Eventually Consistent Distributed Ledger Relying on Degraded Atomic Broadcast

Grégory Bénassy, Fukuhito Ooshita, Michiko Inoue

Nara Institute of Science and Technology (NAIST)
Télécom SudParis
Contents

• Objective
• Distributed Ledger Object
• Related Work
• Proposed Algorithm
  – Distributed ledger under degraded atomic broadcast
• Consideration & Conclusion
Objective

• Distributed ledger (Blockchain)
  – Technology for cryptocurrency
    • Bitcoin, Ethereum

• Attacks to distributed ledger
  – DDOS attack, routing attack, eclipse attack

• Our objective
  – Distributed ledger resilient to some attacks
Attacks

L: Distributed ledger

L.Get() sequence of records

L.Append(r) ACK

Clients

P2P server network

Eclipse attack

Routing attack

Internet

AS

BGP

AS
Distributed Ledger

• Technology for cryptocurrency
    • Maintained in P2P system without central authority
  – Explosive growth in number and applications
    • Development first
    • Sometimes, “the code is the spec.”

• Formalization
  – Define distributed ledger as a shared object [2]

• Shared object
  – Multiple clients can access concurrently
  – Maintains a sequence of records
• Operations
  – Get: return a sequence of records
  – Append: append a record to the sequence
• Operation consists of two events
  – invocation, response

• Sequential Specification
  – Condition for a sequence of operations
    • “Get()” returns a sequence of records
    • “Append(r)” appends a record r

• Eventually Consistency
  – Condition for a history (sequence of events)
• Eventually Consistency (intuitive definition)
  – Operations can be permutated to satisfy sequential specification
  – For every Append(r), eventually, all Get() return sequences containing r
• Implement distributed ledger object [2]
  – using an atomic broadcast service
  – at most $f$ crash servers

Atomic Broadcast

- **Validity**: a correct server broadcasts a message, the server will deliver it.
- **Uniform Agreement**: a correct server delivers a message, all the correct server will deliver it.
- **Uniform Integrity**: a message is delivered at most once if it is broadcasted.
- **Uniform Total Order**: if some servers deliver two messages \( m \) and \( m' \), they deliver those messages in the same order.
Our Contribution

• Implement distributed ledger object under degraded atomic broadcast
  – Uniform agreement is not guaranteed
    Case A: One victim server at a time
    Case B: Multiple victim servers at a time
• Distributed ledger using atomic broadcast
  – at most $f$ crash faults

```
Code for client

function L.Get()
    c ← c + 1
    send request (c, Get) to $f + 1$ or more servers
    wait response (c, GetRes, V) from some server
    return V

function L.Append(r)
    c ← c + 1
    send request (c, Append) to $f + 1$ or more servers
    wait response (c, AppendRes, ACK) from some server
    return ACK

Code for server

Seq ← 0
if receive request (c, Get) from client p
    send response (c, GetRes, Seq) to p
if receive request (c, Append, r) from client p
    ABroadcast(r)
    send response (c, AppendRes, ACK) to p
if ADeliver(r)
    if $r \notin$ Seq
        Seq ← Seq $\parallel$ r // concatenation
```

**Append(r)**
- Clients: request \( (c, \text{Append}, r) \)
- Servers: request \( (c, \text{Append}, r) \)
  - Send more than \( f \) servers
  - \( c \leftarrow c+1 \)
- Clients: return ACK
- Servers: return \( \text{ACK} \)
  - if \( r \notin Seq \)
  - \( Seq \leftarrow Seq \parallel r \) (concatenation)

**Get()**
- Clients: request \( (c, \text{GET}) \)
- Servers: request \( (c, \text{GET}) \)
  - Send more than \( f \) servers
  - \( c \leftarrow c+1 \)
- Clients: return Seq
- Servers: return Seq
  - response \( (c, \text{AppendRes}, Seq) \)
  - response \( (c, \text{AppendRes}, Seq) \)

---

"Outgrow your limits"
Case A: One Victim Server at a Time

- At most $f$ crash servers
- At most one victim server could not deliver a message at a time

