The **SNOW** Theorem
and Latency-Optimal
Read-Only Transactions

Haonan Lu*,
Christopher Hodsdon*, Khiem Ngo*,
Shuai Mu†, Wyatt Lloyd*

*University of Southern California, †New York University
Huge Web Services Shard Data

Massive amount of data

→ must be distributed across servers

Reads dominate the workloads
– need to be as fast as possible!
Simple Reads Are Insufficient

- **Web Clients**
- **Datacenter**
- **Storage Tier**
  - **Servers**
    - ACL=public
    - Photo=A

**Load Page** → **Web Clients** → **Datacenter** → **Storage Tier** (Read: Public)

Client reads public data directly from the storage tier without involving the web tier.
Simple Reads Are Insufficient
Simple Reads Are Insufficient

- **Web Clients**: Load Page
- **Datacenter**: Read
- **Storage Tier**: Servers
  - ACL=private
  - Photo=B

Private Photo B
Done

Public Set "Private"
Done

Photo B
Done
Simple Reads Are Insufficient
Simple Reads Are Insufficient

Web
Clients

Datacenter

Storage Tier
Servers

Load Page

ACL=private

Public

Read

Photo B

Read

Done

Photo B

Not Acceptable!

Photo B
Read-Only Transactions

• Transactions that do not modify data

• Consistently read data across servers
The Power of Read-Only Txn

• Consistency restricts what can be read
  – Eliminates unacceptable combinations

• Compatibility enables write transactions
  – Write transactions atomically update data

• Higher power \(\rightarrow\) more useful
  – Stronger consistency \(\rightarrow\) higher power
  – Compatibility \(\rightarrow\) higher power
Intuitive Tension

High Power • Reduces anomalies (the ACL – Photo example)
• Easier to reason about

Low Latency • Better user experience
• Higher revenue

Our study proves: highest power + lowest latency is impossible
Intuitive Tension

High Power vs. Low Latency

- Reduces anomalies (the ACL – Photo example)
- Easier to reason about
- Better user experience
- Higher revenue

Our study proves: highest power + lowest latency is impossible
Fundamental Tradeoff

High Power vs Low Latency

- Reduces anomalies (the ACL – Photo example)
- Easier to reason about
- Better user experience
- Higher revenue

Our study proves: highest power + lowest latency is impossible
The SNOW Properties

[S]trict serializability

[N]on-blocking operations

[O]ne response per read

[W]rite transactions that conflict
The SNOW Properties

[S]trict serializability

[W]rite transactions that conflict

[O]ne response per read

[N]on-blocking operations

\{ Highest Power \}

\{ Lowest Latency \}
[S]trict Serializability

• Strongest model: real-time + total order
[S]trict Serializability

• Strongest model: real-time + total order

\[ C_R \quad S_{ACL} \quad S_{Photo} \quad C_W \]

\{ ACL := Private
\{ Upload Photo B

\{ W starts
\{ W finishes
[S]trict Serializability

• Strongest model: real-time + total order

ACL := Private

Upload Photo B

R starts

R finishes

W starts

W finishes
**[S]trict Serializability**

- Strongest model: real-time + total order

---

```
“Photo B is private!”

R starts

R finishes

Photo B

ACL := Private

Upload Photo B

W starts

W finishes
```
[S]trict Serializability

- Strongest model: real-time + total order

Diagram:
- CR
- S_{ACL}
- S_{Photo}
- CW

Events:
- R starts
- R finishes
- W starts
- W finishes

ACL := Private
Upload Photo B
[S]trict Serializability

• Strongest model: real-time + total order

```
CR  S_{ACL}  S_{Photo}  CW
```

R starts

```
“Public + Photo A”
“Photo B is private!”
```

R finishes

```
“Public + Photo B”
“Photo A is private!”
```

W starts

```
ACL := Private
Upload Photo B
```

W finishes
Non-blocking Operations

• Do not wait on external events
  – Locks, timeouts, messages, etc.

• Lower latency
  – Save the time spent blocking
One Response

- One round-trip
  - No message redirection
    - Centralized components: coordinator, etc.
  - No retries
  - Save the time for extra round-trips

- One value per response
  - Less time for transmitting, marshaling, etc.
[W]rite Transactions That Conflict

- Compatible with write transactions
  - Richer system model
  - Easier to program
The SNOW Theorem:

Impossible for read-only transaction algorithms to have all SNOW properties
Why SNOW Is Impossible

C_R  S_A  S_B  C_W
Why SNOW Is Impossible

Assume SNOW

\[ R \]

\[ R_A = \text{new} \]
\[ R_B = \text{old} \]

\[ W \text{ starts} \]
\[ \{ A := \text{new} \]
\[ B := \text{new} \]

\[ W \text{ invisible} \]

\[ W \text{ visible} \]

\[ W \text{ finishes} \]
Why SNOW Is Impossible

Assume SNOW

Violates property S

\( R_A = \text{new} \)
\( R_B = \text{old} \)

\[ W \text{ starts} \]
\[ \{ A := \text{new} \]
\[ B := \text{new} \]

\[ W \text{ invisible} \]
\[ W \text{ visible} \]

W finishes
A Deeper Look at SNOW

• Complete proof in the paper

• SNOW is tight
  – Any combination of 3 properties is possible

• Optimality
  – SNOW-optimal: have any 3 properties
  – Latency-optimal: have property N and O

