Análisis de técnicas de sincronización en estructuras de datos concurrentes

Álvaro Rubira García

Junio de 2024

UNIVERSIDAD DE MURCIA

Índice

Introducción

Problemas con los cerrojos

- Varios hilos que modifican datos compartidos
- Cerrojos (*mutex*)
- Problemas de rendimiento si hilos quedan bloqueados en secciones críticas
  - Interrupciones y cambios de contexto comunes en sistemas modernos

Introducción

Algoritmos *lock-free*

- Garantizan que *alguien* está progresando
  - Hilos bloqueados no impiden progreso del sistema
  - Ventaja con alta contención
- Basados en CAS o similares
  - Difíciles de programar
  - Propensos a errores

Introducción

Algoritmos *lock-free*

- Garantizan que *alguien* está progresando
  - Hilos bloqueados no impiden progreso del sistema
  - Ventaja con alta contención
- Basados en CAS o similares
  - Difíciles de programar
  - Propensos a errores
- Nuevos enfoques
  - Atómicos de varias direcciones de memoria
  - Memoria transaccional
  - *Lock-free locks*

Introducción

Atómicos de varias direcciones de memoria

- Limitaciones de CAS $\rightarrow$ Estructuras muy complejas
- Multi-address Compare-And-Swap (MCAS)

```
1  // Atómicamente
2  if ((*addr0 == old0) && (*addr1 == old1) && (*addr2 == old2) && ...){
3    *addr0 = new0;
4    *addr1 = new1;
5    *addr2 = new2;
6    ...
7    return true;
8  }
9  return false;
```

Introducción

Memoria transaccional

- Modelo de programación muy sencillo

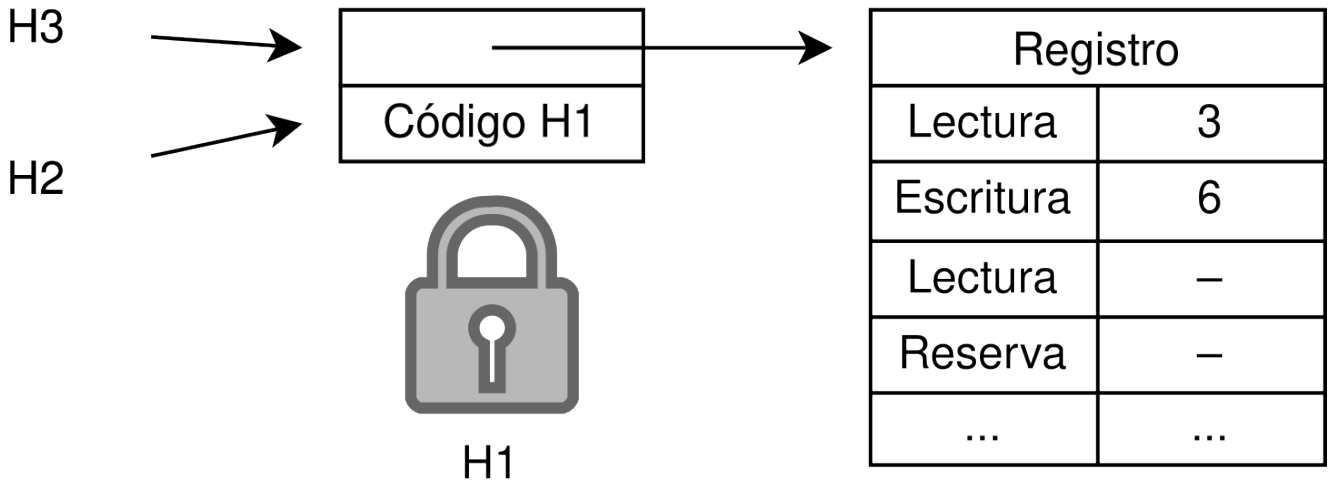
```
1       void push(int data) {
2         Node *new_node = new Node();
3         new_node->data = data;
4
5         TM_BEGIN();
6
7         new_node->prev = head;
8         head = new_node;
9
10        TM_END();
11      }
```

- Cambios especulativos hasta fase *commit*
- Algunas implementaciones *hardware* no son *lock-free*

Introducción

Lock-free locks

- Algoritmos *lock-free* partiendo de estructuras con cerrojos
- Cooperación entre hilos
- Librería `flock`[1]
  - Secciones críticas idempotentes gracias a *log* (registro)
  - Eficiente con cerrojos de grano fino



[1] Naama Ben-David, Guy E Blelloch, and Yuanhao Wei. Lock-free locks revisited. 2022.

