

Service state modeled as partial order on ops

- **Committed History**: Representation of committed transactions
- **Tentative History**: Representation of tentative transactions

Atul Singh
Intuition of eventual consistency

Service state modeled as partial order on ops

Client A: a1 (add item to cart) → a2 → a3
Client B: b1 (add item to cart) → b2 (read items)
Client C: c1 → c2 → c3
Intuition of eventual consistency

- **Client A**
  - a1 (add item to cart)

- **Client B**
  - b1
  - b2 (read items)

- **Client C**
  - c1
  - c2
  - c3

- Service state modeled as partial order on ops
Intuition of eventual consistency

Client A

a1 (add item to cart)

b1

Client B

b2 (read items)

c2

Client C

c1

c3

Service state modeled as partial order on ops

Committed History (Linearizable)
Two kind of replies

Client A
a1 (add item to cart)

Client B
b1
b2 (read items)

Client C
c1
c2
c3

Committed History

Tentative History
Two kind of replies

- **Strong and weak replies**
- Different consistency for different reply types
Two kind of replies

- **Strong** and **weak** replies
  - Different consistency for different reply types
- **Weak replies**
  - Observe eventual consistency
  - May observe incorrect (but not arbitrary!) state
  - May miss previous operations
  - Eventually gets committed
Strong reply

Client A
- a1 (add item to cart)
- a2
- a3

Client B
- b1
- b2 (read items)

Client C
- c1
- c2
- c3

Committed History
Tentative History
Strong reply

- Observe committed history
- Example: a1, b1, c1

Client A

Client B

Client C

a1 (add item to cart)  a2  a3

b1

b2 (read items)

c1

c2

c3

Committed History

Tentative History
Strong reply

- Observe committed history
  - Example: a1, b1, c1

- Two variations possible
  1. Observe **some** weak operations
     - Useful for checkout op in shopping cart
  2. Observe **all** previous weak operations
     - Updating inventories once partition heals
     - Stronger requirements
Merge Operation

- **Merge**: unifies concurrent states
- Triggered when partition heals

Diagram:

Client A
- \( a_1 \) (add item to cart)
- \( a_2 \)
- \( a_3 \)

Client B
- \( b_1 \)
- \( b_2 \) (read items)

Client C
- \( c_1 \)
- \( c_2 \)
- \( c_3 \)
Merge Operation

- **Merge**: unifies concurrent states
- Triggered when partition heals
Merge Operation

- **Merge**: unifies concurrent states
- Triggered when partition heals
Merge Operation

- **Merge**: unifies concurrent states
- Triggered when partition heals
- May need to resolve conflicts
  - Syntactic
  - Semantic (application specific)
  - Shopping cart takes union of two sets [Dynamo]
Merge Operation

- **Merge**: unifies concurrent states
- Triggered when partition heals
- May need to resolve conflicts
  - Syntactic
  - Semantic (application specific)
  - Shopping cart takes union of two sets [**Dynamo**]

<table>
<thead>
<tr>
<th>Event</th>
<th>Client A</th>
<th>Client B</th>
<th>Client C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Achilles
Achilles: a simple run

Strong reply
Achilles: a simple run
Achilles: a simple run

Strong reply
Achilles: a simple run

Strong reply

Strong reply
Achilles: a simple run

Strong reply

Weak reply

Strong reply
Achilles: a simple run

Strong reply
Weak reply
Strong reply

1
2
3
4
Achilles: a simple run

Strong reply

Weak reply

Strong reply
Achilles: a simple run
Achilles: a simple run

Strong reply

Weak reply

Weak reply

Strong reply
Achilles: a simple run

Merge Concurrent Histories

Weak View
Achilles: key insights

- Extensions to Zyzzyva [SOSP07]
- Smaller quorums
- Extended view change machinery for merge
- Clients send sequential requests
- Checkpoint state increases
Evaluation Plan

- Synthetic workload
  - TPC-C: industry standard used to measure e-commerce performance
  - SpecWeb2005: similar kind
- Fault trace
  - Partitions: frequency and duration from PlanetLab
Conclusions

- Highly available services provide relaxed consistency
- Need for Byzantine fault tolerance
- Traditional BFT protocols do not favor availability
- We have built a highly available BFT protocol