WeBridge: Synthesizing Stored Procedures for Large-Scale Real-World Web Applications

(Extra Material)

Abstract

This document contains the extra material of SIGMOD'24 submission #347. The main content is the formal proof of the correctness of WeBridge.

1 Correctness Proof of WeBridge

In this section, we show that the transformation performed by WeBridge preserves the semantic of the original web application, which is the Theorem 1.2 in this section.

We first introduce some preliminary concepts. The program states of the original application API is denoted as $\langle D, S\rangle$, where D represents the database state, S represents the application's heap and local variable state. WeBridge extends the program states of the original application with a query result buffer B. Thus, the program states become $\langle D, S, B\rangle$. Additionally, let C_{sp} be the stored procedures WeBridge generated for the original application. Given the definition of program states, we now describe how the application is evaluated upon each user request. Let I be the set of all possible request inputs of an API. The input contains the request parameters for the API, and a system environment (which determine the return values of native method calls). For each input $i \in I$, suppose the initial state of the API is $\langle D, S\rangle$, the final state of the API is $\langle D, S\rangle$, then evaluating the API with input i is denoted with: $\langle D, S\rangle \xrightarrow{i} \langle D', S'\rangle$. Similarly, evaluating the API transformed by WeBridge is denoted with $\langle D, S, B\rangle \xrightarrow{i} \langle D', S', B'\rangle$. During the above evaluation, the program will execute along a *path* (Section 4 in the paper) P, which is a finite sequence of n boolean branch decisions denoted as $P = [b_1, b_2, ..., b_n]$. If $b_i = 1$, it signifies the i-th branch decision took the '$then$' branch; otherwise the '$else$' branch was taken. Let Π be the set of all possible paths and Φ be the set of paths optimized by WeBridge, where $\Phi \subset \Pi$. A hot path is a path p where $p \in \Phi$, a cold path p_c is a path where $p_c \in \Pi \wedge p_c \notin \Phi$. Additionally, the execution of each path of the program will lead to a sequence of SQL statements issued to the database. Each SQL statements consists of a *template* string and a list of *parameters*. We use the following notation: $\langle D, S\rangle \overset{i}{\Rightarrow} Q$ to represent that evaluating the original application API with initial state $\langle D, S\rangle$ on input i produces the sequence of SQL statements Q. We use $Q[i]$ to denote the i-th SQL statement in Q, where $i \in \{1, \ldots, |Q|\}$. Similarly, for the application API transformed by WeBridge, we have $\langle D, S, B\rangle \overset{i}{\Rightarrow} Q'$, where Q' is the sequence of executed SQL statements.

We next define and proof the following lemma based on the above definitions.

Lemma 1.1. *Given an API program C and initial states D_0, S_0 and B_0, let I be the set of all possible inputs for C. $\forall i \in I$, If $\langle D_0, S_0\rangle \overset{i}{\Rightarrow} Q$ under the original application, $\langle D_0, S_0, B_0\rangle \overset{i}{\Rightarrow} Q'$ under the application transformed by WeBridge, then $Q = Q'$.*

Proof

The premise contains

$$\langle D_0, S_0\rangle \overset{i}{\Rightarrow} Q \tag{1}$$

and

$$\langle D_0, S_0, B_0 \rangle \stackrel{i}{\Rightarrow} Q' \tag{2}$$

The conclusion is

$$Q = Q' \tag{3}$$

The conclusion is proved by classifying the type of the path taken by C into two cases.

- A Hot path p. By definition of the hot path, we have

  $$p \in \Phi \tag{4}$$

  indicating that C is taking a path that have already been optimized by WeBridge. If p is optimized by WeBridge, then all the database accesses along the path are replace with calls to C_{sp}, which is the stored procedures generated by WeBridge for p. Let Q_{sp} be the sequence of SQL statements of C_{sp}, in which the SQL statement template string and parameters are extracted from Algorithm 1 by concolic execution. Since the concolic execution in Algorithm 1 is done by a deterministic reply of the hot path p, which implies that the collected concrete SQL templates in Q_{sp} must be the same with Q. Along with (1), we have:

  $$|Q| = |Q_{sp}| \tag{5}$$

  and

  $$\forall k \in \{1, \ldots, |Q|\}.\ Q_{sp}[k].template = Q[k].template \tag{6}$$

  For the query parameters, since WeBridge assumes the absence of global application states, the value of parameters are only determined by the external input states (Section 4 in the paper). All these states have been symbolized and their related computations are tracked in symbolic form via Algorithm 1. These symbolic computations are then transformed into equivalent stored procedure code in C_{sp} by Algorithm 4, by transformation rules that WeBridge assume to be semantic preserving. Consequently, any computations that might change the value of a parameter have been transformed into equivalent computations in the stored procedure. Therefore, given the same set of input states to the original application and C_{sp}, the parameter values computed from these input states must be the same:

  $$\forall k \in \{1, \ldots, |Q|\}.\ Q_{sp}[k].parameters = Q[k].parameters \tag{7}$$

  By the definition of a SQL statement, (5), (6) and (7), we have:

  $$Q_{sp} = Q \tag{8}$$

Since p is a hot path, invocations of all the SQL statements are replaced with calls to C_{sp}. Meanwhile, the path conditions for all SQL statements in Q_{sp} should evaluate to `true`, which means that all SQL statements in Q_{sp} will execute. Along with (2), we have:

$$Q_{sp} = Q' \tag{9}$$

By (8) and (9), the conclusion (3) is proved in this case.