Code for client
\[
c \leftarrow 0
\]
\[
\text{function } L.\text{Get}()
\]
\[
c \leftarrow c + 1
\]
\[
\text{send} \text{ request } (c, \text{Get}) \text{ to } f + 3 \text{ or more servers}
\]
\[
\text{wait} \text{ response } (c, \text{GetRes}, V_i) \text{ from } 3 \text{ servers}
\]
\[
\text{return} \text{ combine}(V_1, V_2, V_3)
\]

\[
\text{function } L.\text{Append}(r)
\]
\[
c \leftarrow c + 1
\]
\[
\text{send} \text{ request } (c, \text{Append}) \text{ to } f + 1 \text{ or more servers}
\]
\[
\text{wait} \text{ response } (c, \text{AppendRes, ACK}) \text{ from some server}
\]
\[
\text{return} \text{ ACK}
\]

\[
\text{function} \text{ combine}(V_1, V_2, V_3)
\]
\[
V \leftarrow [V_1, V_2, V_3]
\]
\[
\text{Seq} \leftarrow \emptyset
\]
\[
\text{ind}_1, \text{ind}_2, \text{ind}_3 \leftarrow 0
\]
\[
\text{while } V \neq \emptyset \text{ do}
\]
\[
\text{RecordList} = [V_1(\text{ind}_1), V_2(\text{ind}_2), V_3(\text{ind}_3)]
\]
\[
\text{if } r \text{ appears at least twice in RecordList}
\]
\[
\text{Seq} \leftarrow \text{Seq} \parallel r
\]
\[
\text{else}
\]
\[
\text{break}
\]
\[
\text{for } i \leftarrow 1 \text{ to } 3 \text{ do}
\]
\[
\text{while } V_i(\text{ind}_i) \in \text{Seq}
\]
\[
\text{ind}_i \leftarrow \text{ind}_i + 1
\]
\[
\text{if } \text{ind}_i = |V_i|
\]
\[
\text{remove } V_i \text{ from } V
\]
\[
\text{return} \text{ Seq}
\]
• Client combines 3 sequences of records by a majority vote

\[
\begin{array}{cccccccc}
V_1 & r1 & r2 & r4 & r3 & r5 & r6 & r7 & r8 \\
V_2 & r1 & r2 & r3 & & r5 & r6 & r4 & r7 & r8 \\
V_3 & r1 & r2 & r3 & r4 & & r5 & r6 & & r8 \\
Seq & r1 & r2 & r3 & r4 & r5 & r6 & r7 & r8 \\
\end{array}
\]
Case B: At Most $m$ Victim Server at a Time

- At most $f$ crash servers
- At most $m$ victim servers could not deliver a message

**Code for client**

```plaintext
Case A
Code for client
c ← 0
function L.Get()
c ← c + 1
send request (c, Get) to $f+3$ or more servers
wait response (c, GetRes, $V_i$) from 3 servers
return combine($V_1, V_2, V_3$)

function L.Append(r)
c ← c + 1
send request (c, Append) to $f+1$ or more servers
wait response (c, AppendRes, ACK) from some server
return ACK
```

```plaintext
Case B
Code for client
c ← 0
function L.Get()
c ← c + 1
send request (c, Get) to $f + 2m +1$ or more servers
wait response (c, GetRes, $V_i$) from $2m+1$ servers
return combine($V_1, V_2, V_3, \ldots, V_{2m+1}$)

function L.Append(r)
c ← c + 1
send request (c, Append) to $f+1$ or more servers
wait response (c, AppendRes, ACK) from some server
return ACK
```
• Case B: at most $m$ victim servers
  – Algorithm requires at least $f + 2m + 1$ servers
    $f$: crash servers, $m$: victim servers

$ f = 2, m = 2 $
• Possible to reduce the number of servers?
  – Consideration
    • Maintain a correct sequence of records in each server
      – Exchange a sequence of records among servers
      – Inform a missing record

\[
\text{ABroadcast} (r, \text{Seq}) \rightarrow \text{ABroadcast} (r', \text{Seq})
\]

\[
\text{Modify}, \text{Seq} || r || r'
\]
• Case C: At least one server maintains the correct sequence
  – Possible to achieve eventually consistency?
    • No!

Counterexample:
Conclusion

• Implemented distributed ledger object under degraded atomic broadcast
  – Uniform agreement is not guaranteed
    Case A: One victim server at a time
    Case B: Multiple victim servers at a time

• Future works
  – Required number of servers
  – Other faults