Automating the Choice of Consistency Levels in Replicated Systems

Cheng Li†, João Leitão§, Allen Clement†
Nuno Preguiça§, Rodrigo Rodrigues§, Viktor Vafeiadis†

Max Planck Institute for Software Systems (MPI-SWS)†,
NOVA-LINCS / Universidade Nova de Lisboa§
RedBlue Consistency [OSDI’12]

Builds replicated systems that are fast
RedBlue Consistency [OSDI’12]

Builds replicated systems that are **fast**

**Blue ops:** local, fast, weakly consistent
RedBlue Consistency [OSDI’12]

Builds replicated systems that are **fast** and **correct**

**Blue ops**: local, fast, weakly consistent

State convergence
Invariant preservation
RedBlue Consistency [OSDI’12]

Builds replicated systems that are **fast** and **correct**

**Blue ops**: local, fast, weakly consistent

**Red ops**: global, slow, strongly consistent
Choosing between Blue or Red

Ensuring state convergence

operation \( u \)

commutative?

No

Red
Choosing between **Blue** or **Red**

 Ensuring state convergence

 operation $u$

 commutative?

 Yes

 breaks invariants?

 Yes

 Ensuring invariant preservation

 No

 Red

 Yes

 No

 Blue
Choosing between Blue or Red

- Good performance obtained if blue ops dominate op space

```
Ensuring state convergence

operation \( u \)

commutative?

Yes

Ensuring invariant preservation

No

Ensuring state convergence

Yes

Red

No

Blue

breaks invariants?

Yes

No
```
RedBlue Consistent Bank

Replica 1

10

Deposit:
\[ b = b + 10 \]

20

Replica 2

10

AccrueInterest
\[ b = b \times 1.5 \]

15
RedBlue Consistent Bank

Replica 1

10

20

30

AccrueInterest
\[ b = b \times 1.5 \]

Replica 2

10

15

25

Deposit:
\[ b = b + 10 \]

\[ \neq \]
Replica 1
10
20
30

Replica 2
15

Accrue Interest
\[ b = b \times 1.5 \]

Deposit:
\[ b = b + 10 \]

Operations are not commuting!
RedBlue Consistent Bank

Replica 1

Replica 2

Operations are not commuting!

Deposit: $b = b + 10$

AccrueInterest

$b = b \times 1.5$

Replica 1

Replica 2

Deposit: $b = b + 10$

AccrueInterest

$b = b \times 1.5$

AccrueInterest

$b = b \times 1.5$

6/20/2014

Cheng Li @ USENIX ATC’14

MPI-SWS/NOVALINCS
RedBlue Consistent Bank

Replica 1

Replica 2

Operations are not commuting!

AccrueInterest
\[ b = b \times 1.5 \]

Deposit:
\[ b = b + 10 \]

\[ \neq \]

\[ 30 \]

\[ 15 \]

\[ 25 \]

Replica 1

Replica 2

Deposit:
\[ b = b + 10 \]

AccrueInterest
\[ b = b \times 1.5 \]

\[ 20 \]

\[ 15 \]

\[ +10 \]

\[ +5 \]

6/20/2014

Cheng Li @ USENIX ATC’14

MPI-SWS/NOVALINCS
RedBlue Consistent Bank

Replica 1

Replica 2

AccrueInterest
\[ b = b \times 1.5 \]

Deposit:
\[ b = b + 10 \]

Operations are not commuting!

Replica 1

Replica 2

Deposit:
\[ b = b + 10 \]

AccrueInterest
\[ b = b \times 1.5 \]
RedBlue Consistent Bank

Maximize the blue op space by encoding side effects into commutative shadow operations

Operations are not commuting!

