Effective Race Detection for Event-Driven Programs

by Raychev et al. [OOPSLA ‘13]

Papers We Love Panamá, 29 de octubre de 2019

Danilo Domínguez Pérez @danilo04

Agenda

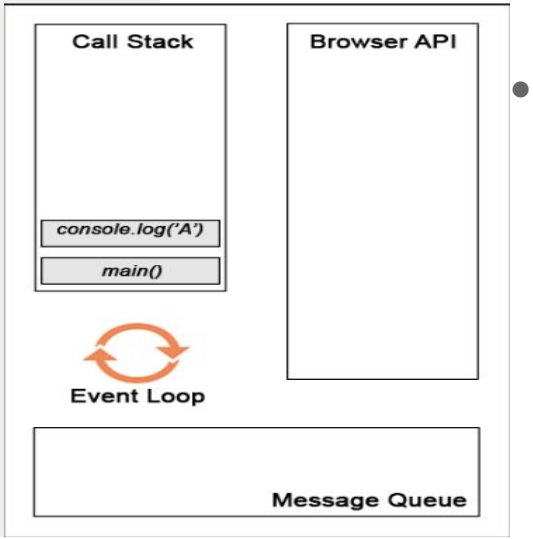
1. **Background**
   Sistemas Basados en Eventos en especial aplicaciones Web
2. **Problemas**
   Introducimos los problemas que el paper resuelve
3. **Soluciones**
   Presentamos las soluciones que el paper desarrolla
4. **Resultados**
   Estudiamos los experimentos presentados

Sistemas Basados en Eventos (Event-driven Systems)

- El flujo del programa determinado por eventos:
  - e.j. clicks, eventos de sensores, etc.
- Ejemplos de aplicaciones que utilizan esta arquitectura:
  - Desktop GUI Apps
  - **Web Apps**
  - Mobile Apps
- Aplicaciones registran **callbacks o event listeners** que se ejecutan como la acción de un evento
- Callbacks se ejecutan atómicamente en un **thread** único utilizando un **event loop**

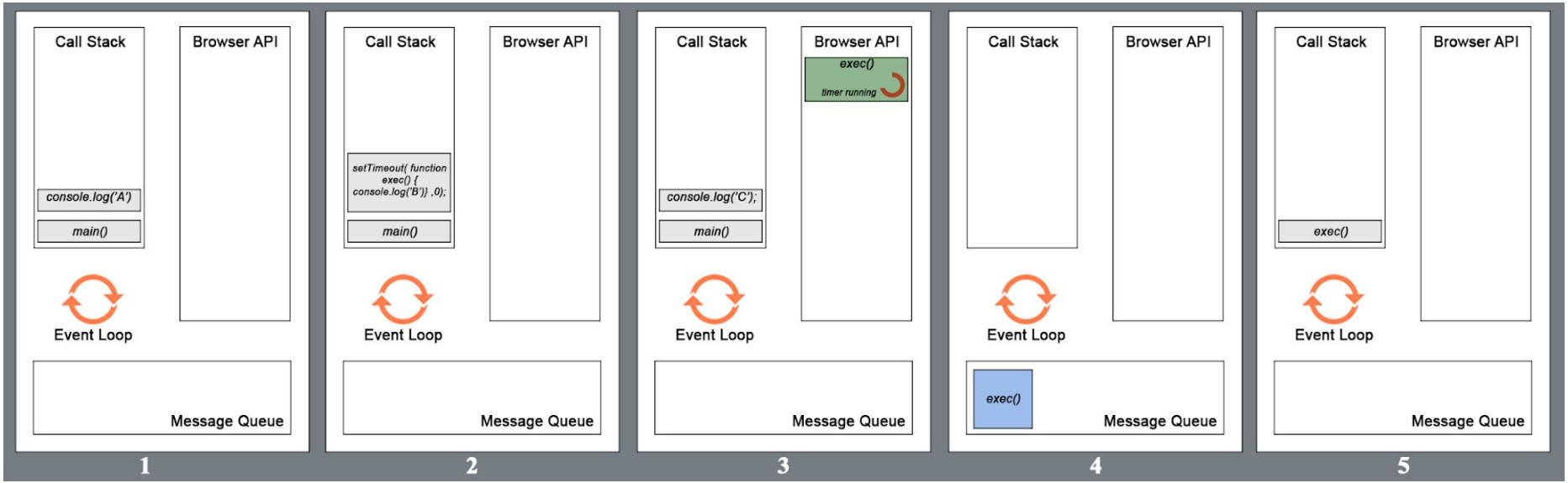
Event Loop en Aplicaciones Web

```
function main(){
  console.log('A');
  setTimeout(
    function display(){
      console.log('B');
    }, 0);
  console.log('C');
}
main();
//   Output
//   A
//   C
//   B
```



