Partiqle: Relational Queries Over Program Traces

Simon Goldsmith
Robert O'Callahan
Alex Aiken

OOPSLA 2005
October 20, 2005
Does `doTransaction` call `sleep`?

```
public class DB {
    void doTransaction() {
        (new B()).y();
    }
}
public class B {
    void y() { sleep(); }
    void sleep() {
    }
}
```

- Obviously yes for this example
- How might one find out?
public class DB {
    public static boolean active = false;
    void doTransaction() {
        active = true;
        (new B()).y();
        active = false;
    }
}

public class B {
    void y() { sleep(); }
    void sleep() {
        if (DB.active) {
            println("call to sleep()!");
        }
    }
}
Failings of Manual Instrumentation

- Easy to get wrong
  - recursion, exceptions, threads
- Managing lots of data
- Non-local
  - hard to maintain
More Generally...

- How does one answer questions about program behavior?
- For example
  - Does `doTransaction` call `sleep`?
  - Does my program leak resources?
  - Does it use the API correctly?
  - Does it pass a `null` pointer to method `foo`?
Solution

a query language over program traces
Terminology

• An **event** is a method call, object allocation, etc.

• A **program trace** is a sequence of time-stamped **events** that happen during a given program's execution.

• A **query** is an SQL query against the **program trace** regarded as a table of **events**.
Artifacts

• Program Trace Query Language (PTQL)
  – a query language over program traces
  – subset of SQL => familiar, declarative

• Partiqle compiler
  – compiles PTQL query to optimized instrumentation of Java bytecode
  – instrumentation outputs query results as they become available
Does `doTransaction` call `sleep`?

```
SELECT sleep.backTrace
FROM MethodInvoc('DB.doTransaction') trans
JOIN MethodInvoc('B.sleep') sleep
    ON trans.thread = sleep.thread
    AND trans.startTime < sleep.startTime
    AND sleep.startTime < trans.endTime
```
Advantages

• Partiqle manages the data

• Partiqle instrumentation is general
  – it works in the presence of threads, exceptions, recursion

• You write a declarative PTQL query
  – not a new dynamic analysis tool
  – not manual instrumentation
**Program Trace Query Language (PTQL)**

- Regard program trace as tables:
  - `MethodInvoc`
  - `ObjectAlloc`
- Event happens => record in table
  - A call to `foo()` adds a record to `MethodInvoc`
- PTQL = SQL query over this schema
Example PTQL Query I

- What methods does method foo call?

```
SELECT m.*
FROM MethodInvoc('foo') foo
JOIN MethodInvoc m
  ON m.thread = foo.thread
  AND foo.startTime < m.startTime
  AND m.endTime < foo.endTime
```
Example PTQL Query II

- Show streams closed >1s after the last read/write

```sql
SELECT close.*
FROM MethodInvoc('read'|'write') rw
JOIN MethodInvoc('close') close
    ON rw.receiver = close.receiver
    AND close.endTime > rw.endTime + 1000
ANTIJOIN MethodInvoc nrw('read'|'write')
    ON nrw.receiver = rw.receiver
    AND rw.endTime < nrw.endTime
    AND nrw.endTime < close.endTime
```
Example PTQL Query III*

• Look for SQL injection attacks

```java
SELECT tainted.result
FROM MethodInvoc('HttpServletRequest.getParameter') tainted
JOIN MethodInvoc('Connection.execute') exec
    ON tainted.result = exec.param1
```
Example PTQL Query III*

- Ok if you check input before calling `execute`

```sql
SELECT tainted.result
FROM MethodInvoc('HttpServletRequest.getParameter') tainted
JOIN MethodInvoc('Connection.execute') exec
ON tainted.result = exec.param1
ANTIJOIN MethodInvoc('Util.inputOk') check
ON check.param1 = tainted.result
AND check.result = true
AND check.endTime < exec.startTime
```
Partiqle: Overview

- Compiles PTQL query to instrumentation
  - Record “interesting” events in runtime tables
    - Those that might contribute to query results
  - Search tables for query results
    - Sets of events that match the query
Does `doTransaction` call `sleep`?

