SYMBOLIC ANALYSIS OF WEAK CONCURRENCY SEMANTICS IN MODERN DATABASE PROGRAMS

Kia Rahmani
Aug 2, 2022
University of Texas at Austin
Austin, TX
INTRODUCTION

(MAIN SUBJECT: DATABASE SYSTEMS)
- Data is ubiquitously managed by databases
- Data is ubiquitously managed by databases
- Well-defined abstractions
  ‣ data definition language (DDL)
  ‣ data manipulation language (DML)
- Data is ubiquitously managed by databases
- Well-defined abstractions
  - data definition language (DDL)
  - data manipulation language (DML)
- Single machine
- Data is ubiquitously managed by databases
- Well-defined abstractions
  - data definition language (DDL)
  - data manipulation language (DML)
- Single machine
- Strong concurrency guarantees
- Single machine = poor performance
- Single machine = poor performance
- Amazon DynamoDB
  - Cluster of nodes on the cloud
- Single machine = poor performance
- Amazon DynamoDB
  - Cluster of nodes on the cloud
- Designed to scale
- Single machine = poor performance
- Amazon DynamoDB
  - Cluster of nodes on the cloud
- Designed to scale
- Clustering architectures
  ▪ Partitioning
  ▪ Replicating
SCALING DATABASE CLUSTERS

- Clustering architectures
  ‣ Partitioning
  ‣ Replicating
SCALING DATABASE CLUSTERS

- Clustering architectures
  - Partitioning
  - Replicating
SCALING DATABASE CLUSTERS

- Clustering architectures
  - Partitioning
  - Replicating
- Clustering architectures
  - Partitioning
  - Replicating

RTT = 600ms
WEAK CONCURRENCY SEMANTICS

- Clustering architectures
  - Partitioning
  - Replicating
- Weak concurrency guarantees
WEAK CONCURRENCY SEMANTICS

- Clustering architectures
  - Partitioning
  - Replicating
- Weak concurrency guarantees
  - Programmer is exposed to concurrency
WEAK CONCURRENCY SEMANTICS

- Clustering architectures
  - Partitioning
  - Replicating
- Weak concurrency guarantees
- Programmer is exposed to concurrency
- Concurrency bugs
  - prevalent
  - dangerous
  - alerted database community
MY THESIS: THE OVERARCHING GOAL

- Ease of strong semantics
- Performance of cloud-native
MY THESIS: THE OVERARCHING GOAL

- Ease of strong semantics
- Performance of cloud-native
- Language-oriented solutions
  - SMT Solvers
  - Model Checking
- Ease of strong semantics
- Performance of cloud-native
- Language-oriented solutions
  - SMT Solvers
  - Model Checking

\[ \varphi \]
MY THESIS: THE OVERARCHING GOAL

- Ease of strong semantics
- Performance of cloud-native
- Language-oriented solutions
  - SMT Solvers
  - Model Checking

<table>
<thead>
<tr>
<th></th>
<th>Guided Testing Framework</th>
<th>OOPSLA 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOTHO</td>
<td>Automated repair of replication anomalies</td>
<td>PLDI 2021</td>
</tr>
<tr>
<td>ATROPOS</td>
<td>Automated repair of partitioning anomalies</td>
<td>under submission</td>
</tr>
<tr>
<td>LACHESIS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MY THESIS: THE OVERARCHING GOAL

- Ease of strong semantics
- Performance of cloud-native
- Language-oriented solutions
  - SMT Solvers
  - Model Checking

<table>
<thead>
<tr>
<th>CLOTHO</th>
<th>Guided Testing Framework</th>
<th>OOPSLA 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATROPOS</td>
<td>Automated repair of replication anomalies</td>
<td>PLDI 2021</td>
</tr>
<tr>
<td>LACHESIS</td>
<td>Automated repair of partitioning anomalies</td>
<td>under submission</td>
</tr>
</tbody>
</table>
MY THESIS: THE OVERARCHING GOAL

- Ease of strong semantics
- Performance of cloud-native
- Language-oriented solutions
  - SMT Solvers
  - Model Checking

<table>
<thead>
<tr>
<th>CLOTHO</th>
<th>Guided Testing Framework</th>
<th>OOPSLA 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATROPOS</td>
<td>Automated repair of replication anomalies</td>
<td>PLDI 2021</td>
</tr>
<tr>
<td>LACHESIS</td>
<td>Automated repair of partitioning anomalies</td>
<td>under submission</td>
</tr>
</tbody>
</table>
CLOTHO

