IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill
Microsoft Research

Presenter: Armin Vakil
Abstract

The Paxos algorithm, when presented in plain English, is very simple.
Abstract

The Paxos algorithm, when presented in plain English, is very simple.
Abstract

The Paxos algorithm, when presented in plain English, is very simple.

Implementing correct distributed protocols is notoriously difficult.
Examples of bug disruptions
Examples of bug disruptions:

Facebook Explains October 4 Outage

Facebook apologizes for second outage in a week, services back up

Amazon: Here's what caused the major AWS outage last week

AWS explains how adding a small amount of capacity to Kinesis servers knocked out dozens of services for hours.

Amazon Web Services dashboard goes temporarily offline

TECH - AMAZON WEB SERVICES

BY CHRIS MORRIS

October 12, 2021 12:27 PM EDT
Verification to the Rescue
Verification to the Rescue

• Automated proof for correctness
Verification to the Rescue

• Automated proof for correctness
Contributions
Contributions

- First mechanically-checked proof of implementation
Contributions

• First mechanically-checked proof of implementation
Contributions

• First mechanically-checked proof of implementation
Contributions

• First mechanically-checked proof of implementation
• IronRSL - Replicated state library
Contributions

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV - Sharded Key-Value Store
Contributions

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV - Sharded Key-Value Store
- Liveness proof along with safety
Contributions

- First mechanically-checked proof of implementation
- IronRSL - Replicated state library
- IronKV - Sharded Key-Value Store
- Liveness proof along with safety
- Reasonable proof-to-code ratio
Outline

• Safety
• Parallelism
• Liveness
• Evaluation
• Conclusion
How verification works?

Implementation
How verification works?

- Specification
- Implementation
How verification works?

Specification

≡

Implementation
How verification works?

Specification

= 

Implementation

What about specification?
Example: Hash Table
Example: Hash Table
Example: Hash Table
Example: Hash Table

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
</tr>
</tbody>
</table>
Example: Hash Table

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Example: Hash Table
Example: Hash Table

```
K  V
-- --
-- --
-- --
-- --

K  V
-- --
-- --
--- A
--- 6

K  V
-- --
-- --
--- A
--- 6

K  V
-- --
-- --
--- B
--- 11

K  V
--- C
--- 1

K  V
--- A
--- 6

K  V
--- B
--- 11
```
Example: Hash Table
Example: Hash Table

- 6->A

K | V
---|---
- | -
- | -
- | -
- | -

6 | A

11->B

6 | A

1 | C

6 | A

11 | B

11 | B

6 | A

11 | B
Sharded Hash Table
Sharded Hash Table

<table>
<thead>
<tr>
<th>Host 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host 2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Sharded Hash Table

Host 1

K | V
---|---
- | -
- | -

Host 2

K | V
---|---
- | -
- | -

Host 1

K | V
---|---
- | -
- | -

Host 2

K | V
---|---
- | -
6 | A
- | -
Sharded Hash Table
# Sharded Hash Table

<table>
<thead>
<tr>
<th>Host 1</th>
<th>Host 2</th>
<th>Host 1</th>
<th>Host 2</th>
<th>Host 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>V</td>
<td>K</td>
<td>V</td>
<td>K</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>6</td>
<td>A</td>
<td>11</td>
</tr>
</tbody>
</table>

- Host 1: K, V
- Host 2: K, V
- Host 1: K, V, 6, A
- Host 2: K, V, 6, A, 11, B
Sharded Hash Table
Sharded Hash Table

Host 1

K | V
---|---
6 | A

Host 2

K | V
---|---
6 | A

Host 1

K | V
---|---
6 | A

Host 2

K | V
---|---
11 | B

Host 1

K | V
---|---
6 | A

Host 2

K | V
---|---
11 | B
Sharded Hash Table

Host 1

- - -

K V

- - -

Host 2

- - -

K V

6 A

- -

Host 1

- - -

K V

6 A

11 B

Host 2

- - -

K V

- -

11 B

Host 1

- - -

K V

6 A

- -

Host 2

- - -

K V

- -

11 B
Sharded Hash Table
Sharded Hash Table
Sharded Hash Table

Host 1

K | V
---|---
- | -
- | -

Host 2

K | V
---|---
- | -
- | -

6 | A
---|---
11 | B
---|---
- | -
Sharded Hash Table

Host 1

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Host 2

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Host 1

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Host 2

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
**Sharded Hash Table**

Each host communicates with the next host in the cycle, which is determined by the hash of the key. The hash table is distributed across hosts, with keys mapped to values.