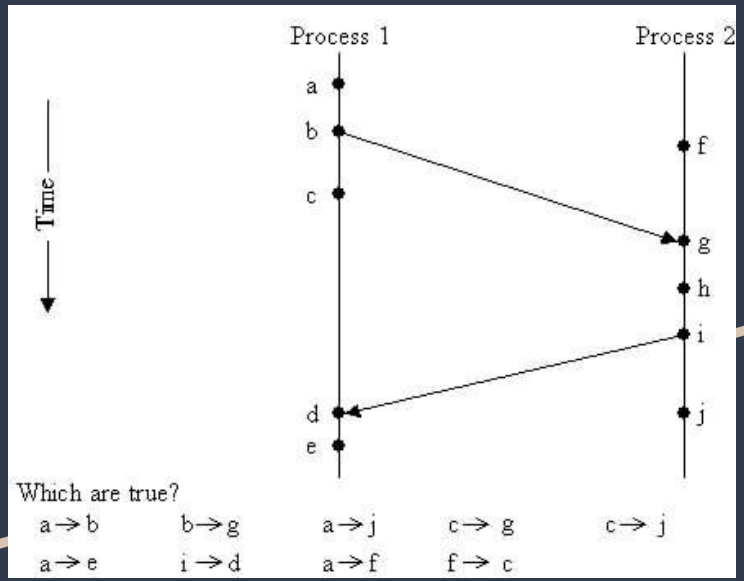
- Heap
  - se asignan (allocate) objetos
- Stack
  - Pila (LIFO) single threaded conformado por llamadas a funciones
- Message Queue
  - Cola de callbacks por ejecutar

Event Loop



Fuente: https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4

Logical Clocks



- Utilizados para determinar el orden de eventos en un sistema distribuido
- Provee un orden parcial de eventos
- Existen diferentes algoritmos:
  - Lamport Timestamps
  - **Vector Clocks**
  - Version Vectors
  - Matrix Clocks

Race Conditions

```
<html><body>
  <input type="button" id="b1"
       onclick="javascript:f()">
  ... <!-- many elements -->
  <script>
   function f() {
     if (init)
       alert(y.g);
     else
       alert("not ready");
   }
   var init = false, y = null;
  </script>
  ...
  <script>
  y = { g: 42 };
  init = true;
  </script>
</body></html>
```

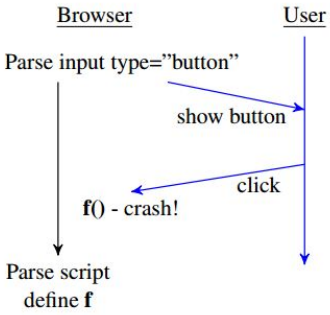


Figure 1. Example web page with both a harmful race and ad hoc synchronization. The trace on the right shows the harmful interleaving.

- Condición que ocurre cuando un sistema es dependiente de la **secuencia en la que se ejecutan eventos incontrolables**
- Se convierte en bug cuando un orden específico de estos eventos no es deseable o quiebra el sistema

Agenda

1. **Background**
   Sistemas Basados en Eventos en especial aplicaciones Web
2. **Problemas**
   Introducimos los problemas que el paper resuelve
3. **Soluciones**
   Presentamos las soluciones que el paper desarrolla
4. **Resultados**
   Estudiamos los experimentos presentados

Problemas

- En aplicaciones web es difícil identificar correctamente race conditions
- Sincronización **ad-hoc** incrementa posibilidad de falsos positivos
- Soluciones basadas en vector clocks **no son escalables**

```
<html><body>
  <input type="button" id="b1"
       onclick="javascript:f()">
  ... <!-- many elements -->
  <script>
   function f() {
     if (init)
       alert(y.g);
     else
       alert("not ready");
   }
   var init = false, y = null;
  </script>
  ...
  <script>
  y = { g: 42 };
  init = true;
  </script>
</body></html>
```

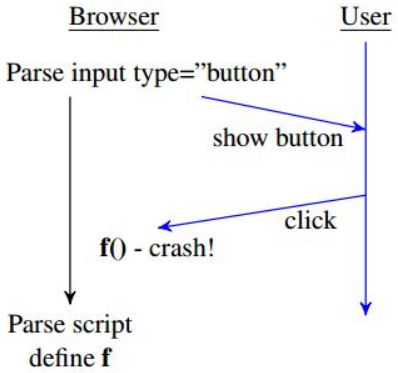


Figure 1. Example web page with both a harmful race and ad hoc synchronization. The trace on the right shows the harmful interleaving.