(SYMBOLIC PROGRAM ANALYSIS & GUIDED TESTING)
- ACID guarantees
- **ACID** guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- **ACID** guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- **ACID** guarantees
  
  ▶ Atomicity
  ▶ Consistency
  ▶ Isolation
  ▶ Durability

TRANSACTIONAL GUARANTEES

**All or None**

**Single Copy of Data**

**Txn**

- READ
- READ
- WRITE
- **ACID** guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Transactional Guarantees**: All or None, Single Copy of Data, No Interference from the Environment.
- **ACID** guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- **ACID** guarantees
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Serializability
- **ACID guarantees**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Serializability

- Facilitates program design and reasoning

**Diagram:**
- **All or None**
- **Single Copy of Data**
- **No Interference from the Environment**
- **Persistent Commits**

**Transaction (Txn):**
- READ
- READ
- WRITE
- Serializability is costly
- Serializability is costly
- Developers forced to use weaker semantics
  - Transactions are interleaved
  - Asynchronous replication of data
- Serializability is costly
- Developers forced to use weaker semantics
  - Transactions are interleaved
  - Asynchronous replication of data
- Less intuitive behaviors
- Serializability is costly
- Developers forced to use weaker semantics
  - Transactions are interleaved
  - Asynchronous replication of data
- Less intuitive behaviors
- Online gaming platform
- Online gaming platform
  - 2 tables

<table>
<thead>
<tr>
<th>PLAYER</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>p_t_id</td>
<td>p_role</td>
<td>p_stat</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEAM</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_id</td>
<td>t_name</td>
<td>p_cnt</td>
<td></td>
</tr>
</tbody>
</table>
- Online gaming platform
  » 2 tables

<table>
<thead>
<tr>
<th>PLAYER</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>t_id</td>
</tr>
<tr>
<td>p_t_id</td>
<td>t_name</td>
</tr>
<tr>
<td>p_role</td>
<td>p_cnt</td>
</tr>
<tr>
<td>p_stat</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE (TRANSACTIONS)

- Online gaming platform
  - 2 tables
  - 3 transactions

<table>
<thead>
<tr>
<th>PLAYER</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>p_t_id</td>
<td>p_role</td>
<td>p_stat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEAM</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_id</td>
<td>t_name</td>
<td>p_cnt</td>
</tr>
</tbody>
</table>
- Online gaming platform
  ▪ 2 tables
  ▪ 3 transactions

```
addPlayer(p, t)

old_cnt := select p_cnt from TEAM where t_id=t
Insert (p, t, ∅, ∅) into PLAYER
update TEAM set p_cnt=old_cnt+1 where t_id=t
```
EXAMPLE (TRANSACTIONS)

- Online gaming platform
  ▪ 2 tables
  ▪ 3 transactions

addPlayer\((p, t)\)

\[
\text{old}_\text{cnt} := \text{select } p_{\_\text{cnt}} \text{ from TEAM where } t_{\_\text{id}}=t
\]

\[
\text{Insert } (p, t, \emptyset, \emptyset) \text{ into PLAYER}
\]

\[
\text{update TEAM set } p_{\_\text{cnt}}=\text{old}_\text{cnt}+1 \text{ where } t_{\_\text{id}}=t
\]

PlayerByTeam \((t)\)

\[
ps := \text{select } * \text{ from PLAYER where } p_{\_t\_id}=t
\]

\[
\text{foreach } p \text{ in } ps \text{ do}
\]

\[
\text{print}(p.\_\*)
\]
- Online gaming platform
  - 2 tables
  - 3 transactions

addPlayer(p, t)

old_cnt := select p_cnt from TEAM where t_id=t
Insert (p,t,∅,∅) into PLAYER
update TEAM set p_cnt=old_cnt+1 where t_id=t

PlayerByTeam (t)

ps := select * from PLAYER where p_t_id=t
foreach p in ps do
  print(p.*)

setPlayer (p, r, s)

update PLAYER set p_role=r and p_stat=p where p_id=p
EXAMPLE (ANOMALY)

TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

addPlayer(p=P0, t=T1)

Select p_cnt // 2
...
Update p_cnt = 3

addPlayer(p=P0, t=T1)

Select p_cnt // 2
...
Update p_cnt = 3
EXAMPLE (ANOMALY)

TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>&quot;A&quot;</td>
<td>2</td>
</tr>
</tbody>
</table>

addPlayer(p=P0, t=T1)

Select p_cnt // 2

... Update p_cnt = 3

addPlayer(p=P0, t=T1)

Select p_cnt // 2

... Update p_cnt = 3
EXAMPLE (ANOMALY)

TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

addPlayer(p=P0, t=T1)

Select p_cnt // 2

Update p_cnt = 3

addPlayer(p=P0, t=T1)

Select p_cnt // 2

Update p_cnt = 3

TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>3</td>
</tr>
</tbody>
</table>
### TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

**addPlayer(p=P0, t=T1)**

1. Select `p_cnt // 2`
2. Select `p_cnt // 2`
3. Update `p_cnt = 3`
4. Update `p_cnt = 3`

```
addPlayer(p=P0, t=T1)
Select p_cnt // 2
... Update p_cnt = 3
```

**Lost Update**

```
addPlayer(p=P0, t=T1)
Select p_cnt // 2
... Update p_cnt = 3
```

---

**Lost Update**
### TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

```
addPlayer(p=P0, t=T1)  
Select p_cnt // 2  
...  
Update p_cnt = 3  
```

```
addPlayer(p=P0, t=T1)  
Select p_cnt // 2  
...  
Update p_cnt = 3  
```

“The column ‘p_cnt’ reflects the number of players in each team”

---

Lost Update

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>3</td>
</tr>
</tbody>
</table>
Example (Anomaly)

```
<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>&quot;A&quot;</td>
<td>2</td>
</tr>
</tbody>
</table>
```

```
addPlayer(p=P0, t=T1)
Select p_cnt // 2
... 
Update p_cnt = 3
```

```
addPlayer(p=P0, t=T1)
Select p_cnt // 2
... 
Update p_cnt = 3
```

```
<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>&quot;A&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>
```

"The column ‘p_cnt’ reflects the number of players in each team”

Invariant Violated!

Lost Update
- Serializability anomalies are subtle
- Serializability anomalies are subtle
  ▪ initial state

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

TEAM

addPlayer(p=P0, t=T1)

Select p_cnt // 2

... Update p_cnt = 3

addPlayer(p=P0, t=T1)

Select p_cnt // 2

... Update p_cnt = 3
- Serializability anomalies are subtle
  • initial state
  • transaction arguments

```
TEAM

<table>
<thead>
<tr>
<th>id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

addPlayer(p=P0, t=T1)
Select p_cnt // 2
...
Update p_cnt = 3
```

```
addPlayer(p=P0, t=T1)
Select p_cnt // 2
...
Update p_cnt = 3
```
- Serializability anomalies are subtle
  ▪ initial state
  ▪ transaction arguments
  ▪ interleaved order

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

addPlayer(p=P0, t=T1)
Select p_cnt // 2
Update p_cnt = 3
addPlayer(p=P0, t=T1)
Select p_cnt // 2
Update p_cnt = 3
CHALLENGES OF TESTING

- Serializability anomalies are subtle
  - initial state
  - transaction arguments
  - interleaved order

Exponential state space!
BLACKBOX TESTING

App Runtime

Bugs
- Run and monitor apps in (semi-) production environment
BLACKBOX TESTING

- Run and monitor apps in (semi-) production environment
  + real bugs (no false positive)
  - costly
  - too specific
  - no guarantee of coverage
  - manual effort
- Cloud-based testing framework
  - OLTPBench
  - Jepsen
- Cloud-based testing framework
  - OLTPBench
  - Jepsen
- TPC-C Benchmark
  - 5txn and 9 tables
- Cloud-based testing framework
  - OLTPBench
  - Jepsen
- TPC-C Benchmark
  - 5 txn and 9 tables
  - 21 application-level invariants
- Cloud-based testing framework
  ▪ OLTPBench
  ▪ Jepsen
- TPC-C Benchmark
  ▪ 5 txn and 9 tables
  ▪ 21 application-level invariants
  ▪ only 14 invariants broken

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Broken?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
</tr>
</tbody>
</table>
ANOTHER SOLUTION?