- **Host 1**
  - Key: 6, Value: A
  - Key: 11, Value: B

- **Host 2**
  - Key: 6, Value: A
  - Key: 11, Value: B
Sharded Hash Table

Host 1

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
</tr>
</tbody>
</table>

Host 2

<table>
<thead>
<tr>
<th>K</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
</tr>
</tbody>
</table>
Sharded Hash Table
Sharded Hash Table
Sharded Hash Table
Induction
Induction

Base
Induction

Base
Induction

Base

v
Induction

Base

v

v′
Induction

Base

v

v′
Refinement Proof by Induction (Base)
Refinement Proof by Induction (Base)
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
ensures Safe(v) // Safe base case
Refinement Proof by Induction (Base)

function Abstraction(L:Variables) : H.Variables
predicate Safe(v:Variables)

lemma RefinementInit(v:Variables)
requires Init(v)
ensures Safe(v) // Safe base case
ensures MapSpec.Init(Abstraction(v)) // Refinement base case
Refinement Proof by Induction (Step)
Refinement Proof by Induction (Step)
Refinement Proof by Induction (Step)

lemma RefinementNext(v:Variables, v':Variables)
lemma RefinementNext(v:Variables, v':Variables)
  requires Next(v, v')
Refinement Proof by Induction (Step)

lemma RefinementNext(v:Variables, v':Variables)
  requires Next(v, v')
  requires Safe(v)
Refinement Proof by Induction (Step)

lemma \texttt{RefinementNext}(v:\text{Variables}, v':\text{Variables})
    requires \texttt{Next}(v, v')
    requires \texttt{Safe}(v)
    ensures \texttt{Safe}(v')  // Safe inductive step
lemma RefinementNext(v:Variables, v':Variables)
  requires Next(v, v')
  requires Safe(v)
  ensures Safe(v')  // Safe inductive step
  ensures MapSpec.Next(Abstraction(v), Abstraction(v'))
Refinement Proof by Induction (Step)

lemma RefinementNext(v:Variables, v':Variables)
  requires Next(v, v')
  requires Safe(v)
  ensures Safe(v')  // Safe inductive step
  ensures MapSpec.Next(Abstraction(v), Abstraction(v'))
    || Abstraction(v) == Abstraction(v')
Other implementation subtleties
Other implementation subtleties

Subtleties of distributed protocols
Other implementation subtleties

- Subtleties of distributed protocols
- Implementation difficulties
Other implementation subtleties

- Subtleties of distributed protocols
- Maintaining safety invariants
- Implementation difficulties
Other implementation subtleties

- Subtleties of distributed protocols
- Maintaining safety invariants
- Concurrent Hosts
- Implementation difficulties
Other implementation subtleties

- Subtleties of distributed protocols
- Concurrent Hosts
- Implementation difficulties
- Maintaining safety invariants
- Liveness
Other implementation subtleties

- Subtleties of distributed protocols
- Concurrent Hosts
- Implementation difficulties
- Maintaining safety invariants
- Liveness
- Memory management
Other implementation subtleties

- Subtleties of distributed protocols
- Concurrent Hosts
- Implementation difficulties
- Efficient Data structures
- Maintaining safety invariants
- Liveness
- Memory management
Other implementation subtleties

- Subtleties of distributed protocols
- Concurrent Hosts
- Implementation difficulties
- Efficient Data structures
- Maintaining safety invariants
- Liveness
- Memory management
- Integer overflow
Other implementation subtleties

- Subtleties of distributed protocols
- Concurrent Hosts
- Implementation difficulties
- Efficient Data structures
- Maintaining safety invariants
- Liveness
- Memory management
- Integer overflow
Two-level Refinement
Two-level Refinement

I0 → I1 → I2 → I3

Impl.
Two-level Refinement

Protocol

Impl.
Two-level Refinement

Protocol

Impl.
Two-level Refinement

Spec.

Protocol

Impl.
Two-level Refinement

Spec.