Agenda

1. **Background**
   Sistemas Basados en Eventos en especial aplicaciones Web
2. **Problemas**
   Introducimos los problemas que el paper resuelve
3. **Soluciones**
   Presentamos las soluciones que el paper desarrolla
4. **Resultados**
   Estudiamos los experimentos presentados

Contribuciones

- Introducen **race coverage** para identificar sincronización ad-hoc y evitar falsos positivos
- Implementan **chain decomposition** para detectar race conditions eficientemente utilizando vector clocks

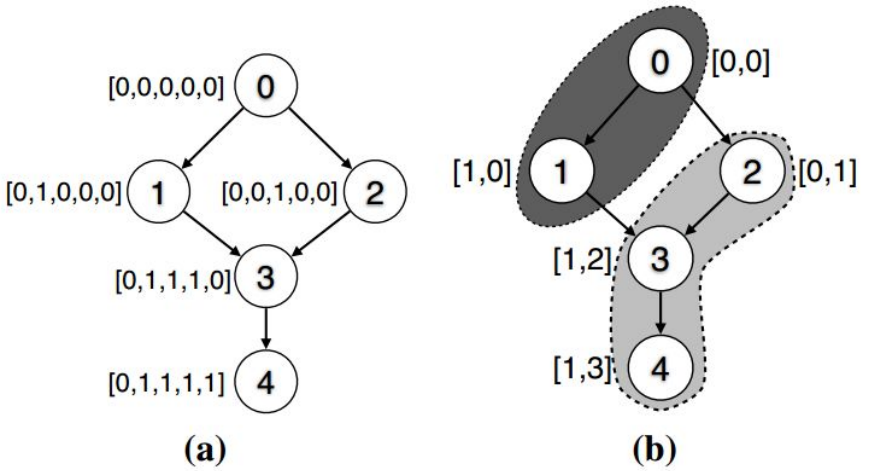
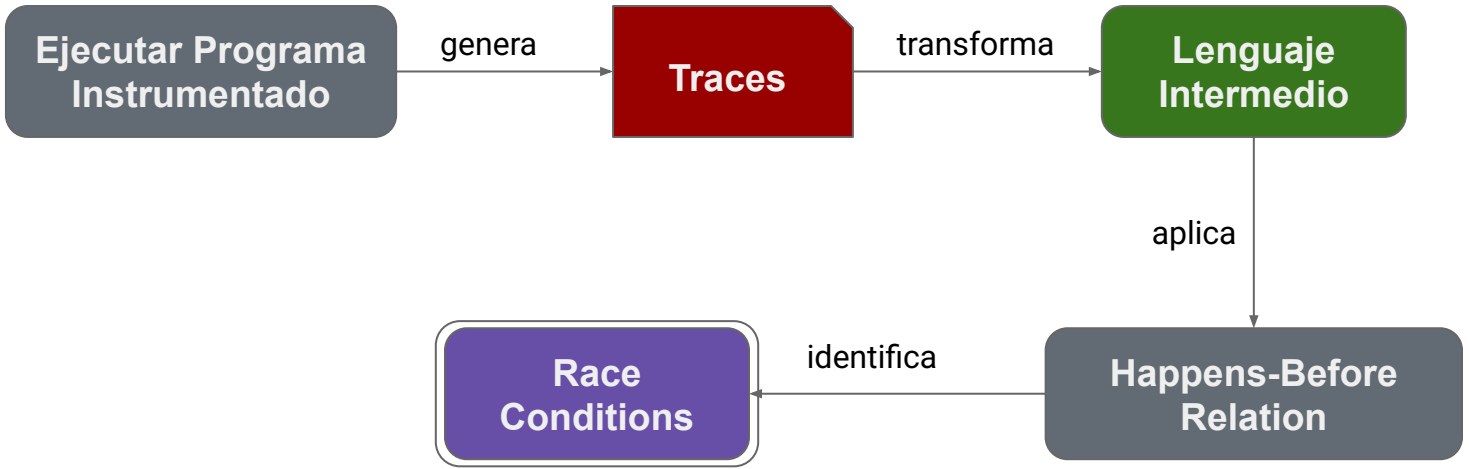


Figure 6. Example showing vector clocks with and without chain decomposition.