- Cloud-based testing framework
  - OLTPBench
  - Jepsen
- TPC-C Benchmark
  - 5 txn
  - 9 tables
  - only 14 invariants broken

---

- **Invariant**
  - CR1
  - CR2
  - CR3
  - CR4
  - CR5A
  - CR5B

---

- **Invariant**
  - CR12
  - NCR1
  - NCR2
  - NCR3
  - NCR4
  - NCR5
  - NCR6
  - NCR7

---

**— KEY IDEA —**

1. Statically analyze programs and find abstract anomalies (white-box testing)
2. Construct and run anomalous executions to the devs
- Non-serializable executions in a program, iff
  • Cyclic dependencies between txns
- Non-serializable executions in a program, iff
  ◦ Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Non-serializable executions in a program, iff
  ▶ Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Three types of dependency edges
- Non-serializable executions in a program, iff
  ▶ Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Three types of dependency edges
  ▶ write dependency (WW)
- Non-serializable executions in a program, iff
  ▶ Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Three types of dependency edges
  ▶ write dependency (WW)
  ▶ read dependency (WR)
- Non-serializable executions in a program, iff
  ▶ Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Three types of dependency edges
  ▶ write dependency (WW)
  ▶ read dependency (WR)
  ▶ read anti-dependency (RW)
- Non-serializable executions in a program, iff
  - Cyclic dependencies between txns
- Executions abstracted by directed dependency graphs
- Three types of dependency edges
  - write dependency ($WW$)
  - read dependency ($WR$)
  - read anti-dependency ($RW$)
- Example: lost update

```
Select p_cnt // 2
... addPlayer(p=P0, t=T1)
Update p_cnt = 3
... addPlayer(p=P0, t=T1)
Select p_cnt // 2
... Update p_cnt = 3
```
- Key idea: reduction to SMT
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula $\varphi$
FINDING CYCLES STATICALLY

- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula $\varphi$
- Components of the encoding

$$\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP} \rightarrow} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}}$$
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula $\varphi$
- Components of the encoding

$$\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP} \rightarrow} \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}}$$

conditions satisfied by any execution of any program
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula \( \varphi \)
- Components of the encoding

\[
\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi_{\rightarrow \text{DEP}} \land \varphi_{\text{ANOMALY}}
\]

- Conditions satisfied by any execution of any program
- Database-specific consistency constraints
FINDING CYCLES STATICALLY

- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula $\varphi$
- Components of the encoding

$$\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi \rightarrow \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}}$$

- Database-specific consistency constraints
- Sufficient conditions to establish a dependency relation
- Necessary conditions to establish a dependency relation
- Conditions satisfied by any execution of any program
- Key idea: reduction to SMT
- Use efficient SMT-solvers, e.g. Z3
- Axiomatic relations encoded within a decidable fragment of FOL
- Finding bounded anomalies against a database abstraction reduced to finding satisfying assignments to a formula \( \varphi \)
- Components of the encoding

\[
\varphi \equiv \varphi_{\text{CONTEXT}} \land \varphi_{\text{DB}} \land \varphi_{\text{DEP}} \rightarrow \land \varphi \rightarrow \varphi_{\text{DEP}} \land \varphi_{\text{ANOMALY}}
\]

- Conditions satisfied by any execution of any program
- **Necessary** conditions to establish a dependency relation
- **Sufficient** conditions to establish a dependency relation
- Enforces the existence of a dependency cycle
- Static analysis engine for java programs
- Automatic replay of anomalous executions
- 7 benchmarks of various complexity analyzed
- Serializability anomalies found and replayed in 5 benchmark
EXPERIMENTAL RESULTS

- **7 benchmarks** of various complexity analyzed
- Serializability anomalies found and replayed in 5 benchmarks

![Bar chart showing number of anomalies per application.]

- ~25m per application (avg)
- 17 anomalies per application (avg)
- 22 anomalies mapped to invariant violations
  - All invariants were broken
  - Only 3 anomalies did NOT violate any invariant

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Blackbox</th>
<th>CLOTHO</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR3</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR4</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR5A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR5B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR7A</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR7B</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>CR8</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR9</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR10</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR11</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>CR12</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR2</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR3</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR4</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NCR5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR6</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NCR7</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
ATROPOS

(REPAIRING REPLICATION ANOMALIES)
- Serializability anomaly

\[
\text{addPlayer}(p, t) \\
\text{old}_\text{cnt} := \text{select } p\_\text{cnt} \text{ from TEAM where } t\_\text{id}=t \\
\text{Insert } (p, t, \emptyset, \emptyset) \text{ into PLAYER} \\
\text{update TEAM set } p\_\text{cnt}=\text{old}_\text{cnt}\!+\!1 \text{ where } t\_\text{id}=t
\]
REPLICATION ANOMALIES

