Argosy: Verifying layered storage systems with recovery refinement

Tej Chajed, Joseph Tassarotti, Frans Kaashoek, Nickolai Zeldovich

MIT
Bob writes a replication system
Bob writes a replication system
Bob writes a replication system.
Bob writes a replication system
Bob writes a replication system and implements its recovery procedure.
Bob writes a replication system and implements its recovery procedure.
Bob is careful and writes a machine-checked proof of correctness.

Read and write are atomic if you run rep_recover after every crash.
Transactions

write-ahead logging

... log_recover

Disk interface
Transactions

write-ahead logging

... log_recover

Disk interface

√ ops are atomic if you run log_recover after every crash
Transactions

write-ahead log

Disk interface

replication

Two-disk interface

logging + replication?
Transactions

- write-ahead log

Disk interface

- replication

Two-disk interface

logging + replication

rep_recover; log_recover
Challenge: crashes during composed recovery

rep_recover ✓ under crashes
log_recover ✓ under crashes

rep_recover ; log_recover  how do we prove correctness under crashes using the existing proofs?
Prior work cannot handle multiple recovery procedures

- **CHL** [SOSP ’15] not modular
- **Yggdrasil** [OSDI ’16] single recovery
- **Flashix** [SCP ’16] restricted recovery procedures
Argosy supports modular recovery proofs

Transactions

write-ahead log

Disk interface

replication

Two-disk interface
Argosy supports modular recovery proofs
Contributions

Recovery refinement for modular proofs
Contributions

Recovery refinement for modular proofs

see paper CHL for proving recovery refinement

see paper Verified example: logging + replication
Contributions

Recovery refinement for modular proofs

see paper CHL for proving recovery refinement

see paper Verified example: logging + replication

see code Machine-checked proofs in Coq
Preview: recovery refinement

1. Normal execution correctness using refinement
2. Crash and recovery correctness using recovery refinement
Refinement
Disk interface

replication

Two-disk interface
Disk interface

replication

write

write_impl

Two-disk interface

write_1
write_2
Disk interface

write

replication

write_impl

Two-disk interface

write_1  write_2
correctness is based on how we use replication: run code using Disk interface on top of two disks
Correctness: trace inclusion

Disk interface  \rightarrow^{\text{replication}} \rightarrow \text{Two-disk interface}

\text{code implemented by code_impl} \supseteq \text{spec's behaviors}

\supseteq \text{running code's behaviors}
Proving correctness with an abstraction relation

1. developer provides abstraction relation $R$
Proving correctness with an abstraction relation

1. developer provides abstraction relation R
Proving correctness with an abstraction relation

1. developer provides abstraction relation $R$
2. prove spec execution exists

[Diagram showing logical disk, spec state, and write operations]
Proving correctness with an abstraction relation

1. developer provides abstraction relation $R$
2. prove spec execution exists
3. and abstraction relation is preserved
Recovery refinement
Disk interface

- read
- write

replication

- read_impl
- write_impl

Two-disk interface

- write\textsubscript{1}
- write\textsubscript{2}
- read\textsubscript{1}
- read\textsubscript{2}
Disk interface

- read
- write

replication

- read_impl
- write_impl

Two-disk interface

- write\(_1\)
- write\(_2\)
- read\(_1\)
- read\(_2\)
- rep_recover

read
write
write
1
write
2
read
rep_recover
Extending trace inclusion with recovery

- Disk interface
- Two-disk interface
- Replication

- Code
- Code_impl

- Specification for crash behavior
- Crash & recovery behavior
Extending trace inclusion with recovery

Disk interface

Two-disk interface

replication

code

⊇

code_impl

specification for crash behavior

⊇

crash & recovery behavior

crash semantics

recover

recovery semantics
Disk interface

replication

Two-disk interface

code

⊇

code_impl

one of these

<table>
<thead>
<tr>
<th>op_1</th>
<th>op_1 op_2</th>
</tr>
</thead>
</table>

_recovery semantics

? recover

code

 |--

code

 |--

crash & recovery behavior
replication

Disk interface

Two-disk interface

code ⊇ code_impl

\[
\begin{align*}
\text{code} & := \text{op}_1 \ | \ \text{op}_1 \text{op}_2 \ | \ \ldots \\
\text{crash & recovery behavior} & \subseteq \text{recover}
\end{align*}
\]

recovery semantics
Disk interface

replication

Two-disk interface

code

⊇

code_impl

code

⊇

code_impl

recover
Disk interface

replication

Two-disk interface

code

⊇

code_impl

code

⊇

code_impl

recover

zero-or-more iterations

recover
Disk interface

Two-disk interface

replication

replication

code

⊇

code_impl

recover

code

⊇

code_impl

recover

recover
Trace inclusion, with recovery

Disk interface

replication

Two-disk interface

code

⊇

code_impl

code

⊇

code_impl

recover

recover

⋆
Proving trace inclusion, with recovery
Proving trace inclusion, with recovery

- `op1_impl` → `op2_impl` → `recover` → `recover`

- Crash must occur during some operation
Proving trace inclusion, with recovery
Proving trace inclusion, with recovery
Proving trace inclusion, with recovery
Proving trace inclusion, with recovery
Recovery refinement