• Spectrums of property S and O
  – Show what is possible to achieve
# Study Existing Systems with SNOW

SNOW-optimal and latency-optimal

<table>
<thead>
<tr>
<th>System</th>
<th>S</th>
<th>N</th>
<th>O</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanner-Snap [OSDI ’12]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Yesquel [SOSP ’15]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MySQL Cluster</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Study Existing Systems with SNOW
SNOW-optimal

<table>
<thead>
<tr>
<th>System</th>
<th>S</th>
<th>N</th>
<th>O</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiger [NSDI '13]</td>
<td>✔</td>
<td>✔</td>
<td>≤ 3</td>
<td>✔</td>
</tr>
<tr>
<td>DrTM [SOSP '15]</td>
<td>✔</td>
<td>✔</td>
<td>≥ 1</td>
<td>✔</td>
</tr>
<tr>
<td>RIFL [SOSP '15]</td>
<td>✔</td>
<td>✔</td>
<td>≥ 2</td>
<td>✔</td>
</tr>
<tr>
<td>Sinfonia [SOSP '07]</td>
<td>✔</td>
<td>✔</td>
<td>≥ 2</td>
<td>✔</td>
</tr>
<tr>
<td>Spanner-RO [OSDI '12]</td>
<td>✔</td>
<td>×</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Study Existing Systems with SNOW
Candidates for Improvement

<table>
<thead>
<tr>
<th>System</th>
<th>S</th>
<th>N</th>
<th>O</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPS</td>
<td>✗</td>
<td>✓</td>
<td>≤ 2</td>
<td>✗</td>
</tr>
<tr>
<td>Rococo</td>
<td>✓</td>
<td>✗</td>
<td>&gt; 1</td>
<td>✓</td>
</tr>
</tbody>
</table>

Many more
Improve Existing Systems with the SNOW Theorem

- **COPS [SOSP ’11]**
  - Geo-replicated
  - Causally consistent
  - Read-only txn: ∙ N ⊗ W

- **Rococo [OSDI ’14]**
  - Supports general transactions
  - Strictly serializable
  - Read-only txn: S ∙ ⊗ W
New Algorithm Designs

• **COPS-SNOW**
  – Latency-optimal (N + O)

• **Rococo-SNOW**
  – SNOW-optimal (S + O + W)

Design insight for optimizing reads: shift the overhead to writes
Rococo’s Read-Only Txn (S + W)
Rococo’s Read-Only Txn (S + W)

A = “old”

B = “new”

W starts

A := “new”
B := “new”

Gather conflict info

W commits

W finishes

R: 1\textsuperscript{st} round

R: 2\textsuperscript{nd} round

R: N\textsuperscript{th} round
Rococo-SNOW (S+O+W)

W starts
\[ \begin{align*}
A &:= \text{"new"} \\
B &:= \text{"new"}
\end{align*} \]

W commits

W finishes

W commits

W finishes
Rococo-SNOW (S+O+W)

C_R

S_A

S_B

C_W

A="old"

W starts
A := "new"
B := "new"

W commits

W finishes
Rococo-SNOW (S+O+W)

W starts
\{ A := “new” \\
B := “new” \}

W commits

W finishes

A = “old”
Rococo-SNOW (S+O+W)

W starts
A := “new”
B := “new”

W commits

W finishes

A = “old”
Rococo-SNOW (S+O+W)

\[ C_R \quad S_A \quad S_B \quad C_W \]

R

A="old"

W starts
\{ A := "new" \\
B := "new" \}

Forward TS

W commits

W commits

W finishes

W finishes
Rococo-SNOW (S+O+W)

W starts
\{ A := "new" \\
B := "new" \}

Forward TS

W commits

W finishes

A = "old"
Rococo-SNOW (S+O+W)

\[ \text{W starts} \]
\[ \{ \text{A := "new"} \]
\[ \{ \text{B := "new"} \]
\[ \text{Forward TS} \]
\[ \text{W commits} \]

\[ \text{W finishes} \]
Rococo-SNOW (S+O+W)

- **CR**: R
- **SA**: A = “old”
- **SB**: B = “old”
- **CW**: W starts
  - \( A := \text{“new”} \)
  - \( B := \text{“new”} \)
- **TS**: Forward TS
- **Blocks**: W commits
- **W**: W finishes

**Strictly Serializable**

- A = old
- B = old
Evaluation of Rococo-SNOW

• To understand
  – Latency of read-only transactions
  – Throughput of other types of transactions

• Experiment configuration
  – Identical to Rococo’s
  – TPC-C workloads

• https://github.com/USC-NSL/Rococo-SNOW
Significantly Lower Latency for Read-Only Txn
Significantly Lower Latency
for Read-Only Txn

Latency (ms)

Concurrent requests/server

OCC
Rococo
2PL
Rococo
-SNOW

Retries
Lock Wait
Always 1 round
Higher Throughput under High Contention

Concurrent requests/server

Throughput (new-order/s)

Rococo
-SNOW

Rococo

2PL

OCC
Higher Throughput under High Contention

-14% throughput (Low Contention)
2X throughput (High Contention)
Conclusion

• The SNOW Theorem for read-only txns
  – Impossible to have all of the SNOW properties

• SNOW helps understand existing systems
  – Many are not yet optimal

• Rococo-SNOW
  – SNOW Theorem guided SNOW-optimal design
  – Significantly higher throughput and lower latency under high contention