Replica 1

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
</table>

AccrueInterest
\[ b = b \times 1.5 \]

Replica 2

| 15 |

Deposit:
\[ b = b + 10 \]

Replica 1

| 10 |

Deposit:
\[ b = b + 10 \]

Replica 2

| 15 |

AccrueInterest
\[ b = b \times 1.5 \]
Operation stream

Transforming

Commutative shadow operations

Classifying

Fast, Weakly consistent

Slow, Strongly consistent
SIEVE

Operation stream

Transforming

Commutative shadow operations
Transforming Operation stream

Commutative shadow operations

Challenges:
- Making arbitrary side effects commute
- Minimizing human intervention
Two-tier Application Model

- Observation: Side effects are encapsulated into a sequence of DB statements

Cheng Li @ USENIX ATC’14

6/20/2014
Two-tier Application Model

- Observation: Side effects are encapsulated into a sequence of DB statements
- Insight: We can model the database using commutative replicated data types (CRDTs)
Leveraging CRDTs

- Transform each DB statement into one or more CRDT operations
- Programmers only annotate schema with CRDTs

DB Table \rightarrow Set \rightarrow DB Field \rightarrow Counter/Rewritable value
Leveraging CRDTs

- Transform each DB statement into one or more CRDT operations

Shadow operation: [CRDT_OP1; CRDT_OP2; CRDT_OP3;...]
Leveraging CRDTs

- Transform each DB statement into one or more CRDT operations

Shadow operation:  [CRDT_OP1; CRDT_OP2; CRDT_OP3;...]

- Programmers **only** annotate schema with CRDTs:

  @[CRDTName][TableName | DataFieldNome]
CREATE TABLE BankAccount(
    id INT(11) NOT NULL,
    balance INT(11) default 0,
    name char(60) default NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB
CRDT Annotation Example

@AUSET CREATE TABLE BankAccount(
    id INT(11) NOT NULL,
@NUMDELTAINBalance INT(11) default 0,
@LWW name char(60) default NULL,
    PRIMARY KEY (id)
) ENGINE=InnoDB

For details, refer to our paper.
SIEVE

Operation stream

Transforming

Commutative shadow operations

Classifying

Fast, Weakly consistent

Slow, Strongly consistent
SIEVE

Classifying

Commutative shadow operations

Fast, Weakly consistent

Slow, Strongly consistent
Challenge:

- How to classify accurately and efficiently?
Straw man Solution

Op
Straw man Solution

\[
\text{Op} + \begin{array}{l}
\text{parameters} \\
\text{initial\_state}
\end{array}
\]
Straw man Solution

\[ \text{Op} + \begin{array}{c} \text{parameters} \\ \text{initial\_state} \end{array} \quad \text{Generates} \quad \text{Shadow op} \]
Straw man Solution

Can we pre-determine a combination that produces a Red or Blue shadow op?

\[ \text{Op} + \begin{array}{c} \text{parameters} \\ \text{initial\_state} \end{array} \]
Straw man Solution

- Statically define, for each original operation, a weakest precondition (WP) for corresponding shadow op to be invariant preserving
- At runtime, we classify shadow operations by evaluating the corresponding WP
WP Computation

- Problem: WP computation is infeasible (inverting hash function)

Invariant: x >= 0
void foo(string s) {
    if (SHA-1(s)==SOME_CONSTANT) {
        if (x>=10)
            x -= 10;
    } else
        x +=10;
}
WP Computation

• Problem: WP computation is infeasible (inverting hash function)
  – Classification will be conservative.

```
Invariant: x >= 0
void foo(string s) {
  if (SHA-1(s) == SOME_CONSTANT) {
    if (x >= 10)
      x -= 10;
  } else
    x += 10;
}
```

Weakest precondition must be FALSE!
WP Computation

• Problem: WP computation is infeasible (inverting hash function)
  – Classification will be conservative.

• Observation: side effects are simple.

```c
void foo(string s) {
    if (SHA-1(s) == SOME_CONSTANT) {
        if (x >= 10) x -= 10;
    } else {
        x += 10;
    }
}
```

Invariant: x >= 0

Weakest precondition must be FALSE!
WP Computation

• Problem: WP computation is infeasible (inverting hash function)
  – Classification will be conservative.

• Observation: side effects are simple.