- Serializability anomaly

\[ old\_cnt := \text{select } p\_cnt \text{ from TEAM where } t\_id=t \]

\textbf{addPlayer}(p, t)

\textbf{Insert} \((p, t, \emptyset, \emptyset)\) \textbf{into} PLAYER

\textbf{update} TEAM \textbf{set} \(p\_cnt=old\_cnt+1\) \textbf{where} \(t\_id=t\)
- Serializability anomaly
- Replication anomalies can be eliminated using consistency annotations

```
addPlayer(p, t)

Consistent {
  old_cnt := select * from TEAM where t_id=t
  Insert (p, t, ∅, ∅) into PLAYER
  update TEAM set p_cnt=old_cnt+1 where t_id=t
}
```
- Serializability anomaly
- Replication anomalies can be eliminated using consistency annotations
- **Equivalent program** without dependency cycles

```sql
addPlayer(p, t)

Consistent {
    old_cnt := select * from TEAM where t_id=t
    Insert (p,t,∅,∅) into PLAYER
    update TEAM set p_cnt=old_cnt+1 where t_id=t
}
```

<table>
<thead>
<tr>
<th>PLAYER</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>t_id</td>
</tr>
<tr>
<td>p_t_id</td>
<td>t_name</td>
</tr>
<tr>
<td>p_role</td>
<td>p_stat</td>
</tr>
<tr>
<td>p_cnt</td>
<td></td>
</tr>
</tbody>
</table>
- Equivalent program without dependency cycles
  ‣ Schema Refactoring
- Equivalent program without dependency cycles
  - Schema Refactoring
  - Program Refactoring
- Equivalent program without dependency cycles
  - Schema Refactoring
  - Program Refactoring
- Only keep track of changes to the balance
- Equivalent program without dependency cycles
  - Schema Refactoring
  - Program Refactoring
- Only keep track of changes to the balance
- No shared item $\implies$ No dependency

```sql
addPlayer(p, t)

old_cnt := select p_cnt from TEAM where t_id=t
Insert (p, t, Ø, Ø) into PLAYER
Insert (t, Ø, uuid(), +1) into TEAM_P_CNT_LOG
```

New record is inserted for each new player

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Equivalent program without dependency cycles
  - Schema Refactoring
  - Program Refactoring
- Only keep track of changes to the balance
- No shared item $\implies$ No dependency

```
TEAM
<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
</table>

cnt := select p_cnt
from TEAM where t_id=t
```

```
TEAM_P_CNT_LOG
<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
</table>

cnt := select SUM(p_cnt_change)
from TEAM_P_CNT_LOG where t_id=t
```

Refactor

Updated expressions to return equivalent logical values as the original program
REPAIRED PROGRAM’S EXECUTION

TEAM_P_CNT_LOG

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“A”</td>
<td>U3</td>
<td>2</td>
</tr>
</tbody>
</table>
## REPAIRED PROGRAM’S EXECUTION

### TEAM_P_CNT_LOG

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“A”</td>
<td>U3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>“A”</td>
<td>U1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>“A”</td>
<td>U3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>“A”</td>
<td>U4</td>
<td>1</td>
</tr>
</tbody>
</table>

### TEAM_P_CNT_LOG Diagram

- **TEAM_P_CNT_LOG**
- **t_id**: 1
- **t_name**: “A”
- **UUID**: U3
- **p_cnt_change**: 2
### TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>2</td>
</tr>
</tbody>
</table>

### TEAM_P_CNT_LOG

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“A”</td>
<td>U3</td>
<td>2</td>
</tr>
</tbody>
</table>

---

### TEAM

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>p_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>“A”</td>
<td>4</td>
</tr>
</tbody>
</table>

### TEAM_P_CNT_LOG

<table>
<thead>
<tr>
<th>t_id</th>
<th>t_name</th>
<th>UUID</th>
<th>p_cnt_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“A”</td>
<td>U1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>“A”</td>
<td>U3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>“A”</td>
<td>U4</td>
<td>1</td>
</tr>
</tbody>
</table>
- Program $P'$ is a sound refactoring of $P$ iff:
- Program $P'$ is a sound refactoring of $P$ iff:
  - for any execution $E'$ of $P'$ there is an execution $E$ of $P$ s.t.
- Program $P'$ is a sound refactoring of $P$ iff:
  - for any execution $E'$ of $P'$ there is an execution $E$ of $P$ s.t.
  - $E$ and $E'$ preserve the containment relation ($\sqsubseteq$) between initial and final DB states.
- Program $P'$ is a sound refactoring of $P$ iff:
  - for any execution $E'$ of $P'$ there is an execution $E$ of $P$ s.t.
  - $E$ and $E'$ preserve the containment relation ($\sqsubseteq$) between initial and final DB states
  - equivalent values returned
- 9 benchmarks
  - Number of statically identified anomalous access pairs