non-crash execution

\[ \text{op_impl} \]

\[ \text{op} \]

\[ \text{R} \]

\[ \text{R} \]

crash and recovery execution

\[ \text{op_impl} \]

\[ \text{op} \]

\[ \text{recover} \]

\[ \text{recover} \]

\[ \text{op_impl} \]

\[ \text{R} \]

\[ \text{R} \]
Recovery refinement

non-crash execution

\[ \text{op_impl} \rightarrow \text{op} \rightarrow \text{op_impl} \]

crash and recovery execution

\[ \text{op_impl} \rightarrow \text{op} \rightarrow \text{recover} \rightarrow \text{recover} \]

**Trace inclusion**

specification behavior \( \supseteq \) running code behavior
Composition theorem
Kleene algebra for transition relations

expression

\[ op_1 \mid op_2 \]

\[ r \star \]
Kleene algebra for transition relations

<table>
<thead>
<tr>
<th>expression</th>
<th>matching transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{op}_1 \</td>
<td>\ \text{op}_2$</td>
</tr>
<tr>
<td>$\text{op} \</td>
<td>\ \text{op}$</td>
</tr>
</tbody>
</table>
| $r \star$ | $r \rightarrow r \rightarrow r \rightarrow$ |...
Theorem: recovery refinements compose
Theorem: recovery refinements compose

If

Transactions

write-ahead log

replication

Disk interface

then

Transactions

logging + replication

Two-disk interface
Goal: prove composed recovery correct

- `rep_recover` under crashes
- `log_recover` under crashes
- `rep_recover ; log_recover`
Goal: prove composed recovery correct

rep_recover
rep
log
log_recover
rep ; log

☑️ under crashes
☑️ under crashes

?
rep ⋆ rep
log ⋆ log
\[
\log \star \text{rep} \star \log \star \text{rep} \star \log \\
(\text{rep} \star \log \star \text{rep} \star \log)^\star \text{rep} \star \log
\]
how to re-use recovery proofs here?
Using Kleene algebra for reasoning

\[(\text{rep} \mid \text{rep log})^* \text{ rep log}\]
Using Kleene algebra for reasoning

\[(\text{rep} \mid \text{rep} \text{log} \text{rep})^* \text{log} \text{rep}\]

after de-nesting \((p \mid q)^* = p^*(qp^*)^*\)
Using Kleene algebra for reasoning

\[(\text{rep } \log \mid \text{rep } \log)^* \text{rep } \log\]

after de-nesting \((p \mid q)^* = p^*(qp^*)^*\)

\[= \text{rep } \log^* (\text{rep } \log \mid \text{rep } \log)^* \text{rep } \log\]
Using Kleene algebra for reasoning

\[
(\text{rep} \text{log} | \text{rep} \text{log})^* \text{rep} \text{log}
\]

after de-nesting \((p | q)^* = p^*(qp)^*\)

= \text{rep} \text{log}^* (\text{rep} \text{log} \text{rep} \text{log})^* \text{rep} \text{log}

after sliding \((pq)^* p = p(qp)^*\)

= \text{rep} \text{log}^* \text{rep} (\text{log} \text{rep} \text{log})^* \text{rep} \text{log}
After rewrite both proofs apply
After rewrite both proofs apply

rep invariants restored

replication proof
After rewrite both proofs apply

rep invariants restored

behaves like

replication proof
After rewrite both proofs apply

rep invariants restored

behaves like

log invariants restored

replication proof

write-ahead log proof
Argosy is implemented and verified in Coq

3,200 lines for framework

4,000 lines for verified example (logging + replication)

Example extracts to Haskell and runs

github.com/mit-pdos/argosy
Argosy: modular proofs of layered storage systems
Argosy: modular proofs of layered storage systems

Kleene algebra

$(\text{rep} \mid \text{rep log})^*$
Argosy: modular proofs of layered storage systems

Kleene algebra

recovery refinement

\((\text{rep} \oplus \text{op} \mid \text{rep} \oplus \log \oplus \text{op})^*\)

\(\text{impl} \oplus \text{op} \mid \text{r} \oplus \text{op}^* \mid \text{r} \)
Argosy: modular proofs of layered storage systems

Kleene algebra

recovery refinement

modular proofs

(impl ▶ op ▶ rep ▶ rep log ▶ r ▶ r)^*
Argosy: modular proofs of layered storage systems

Kleene algebra

recovery refinement

modular proofs

(\text{rep} \oplus \text{rep} \log \oplus )^*

come find us after!

Tej and Joe