Invariant: x >= 0
void foo(string s) {
  if (SHA-1(s)==SOME_CONSTANT) {
    if (x>=10)
      x -= 10;
  } else
    x += 10;
}

Apply -10 to x
Do nothing
Apply 10 to x
Path-basis Analysis

• Creates a **template** per control flow path to capture all possible shadow operations following that path

```
Invariant: x >= 0
void foo(string s) {
    if (SHA-1(s)==SOME_CONSTANT) {
        if (x>=10)
            x -= 10;
    } else
        x +=10;
}
```

[-10] [-10]
[+10] [+10]
Path-basis Analysis

- Creates a **template** per control flow path to capture all possible shadow operations following that path
- WP computation is done over parameters of CRDT invocations in templates

**Invariant:** \( x \geq 0 \)

```c
void foo(string s) {
    if (SHA-1(s)==SOME_CONSTANT) {
        if (x>=10) x -= 10;
    } else
        x += 10;
}
```

WP: FALSE

WP: TRUE

WP: TRUE
Do most (not all) work offline!

Static analyzer

Runtime Generator/checker

SIEVE
Do most (not all) work offline!

App Code

Programmer inputs:
Annotations
Invariants

Static analyzer

Runtime Generator/checker

SIEVE
Do most (not all) work offline!

Programmer inputs:
Annotations
Invariants

App Code

Static analyzer

Produces

<table>
<thead>
<tr>
<th>Shadow operation template identifier</th>
<th>Weakest precondition (WP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>$a &gt; 100$</td>
</tr>
<tr>
<td>id2</td>
<td>$b &lt;&gt; 0$</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

Runtime Generator/checker

 MPI-SWS/NOVALINCS

6/20/2014
Cheng Li @ USENIX ATC’14

Programmer inputs:
Annotations
Invariants

App Code

Static analyzer

Produces

<table>
<thead>
<tr>
<th>Shadow operation template identifier</th>
<th>Weakest precondition (WP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>$a &gt; 100$</td>
</tr>
<tr>
<td>id2</td>
<td>$b &lt;&gt; 0$</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

Runtime Generator/checker

 MPI-SWS/NOVALINCS

6/20/2014
Cheng Li @ USENIX ATC’14
Do most (not all) work offline!

App Code

Programmer inputs:
Annotations
Invariants

Static analyzer

App runtime

Produces

<table>
<thead>
<tr>
<th>Shadow operation template identifier</th>
<th>Weakest precondition (WP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>a &gt; 100</td>
</tr>
<tr>
<td>id2</td>
<td>b &lt;&gt; 0</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
Do most (not all) work offline!

App Code

Programmer inputs: Annotations Invariants

Static analyzer

Produces

Runtime Generator/checker

Lookups

<table>
<thead>
<tr>
<th>Shadow operation template identifier</th>
<th>Weakest precondition (WP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$id1$</td>
<td>$a &gt; 100$</td>
</tr>
<tr>
<td>$id2$</td>
<td>$b &lt;&gt; 0$</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>
Do most (not all) work offline!

Programmer inputs:
Annotations
Invariants

App Code
Static analyzer

Produces

Shadow operation template identifier
id1
id2

Weakest precondition (WP)
a > 100
b <> 0

......

Runtime Generator/checker

Lookups

True -> Blue
False -> Red

SIEVE
Evaluation
Questions

• Application adaptation
  – How easy is it to adapt apps to SIEVE?

• Static analysis
  – How long does the static analysis process take?
  – How well does the static analysis scale?

• Runtime part
  – Is the runtime classification accurate?
  – What is the overhead?
  – How does the replicated application perform?
Questions

• Application adaptation
  – How easy is it to adapt apps to SIEVE?

• Static analysis
  – How long does the static analysis process take?
  – How well does the static analysis scale?

• Runtime part
  – Is the runtime classification accurate?
  – What is the overhead?
  – How does the replicated application perform?
Experimental Setup

Original

TPCW/RUBiS

MySQL

Manual

TPCW/RUBiS + Manual adaptation

RedBlue framework

MySQL

SIEVE

TPCW/RUBiS + SIEVE

RedBlue framework

MySQL

Baselines
- SIEVE performs almost as well as manual adaptation
- Runtime labeling takes negligible time
  - TPCW : $0.064 \pm 0.002$ ms
  - RUBiS : $0.072 \pm 0.001$ ms
Conclusion

SIEVE *automatically* and *efficiently* chooses weak consistency (blue) *whenever possible*, and strong consistency (red) *when needed*, only requiring little programmer input.