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Original + Eventual Consistency</th>
<th>Original + Causal Consistency</th>
<th>Original + Repeatable Read</th>
<th>Refactored + Eventual Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-C</td>
<td>30</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>SEATS</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>CW</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>SB</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>TWT</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>FMKe</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>SIB</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>WP</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>KC</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

ONE benchmark (avg 12%)

THREE benchmarks (avg 12%)

ALL benchmarks (avg 73%)
EVALUATION (APPLICABILITY + EFFECTIVENESS)

- 9 benchmarks
  ▪ Number of statically identified anomalous access pairs

ONE benchmark (avg 12%)
THREE benchmarks (avg 12%)
ALL benchmarks (avg 73%)

Original + Eventual Consistency
Original + Causal Consistency
Original + Repeatable Read
Refactored + Eventual Consistency

# anomalous access patterns

TPC-C  SEATS  CW  SB  TWT  FMKe  SIB  WP  KC
- 3 benchmarks
- Latency and throughput
  - EC, SC: original program (eventual consistency/serializability)
  - AT_EC, AT_SC: refactored program (eventual consistency/serializability)
**EVALUATION (PERFORMANCE)**

SmallBank

<table>
<thead>
<tr>
<th>Latency (x100 ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>150</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>250</td>
</tr>
</tbody>
</table>

**Latency vs Number of Clients**

- **EC, SC**: original program + eventual consistency and serializability
- **AT_EC, AT_SC**: original program + eventual consistency and serializability

Serializability is slow!

Refactoring enables performance (45% lower latency)

Refactoring has limited overhead — vs original
LACHESIS

(REPAIRING PARTITIONING ANOMALIES)
- Partitioned Database Clusters
  ▶ Same data-center deployment
- Partitioned Database Clusters
  - Same data-center deployment
  - Subset of records at each node
- Partitioned Database Clusters
  ▶ Same data-center deployment
  ▶ Subset of records at each node
- Partitioning Policy \( \Pi \)
  ▶ Performance
- Partitioned Database Clusters
  - Same data-center deployment
  - Subset of records at each node

- Partitioning Policy $\prod$
  - Performance
- Partitioned Database Clusters
  ▶ Same data-center deployment
  ▶ Subset of records at each node
- Partitioning Policy \(\Pi\)
  ▶ Performance
- Schism: [Curino et.al]

Schism: a Workload-Driven Approach to Database Replication and Partitioning

Carlo Curino  
curino@mit.edu

Evan Jones  
evanj@mit.edu

Yang Zhang  
yang@csail.mit.edu

Sam Madden  
madden@csail.mit.edu
- Correctness?
- Correctness?
- single-partition access = isolated & atomic
- Correctness?
- Single-partition access = Isolated & Atomic
- Eliminate concurrency anomalies by program refactoring
- Correctness?
- Single-partition access = Isolated & Atomic
- Eliminate concurrency anomalies by program refactoring
- Partitioning-aware symbolic execution
- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation

\[ P_0 \xrightarrow{\Pi} \text{Static Anomaly Detection} \]
INTRODUCTION TO LACHESIS

- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation
- Annotation & Runtime
  - Isolated { }
- Symbolic analysis engine
  ▪ program P0 (transactions & schema)
  ▪ partitioning relation

- Annotation & Runtime
  ▪ Isolated { }
Symbolic analysis engine
• program P0 (transactions & schema)
• partitioning relation

Annotation & Runtime
• Isolated { }

```
PlayerByTeam (t)

\[ ps := \text{select} * \text{from} \ \text{PLAYER} \]
\[ \text{where} \ \text{p}_t\text{_id}=t \]
\[ \text{foreach} \ p \ \text{in} \ ps \ \text{do} \]
\[ \text{print}(p.*) \]
```

<table>
<thead>
<tr>
<th>PLAYER</th>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>
- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation

- Annotation & Runtime
  - Isolated { }

```sql
ps := select * from PLAYER
    where p_t_id = t
foreach p in ps do
  print(p.*)
```