SELECT sleep.backTrace
FROM MethodInvoc('DB.doTransaction') trans
JOIN MethodInvoc('B.sleep') sleep
  ON trans.thread = sleep.thread
AND trans.startTime < sleep.startTime
AND sleep.startTime < trans.endTime

• Query result = 2 events
  - a call to `doTransaction`
  - and a call to `sleep`
Recording Events

- Instrument code that may generate events

Query

```
FROM MethodInvoc('DB.doTransaction') trans
```

Code

```
void doTransaction() {
    b.y();
}
```
Recording Events

- Instrument code that may generate events
- to add events records to the runtime tables

```java
void doTransaction() {
    trans_Record r;
    synchronized(partiqleLock) {
        r = trans_Table.add(getTime(), getThread());
    }
    try {
        b.y(); // method body
    } finally {
        synchronized(partiqleLock) {
            r.setEndTime(getTime());
        }
    }
}
```
Timing

➢ In what order must the events happen?

Query

\[
\text{trans.startTime} < \text{sleep.startTime} \\
\text{AND sleep.startTime} < \text{trans.endTime}
\]

dotransaction

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleep</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Query Evaluation

- Any event that may be last triggers query evaluation

```java
void sleep() {
    // method body
}

void sleep() {
    queryEval( getThread(),
               getTime() );
    // method body
}
```
Query Evaluation

- Query evaluation searches runtime tables for matching events

```java
void queryEval(int threadId, long now) {
    synchronized(partiqueLock) {
        foreach r in trans_Table {
            if ( threadId == r.threadId
                && r.startTime < now
                && r.endTime > now ) {
                print getBackTrace();
            }
        }
    }
}
```
Optimization

- Finished calls to `doTransaction` cannot contribute to query results

```java
void doTransaction() {
    trans_Record r;
    synchronized(partiqleLock) {
        r = trans_Table.add(getTime(), getThread());
    }
    try {
        b.y(); // method body
    } finally {
        synchronized(partiqleLock) {
            r.setEndTime(getTime());
            trans_Table.delete(r);
        }
    }
}
```
Optimization

- Finished calls to `doTransaction` cannot contribute to query results

```java
void doTransaction() {
    trans_Record r;
    synchronized(partigleLock) {
        r = trans_Table.add(getThread());
    }
    try {
        b.y(); // method body
    } finally { synchronized(partigleLock) {
        trans_Table.delete(r);
    } }
}
```
Optimization

- Finished calls to `doTransaction` cannot contribute to query results

```java
void queryEval(int threadId, long now) {
    synchronized(partiqleLock) {
        foreach r in trans_Table {
            if ( threadId == r.threadId
                && r.startTime < now
                && r.endTime > now ) {
                print getBackTrace();
            }
        }
    }
}
```
Optimization

- Finished calls to `doTransaction` cannot contribute to query results

```java
void queryEval(int threadId) {
    synchronized(partiqleLock) {
        foreach r in trans_Table {
            if (threadId == r.threadId {
                print getBackTrace();
            }
        }
    }
}
```
Runtime Table for \texttt{trans\_Table}

- Store only essential fields
  - just \texttt{thread}

- Support only necessary operations
  - \texttt{add(thread)}, \texttt{delete(thread)}, \texttt{iterate(thread)}

- Pick reasonable data structure
  - map from \texttt{thread} to an integer counter
    - \texttt{add} => increment
    - \texttt{delete} => decrement
Partiqle: Compilation Summary

- Generate specialized data structures to store event records
- Instrumentation to create and store event records
- Generate query evaluation code
Experiments: Queries

- **DelayedClose**
  - Show streams closed >1s after the last read/write
  - Looked at Tomcat-specific stream class

- **StringConcats**
  - No `s=s+“stuff”` many times in a row

- **HashCode**
  - An object's `hashCode` does not change
  - Important if it is in a `Hashtable`
Experiments: Programs

• Ran queries on
  – Apache Tomcat (web server / Java servlets) (17k methods)
  – SpecJVM98 benchmarks
  – Some microbenchmarks

• Measured slowdown and memory footprint
Bugs Found

• Found several performance bugs (string concats)
  – Jack (SpecJVM98 benchmark)
  – Apache Tomcat's XML parser
  – IBM JDK

• Found correct, but subtle code
  – Hash code consistency in Xerces XML parser
Related Work

- Aspect Oriented Programming Languages
  - Tracematches (talk before previous talk)
- Other trace-based query engines
  - PMMS (Liao & Cohen, 1992)
  - PQL (previous talk)
- Program Monitors
  - Eagle (Barringer et al., RV 2004)
- DIDUCE / Daikon / Statistical Debugging
Conclusion

**PTQL**: declarative query language over program traces

**Partiqle**: compiles PTQL to Java bytecode instrumentation

+ answers to questions about program behavior
Thanks!

• Thanks to
  – Michael Martin et al. (PQL) and
  – Oege de Moor et al. (Tracematches)
for sharing early drafts of their papers