Abstracción del Problema

Flujo de Ejecución



Lenguaje Basado en Eventos

$$
\begin{array}{rcl}
S & ::= & S;S \mid rd(t,x) \mid wr(t,x) \mid \\
 & & fork(t,u,EventAction) \\
 & & \\
EventAction & ::= & Joins; begin(t); S; end(t) \\
 & & \\
Joins & ::= & Joins; Joins \mid join(t,u) \\
 & & \\
Program & ::= & EventAction \\
 & & \\
Operation & ::= & rd(t,x) \mid wr(t,x) \mid begin(t) \mid end(t) \mid \\
 & & fork(t,u,EventAction) \mid join(t,u) \\
 & & \\
t,u & \in & EventIds \\
x & \in & Vars \\
a,b & \in & Operation
\end{array}
$$

Figure 2. The *Event* language

Lenguaje Basado en Eventos

- **rd(t, x)**: event action **t** performs a read of shared variable **x**
- **wr(t, x)**: event action **t** performs a write of shared variable **x**
- **fork(t, u, EventAction)**: event action **t** forks event action **u** that executes **EventAction**
- **join(t, u)**: event action **t** waits until event action **u** completes
- **begin(t)** and **end(t)**: denotes beginning and ending of event action **t**

Ejemplo de Abstracción a Lenguaje

```
<html><body>
  <input type="button" id="b1"
      onclick="javascript:f()">
  ... <!-- many elements -->
  <script>
   function f() {
     if (init)
       alert(y.g);
     else
       alert("not ready");
   }
   var init = false, y = null;
  </script>
  ...
  <script>
  y = { g: 42 };
  init = true;
  </script>
</body></html>
```

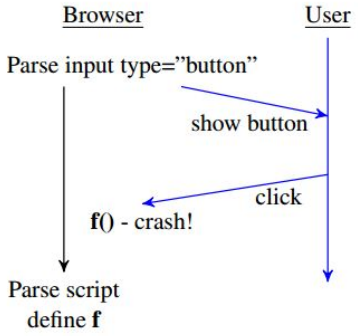


Figure 1. Example web page with both a harmful race and ad hoc synchronization. The trace on the right shows the harmful interleaving.

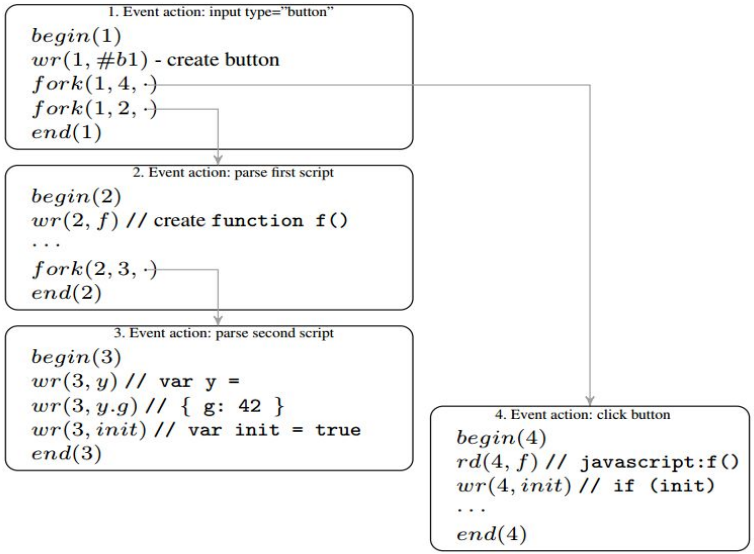


Figure 3. Example trace for the program in Fig. 1 translated to the *Event* language. Some details are missing for clarity.

Traces

Trace

$$\pi = a_0 \cdot a_1 \cdot \ldots \cdot a_n \in \text{Operations}^* \text{ where } n \geq 0$$

Trace Order

1. Para un trace **π**, si un evento **a** ocurre antes que **b** en **π**, entonces **a** $<_\pi$ **b**
2. Un acción **t** precede otra acción de evento **u** si **begin(t)** $<_\pi$ **end(u)**

Happens-Before Relation

Dado un trace **π** la relación happens-before acciones de eventos **t** y **u** es la relación mínima transitiva:

- t = u
- fork(t, u) ∈ π
- join(u, t) ∈ π

Dado **ev(a)** como la acción de evento de una operación **a**, **a** ≤ **b** es cierto si:

- ev(a) ≠ ev(b) and ev(a) ≤ ev(b), or
- ev(a) = ev(b) and a < π b

Race

Definition 2.1 (Race). *Given a trace π and operations a and b where $a <_\pi b$, a race $R = (a, b)$ is a pair where both operations access the same variable, at least one is a write and $a \not\leq b$.*