PlayerByTeam (t)

```
<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>
```

playerByTeam (t=T1)

```
... Select P0
...
Select P1
...```

Select P0

Select P1

...
INTRODUCTION TO LACHESIS

- Symbolic analysis engine
  ▶ program P0 (transactions & schema)
  ▶ partitioning relation

- Annotation & Runtime
  ▶ Isolated { }

```
PlayerByTeam (t)

p = select * from PLAYER
  where p_t_id = t
foreach p in ps do
  print (p.*)
```

```
Player

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p_id</td>
<td>p_t_id</td>
<td>p_role</td>
<td>p_stat</td>
</tr>
<tr>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>
```

```
setPlayer (p=P0, r=soldier)
  Set P0 to soldier

setPlayer (p=P1, r=captain)
  Set P1 to captain
```

playerByTeam (t=T1)

... Select P0

... Select P1

User
INTRODUCTION TO LACHESIS

- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation

- Annotation & Runtime
  - Isolated { }

### PlayerByTeam (t)

```
ps := select * from PLAYER
  where p_t_id=t
foreach p in ps do
  print(p.*)
```

### PLAYER

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>

PlayerByTeam (t=T1)

1. Select P0
2. Set P0 to soldier
3. Select P1
4. Set P1 to captain
INTRODUCTION TO LACHESIS

- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation

- Annotation & Runtime
  - Isolated { }

```
ps := select * from PLAYER
    where p_t_id=t
foreach p in ps do
    print(p.*)
```

```
PLAYER

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>
```

---

// P0, 1, captain, on
// P1, 1, captain, on

Invariant Violated!
- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation

- Annotation & Runtime
  - Isolated { }
- Symbolic analysis engine
  - program P0 (transactions & schema)
  - partitioning relation
- Annotation & Runtime
  - Isolated {}
- Recommended partitioning policy:

$$\forall r, r'. \ r.p\_t\_id = r'.p\_t\_id \Rightarrow (r, r') \in \Pi$$

<table>
<thead>
<tr>
<th></th>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>I</td>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>II</td>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>
- Recommended partitioning policy:

\[ \forall r, r'. \ r.p_t_id = r'.p_t_id \Rightarrow (r, r') \in \Pi \]

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>T1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>T1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>T2</td>
<td>soldier</td>
<td>off</td>
</tr>
</tbody>
</table>

1. Set P0 to soldier
2. Set P1 to captain
3. Select P0
4. Select P1
- Recommended partitioning policy:

\[ \forall r, r'.\; r.p_{\text{t_id}} = r'.p_{\text{t_id}} \Rightarrow (r, r') \in \Pi \]

**PlayerByTeam (t)**

```sql
Isolated {
    ps := select * from PLAYER
    where p_{\text{t_id}}=t

    foreach p in ps do
        print(p.*)
}
```

**After partitioning**

```sql
ps := select * from PLAYER
    where p_{\text{t_id}}=t

foreach p in ps do
    print(p.*)
```
PARTITIONING RECOMMENDATION

- Recommended partitioning policy:
  \[ \forall r, r'. r.p\_t\_id = r'.p\_t\_id \Rightarrow (r, r') \in \Pi \]
- Partitioning eliminates annotations
- Annotations are costly
- 40% fewer annotations:
  - average of 23% higher throughput & 14% lower latency
- Each table is partitioned for one access pattern

- Conflicting access patterns?

- $\forall r, r'. r.p_t_id = r'.p_t_id \Rightarrow (r, r') \in \Pi$
LIMITATION OF PARTITIONING

- Each table is
  - partitioned for one access pattern
- Conflicting access patterns?
  - $\forall r, r'. r.p_{t_id} = r'.p_{t_id} \Rightarrow (r, r') \in \Pi$
LIMITATION OF PARTITIONING

- Refactor the schema:
  ▶ A new table for each anomalous access pattern
  ▶ Rewrite program
  ▶ Repartition
- Running example:
  ‣ additional three tables
- Running example:
  - additional three tables
  - additional one transaction

GAME \times PLAYER

<table>
<thead>
<tr>
<th>g_id</th>
<th>p_id</th>
</tr>
</thead>
</table>

GAME

<table>
<thead>
<tr>
<th>g_id</th>
<th>g_stat</th>
<th>g_min_player</th>
</tr>
</thead>
</table>

PLAYER

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
</table>

playerByGame (g)

Game := select * from GAME where g_id=g
print(game.*)

ids := select p_id from GAME \bowtie PLAYER where g_id=g
foreach id in ids do
  p := select * from PLAYER where p_id=id
  print(p.*)
- Running example:
  ▶ additional three tables
  ▶ additional one transaction
- Anomaly is not fixed by partitioning

playerByGame (g)

Game := select * from GAME where g_id=g
print(game. *)

is := select p_id from GP where g_id=g
foreach id in is do
  p := select * from PLAYER where p_id=id
  print(p.* )

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>captain</td>
<td>off</td>
</tr>
</tbody>
</table>
- Running example:
  ▪ additional three tables
  ▪ additional one transaction

- Anomaly is not fixed by partitioning

**球员按游戏**