Protocol

Impl.
Outline

• Safety
• Parallelism
• Liveness
• Evaluation
• Conclusion
Parallelism between hosts

Host A

Host B

Compute

Receive  Send
Parallelism between hosts

Host A

Host B

Compute

Receive

Send
Parallelism between hosts

Host A

Host B

Compute

Receive  Send
Parallelism between hosts

Host A

Host B

Compute

Receive

Send
Parallelism between hosts

Reasoning about concurrency is extremely difficult.
Enormous number of possible interleaving
Enforce all receives precede sends
Enforce all receives precede sends
Enforce all receives precede sends

Host A

Host B

Compute

Receive  Send
Enforce all receives precede sends

Assume all steps are atomic
Enforce all receives precede sends

Assume all steps are atomic
Enforce all receives precede sends

Why???

Assume all steps are atomic
Outline

• Safety
• Parallelism
• Liveness
• Evaluation
• Conclusion
Liveness

C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_n
Liveness

C₀ → C₁ → C₂ → ... → Cₙ

Client sends request
Liveness

- Client sends request
- Replica receives request
Liveness

Client sends request

Replica receives request

Replica suspects leader
Liveness

Client sends request

Replica receives request

Replica suspects leader

Leader election starts

C0 → C1 → C2 → \ldots → Cn
Liveness

Client sends request

Replica receives request

Replica suspects leader

Leader election starts

Client gets a response
Main Event Handler

```csharp
method Main()
{
    var s: ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```
Main Event Handler

```csharp
method Main()
{
    var s:ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```

- Event Handler runs infinitely often
Main Event Handler

```
method Main()
{
    var s:ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```

- Event Handler runs infinitely often
Main Event Handler

```csharp
method Main()
{
    var s: ImplState;
    s := ImplInit();
    while (true) {
        s := EventHandler(s);
    }
}
```

- Event Handler runs infinitely often
- Each action runs infinitely often, too
Outline

• Safety
• Parallelism
• Liveness
• Evaluation
• Conclusion
Proof-to-code ratio

Lines of Code

<table>
<thead>
<tr>
<th>Common Libraries</th>
<th>IronRSL</th>
<th>IronKV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spec</td>
<td>Impl</td>
<td>Proof</td>
</tr>
<tr>
<td>Safety</td>
<td>Liveness</td>
<td></td>
</tr>
</tbody>
</table>

- Safety
- Liveness
Proof-to-code ratio

Lines of Code

Common Libraries

IronRSL

IronKV

Safety
Liveness

5:1
Proof-to-code ratio

Lines of Code

<table>
<thead>
<tr>
<th>Spec</th>
<th>Impl</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>Liveness</td>
<td></td>
</tr>
<tr>
<td>Common Libraries</td>
<td>IronRSL</td>
<td>IronKV</td>
</tr>
</tbody>
</table>

Is 5:1 small??

5:1
Proof-to-code ratio

Lines of Code

Common Libraries

IronRSL

IronKV

Safety

Liveness

5:1

Is 5:1 small??
Proof-to-code ratio

Lines of Code

Common Libraries

IronRSL

IronKV

Spec

Impl

Proof

Safety

Liveness

5:1

8:1

Is 5:1 small??

Proof - to - code ratio
Proof-to-code ratio

Number of spec (unverified) lines of code is small.

Is 5:1 small??

Common Libraries

IronRSL

IronKV

Lines of Code

Spec  Impl  Proof  Spec  Impl  Proof  Spec  Impl  Proof

Safety

Liveness

5:1

5:1

5:1
IronRSL Performance

Maximum throughput (RPS)

With Batching

Without Batching

IronRSL
Baseline (EPaxos)
IronRSL Performance

Maximum throughput (RPS)

IronRSL
Baseline (EPaxos)

With Batching

Without Batching

2.4x
IronKV Performance

Peak throughput (kRPS)

- IronKV
- Redis

<table>
<thead>
<tr>
<th></th>
<th>Get</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>128B</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>1KB</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>8KB</td>
<td>35</td>
<td>25</td>
</tr>
</tbody>
</table>
IronKV Performance

Peak throughput (kRPS)

IronKV

Redis

1.5x
IronKV Performance

Peak throughput (kRPS)

<table>
<thead>
<tr>
<th></th>
<th>128B</th>
<th>1KB</th>
<th>8KB</th>
<th>128B</th>
<th>1KB</th>
<th>8KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td></td>
<td></td>
<td></td>
<td>2.9x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• First mechanically-checked proof of implementation
• Liveness proof along with safety
• Reasonable proof-to-code ratio
• Comparable performance with state-of-the-art
Conclusion

• First mechanically-checked proof of implementation
• Liveness proof along with safety
• Reasonable proof-to-code ratio
• Comparable performance with state-of-the-art

Thanks!