Race Coverage

Definition 3.1 (Covered race). *We say that a race $R = (a, b)$ is covered by race $S = (c, d)$ if the following conditions hold:*

1. $ev(a) \preceq ev(c)$, *and*
2. $d \preceq b$

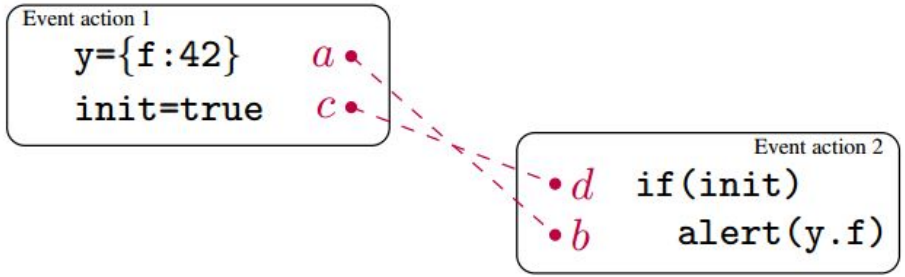


Figure 4. An example of race coverage based on Fig. 1. Race $R = (a, b)$ is covered by race $S = (c, d)$, since with a happens-before edge from c to d, R would clearly not be a race.

Uncovered Races

race(π) es el conjunto de todas las races en el trace **π**

$$uncovered(\pi) = \{R \,|R \in races(\pi), \nexists C \subseteq races(\pi) : C \blacktriangleleft R\}$$

Calcular Uncovered Races

- Calcular Happens-Before via graph connectivity
- Un grafo G = (V , E) para un trace π:
  - V = {t | begin(t) ∈ π}
  - E = {(t, u) | fork(t, u, _) ∈ π or join(u, t) ∈ π}
- Detecta races determinando si dos operaciones que acceden la misma variable están conectadas en **G**

Generar el grafo de conectividad

- Breadth-first-search (BFS)
  - Time complexity O(|E|) pero más lento que Vector Clocks
- Vector Clocks
  - Más eficiente que BFS pero complejidad de memoria $O(|V|^2)$
  - Aplican **chain decomposition** para reducir la complejidad de memory

Agenda

1. **Background**
   Sistemas Basados en Eventos en especial aplicaciones Web
2. **Problemas**
   Introducimos los problemas que el paper resuelve
3. **Soluciones**
   Presentamos las soluciones que el paper desarrolla
4. **Resultados**
   Estudiamos los experimentos presentados

Resultados

Metric	Number of variables with races			
	Mean	Median	90-th %-ile	Max
All	634.6	461	1568	3460
Removed by single coverage (Definition 3.1)	581.1	419	1542	3389
Removed by multi-coverage (Definition 3.2)	8.2	2	30	55
Remaining with uncovered races	45.3	29	103	331
Filtering methods				
Writing same value	0.75	0	3	12
Only local reads	3.42	2	8	43
Late attachment of event handler	16.7	8	41	117
Lazy initialization	4.3	0	11	61
Commuting operations - className, cookie	4.0	1	8	80
Race with unload	1.1	0	2	33
Remaining after filters from Section 5.3	17.8	10	38	261
Uninitialized values	1.9	0	5	36
readyStateChange handler	1.3	0	2	64

Table 1. Usability metrics of EVENTRACER on the Fortune 100 sites.

Resultados

Metric	Mean	Median	Max
Number of event actions	5868	2496	114900
Number of edges	6822	2873	122240
Number of chains	175	134	792

Connectivity algorithm	Running time in seconds		
Breadth-first search	>22.2	>0.4	TIMEOUT
Vector clocks w/o chain decomposition	>0.068	>0.011	OOM
Bit vector clocks	0.081	0.008	3.381
Vector clocks + chain decomposition	**0.043**	**0.007**	**2.395**

Table 3. Performance metrics of EVENTRACER on finding uncovered races in the Fortune 100 sites.

Metric	Mean	Median	Max
Trace file size (uncompressed)	7.9MB	3.7MB	129.5MB
Trace file size (gzip compressed)	1.4MB	0.7MB	16.3MB
Vector clocks memory consumption			
Vector clocks w/o chain decomposition	544MB	12MB	25181MB
Bit vector clocks	33MB	1MB	1573MB
Vector clocks + chain decomposition	**5MB**	**1MB**	**171MB**

Table 4. Memory consumption of different race detection techniques.

Preguntas?