```
Game := select * from GAME where g_id=g
print(game. *)
```

```
ids := select p_id from G⋈P where g_id=g
foreach id in ids do
  p := select * from PLAYER where p_id=id
  print(p.*)
```

**PLAYER**

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>captain</td>
<td>off</td>
</tr>
</tbody>
</table>
- Running example:
  ▶ additional three tables
  ▶ additional one transaction
- Anomaly is not fixed by partitioning

```
playerByGame (g)

Game := select * from GAME where g_id=g
print (game.*)
ids := select p_id from G⋈P where g_id=g
foreach id in ids do
  p := select * from PLAYER where p_id=id
  print (p.*)
```

| PLAYER |
|---|---|---|---|
| p_id | p_t_id | p_role | p_stat |
| P0   | 1     | soldier | on    |
| P1   | 1     | captain | on    |
| P2   | 2     | captain | off   |

| PLAYER_BY_GAME |
|---|---|---|---|---|---|
| g_id | p_id | p_t_id | p_role | p_stat |
| G1   | P0   | 1      | soldier | on    |
| G1   | P2   | 2      | captain | off   |
EXAMPLE

- Running example:
  - additional three tables
  - additional one transaction

- Anomaly is not fixed by partitioning

**PLAYER**

<table>
<thead>
<tr>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>captain</td>
<td>on</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>captain</td>
<td>off</td>
</tr>
</tbody>
</table>

**GAME**

<table>
<thead>
<tr>
<th>g_id</th>
<th>g_stat</th>
<th>g_min_player</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>active</td>
<td>2</td>
</tr>
</tbody>
</table>

**PLAYER_BY_GAME**

<table>
<thead>
<tr>
<th>g_id</th>
<th>p_id</th>
<th>p_t_id</th>
<th>p_role</th>
<th>p_stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>P0</td>
<td>1</td>
<td>soldier</td>
<td>on</td>
</tr>
<tr>
<td>G1</td>
<td>P2</td>
<td>2</td>
<td>captain</td>
<td>off</td>
</tr>
</tbody>
</table>

---

```
playerByGame (g)

Game := select * from GAME where g_id=g
print(game.*)

ids := select p_id from G⋈P where g_id=g
foreach id in ids do
  p := select * from PLAYER_BY_GAME
  where p_id=id
  print(p.*)
```
- ZooKeeper runtime
- 11 Benchmarks
- ZooKeeper runtime
- 11 Benchmarks
- Number of annotations
  - 40% reduction
  - Additional 32% reduction
- ZooKeeper runtime
- 11 Benchmarks
- Number of annotations
  - 40% reduction
  - Additional 32% reduction
- Lower latency + Higher throughput
CONCLUSION
- Modern database systems come with a plethora of complex and subtle concurrency semantics.
- Modern database systems come with a plethora of complex and subtle concurrency semantics.
- The developers have a hard time reasoning about programs.
- Modern database systems come with a plethora of complex and subtle concurrency semantics.
- The developers have a hard time reasoning about programs.
- A symbolic program analysis engine to identify and report undesirable behaviors from a given database program
- Modern database systems come with a plethora of complex and subtle concurrency semantics.
- The developers have a hard time reasoning about programs.
- A symbolic program analysis engine to identify and report undesirable behaviors from a given database program
- By analyzing the access patterns of the database program, the program can be rewritten to be optimized for deployment on a particular clustering architecture.
- Modern database systems come with a plethora of complex and subtle concurrency semantics.

- The developers have a hard time reasoning about programs.

- A symbolic program analysis engine to identify and report undesirable behaviors from a given database program

- By analyzing the access patterns of the database program, the program can be rewritten to be optimized for deployment on a particular clustering architecture.

- We report results from empirical experiments to support our claims.