- A Cold path p_c. By definition of cold path, we can further divide the type of cold path into two cases.
  - $\forall p' \in \Phi.\ head(p') \neq head(p_c)$. In this case, the cold path p_c diverges on the first branch decision of hot paths. Let $b = head(p_c)$ for the cold path, then the first branch decision for all hot paths must be $!b$, indicating that path conditions of SQL statements after the first branch decision should evaluate to `false`. Thus, no SQL statement in C_{sp} after the first branch decision will execute. For SQL statements before the first branch decision, there are two cases.
    - No SQL statements in C_{sp} before the first branch decision. In this case, no SQL statement will be issued by C_{sp}, and the application trivially fallbacks to normal execution to issue all SQL statements in interactive mode. This indicates that $Q = Q'$, and the conclusion (3) is proved in this case.
    - Q_p is the sequence of SQL statements before the first branch decision in C_{sp}, where $|Q_p| > 0$. In this case, the path conditions for Q_p in C_{sp} will all be `true`, indicating that these SQL statements will execute unconditionally for both hot paths and cold paths. Thus, with (8) and (9) we have:

      $$\forall i \in \{1, ..., |Q_p|\}.\ Q_p[i] = Q[i] \tag{10}$$

      and since SQL statements in Q_p are executed by C_{sp}, and by (2), we have:

      $$\forall i \in \{1, ..., |Q_p|\}.\ Q_p[i] = Q'[i] \tag{11}$$

      For SQL statements after the first branch decision b in C_{sp}, their path conditions will include $!b$, evaluating to false. This indicates that no SQL statement in C_{sp} will execute after the first branch decision. Thus, the application fallbacks to normal execution to issue the following SQL statements in interactive mode. Then, we have:

      $$\forall i \in \{|Q_p| + 1, |Q|\}.\ Q[i] = Q'[i] \tag{12}$$

      By (10), (11) and (12), the conclusion (3) is proved in this case.

– $\forall p' \in \Phi.\ head(p') = head(p_c)$. In this case, the cold path p_c and hot path p' "share" a common prefix of branch decisions. The hot path p' that has the longest prefix of branch decisions with p_c is:

$$\begin{aligned} \exists p' \in \Phi, i \in \{2, ..., |p'|\}, \forall j \in \{1, ..., i-1\}. \\ p'[j] = p_c[j] \wedge p'[i] \neq p_c[i] \wedge \\ (\nexists p'' \in \Phi. p''[i] = p_c[i]) \end{aligned} \tag{13}$$

By (4) and (13), C_{sp} contains all the SQL statements, which is denoted with Q_p, for path p'. Then, since p_c and p' share a common prefix branch decisions, by (13) we know that only the SQL statements in Q_p before the i-th branch decision will execute along path p_c. Let these SQL statements be Q'_p, then we know that:

$$\forall i \in \{1, \ldots, |Q'_p|\}. Q'_p[i] = Q[i] \tag{14}$$

and since SQL statements in Q'_p are executed by C_{sp}, and by (2), we have:

$$\forall i \in \{1, ..., |Q'_p|\}.\ Q'_p[i] = Q'[i] \tag{15}$$

For SQL statements after the i-th branch decision b for p_c in C_{sp}, their path conditions will include $!b$, evaluating to false. This indicates that no SQL statement in C_{sp} will execute after the i-th branch decision. Thus, the application fallbacks to normal execution to issue the following SQL statements in interactive mode. Then, we have:

$$\forall i \in \{|Q'_p| + 1, |Q|\}.\ Q[i] = Q'[i] \tag{16}$$

By (14), (15) and (16), the conclusion (3) is proved in this case.

Then we have proved the conclusion (3).

Next, we prove the Theorem 1.2.

Theorem 1.2. *Given an API program C and initial states D_0, S_0 and B_0, let I be the set of all possible inputs for C. $\forall i \in I$, if $\langle D_0, S_0 \rangle \xrightarrow{i} \langle D_c, S_c \rangle$ under the original application, $\langle D_0, S_0, B_0 \rangle \xrightarrow{i} \langle D_{c'}, S_{c'}, B_{c'} \rangle$ under the application transformed by WeBridge, then $D_c = D_{c'} \wedge S_c = S_{c'}$.*

Proof

The premise contains

$$\langle D_0, S_0 \rangle \xrightarrow{i} \langle D_c, S_c \rangle \tag{17}$$

under the original application, and

$$\langle D_0, S_0, B_0 \rangle \xrightarrow{i} \langle D_{c'}, S_{c'}, B_{c'} \rangle \tag{18}$$

under the application transformed by WeBridge.

The conclusion includes

$$D_c = D_{c'} \tag{19}$$

and

$$S_c = S_{c'} \tag{20}$$

By Lemma 1.1, (17) and (18), we know that the SQL statements issued by the original application and the application transformed by WeBridge are the same. Since the initial database state D_0 could only updated by the SQL statements, the database state after executing the same sequence of SQL statements should be the same. Thus, we have $D_c = D_{c'}$, and conclusion (19) is proved.

From Lemma 1.1, (17) and (18), we can conclude that the query execution results of each SQL statement in the original application and the application transformed by WeBridge are the same. Because WeBridge assumes the absence of global states in the applications, the external input states should be sufficient to uniquely determine a path. The input i and query execution results for the original application and the application transformed by WeBridge are exactly the same, so the external input states are also the same. Consequently, the application transformed by WeBridge will execute the same path with the original application. Thus, we have $S_c = S_{c'}$, and conclusion (20) is proved.