Objetivos y metodología

- *mcas-benchmarks*[2]
  - Estructuras *lock-based*, *lock-free* CAS y *lock-free* MCAS
    - Lista ordenada
    - *Hash map*
    - *Deque*, pila y cola
    - Árbol de búsqueda binario
- Añadimos estructuras con nuevos mecanismos de sincronización
  - `flock`
  - HTM: partiendo de estructuras con cerrojos y MCAS
  - MCAS: variantes con cerrojos y HTM
- `https://github.com/AlvaroRGum/data-structure-benchmarks`

[2] Nodari Kankava. Exploring the efficiency of multi-word compare-and-swap. 2020.

Trabajo realizado

Incorporación de `flock`

- Adaptamos lista ordenada y *hash table* ya creadas
- Resto parten de estructuras con cerrojos de *mcas-benchmarks*

```
1   class Deque {
2    private:
3     struct Node {
4       int data;
5       flck::atomic<Node *> prev;
6       flck::atomic<Node *> next;
7       Node() = default;
8       Node(int data) : data(data), prev(nullptr), next(nullptr) {};
9     };
10
11    Node *head;
12    Node *tail;
13
14    flck::lock deque_lock;
15    flck::memory_pool<Node> node_pool;
```

Trabajo realizado

Deque con `flock`

```
1   void push_front(int data) {
2     flck::with_epoch([=] {
3       deque_lock.with_lock([=] {
4         Node *new_node = node_pool.new_obj(data);
5         Node *hn = (head->next).load();
6
7         new_node->next = hn;
8         new_node->prev = head;
9
10        hn->prev = new_node;
11        head->next = new_node;
12
13        return true;
14      });
15    });
16  }
```

Trabajo realizado

Incorporación de HTM lock

- Hardware Transactional Memory (HTM) con Intel TSX y cerrojo de respaldo
- HTM lock (partiendo de *lock-based*)

```
1   void push_front(int data) {
2     Node *new_node = new Node();
3     new_node->data = data;
4
5     TM_BEGIN();
6
7     new_node->next = head->next;
8     new_node->prev = head;
9
10    head->next->prev = new_node;
11    head->next = new_node;
12
13    TM_END();
14  }
```

Trabajo realizado

Incorporación de HTM MCAS

- HTM partiendo de MCAS

```
1   int pop_front() {
2     while (true) {
3       Node* lh = LeftHat;
4       Node* lhL = lh->L;
5       Node* lhR = lh->R;
6
7       if (lhL == lh) {
8         if (LeftHat == lh) return -1;
9       } else {
10        if (tcas(&LeftHat, lh, lhR,
11                 &lh->R, lhR, lh,
12                 &lh->L, lhL, lh))) {
13          int result = lh->data;
14          return result;
15        }
16      }
17    }
18  }
```

Trabajo realizado

Incorporación de HTM MCAS

```
1   int pop_front() {
2     int result;
3     TM_BEGIN();
4
5     Node* lh = LeftHat;
6     Node* lhL = lh->L;
7     Node* lhR = lh->R;
8
9     if (lhL == lh) {
10      result = -1;
11    } else {
12      LeftHat = lhR;
13      lh->R = lh;
14      lh->L = lh;
15      result = lh->data;
16    }
17
18    TM_END();
19
20    return result;
21  }
```

Trabajo realizado

Incorporación de nuevas versiones de MCAS

- Cerrojo global

```
1   uint64_t dcas(uint64_t* addr0, uint64_t old0, uint64_t new0,
2                 uint64_t* addr1, uint64_t old1, uint64_t new1) {
3     std::lock_guard<std::mutex> lock(global_cas_lock);
4     if ((*addr0 == old0) && (*addr1 == old1)) {
5       *addr0 = new0;
6       *addr1 = new1;
7       return true;
8     }
9     return false;
10  }
```

- Otra opción: usar HTM

Pruebas y resultados

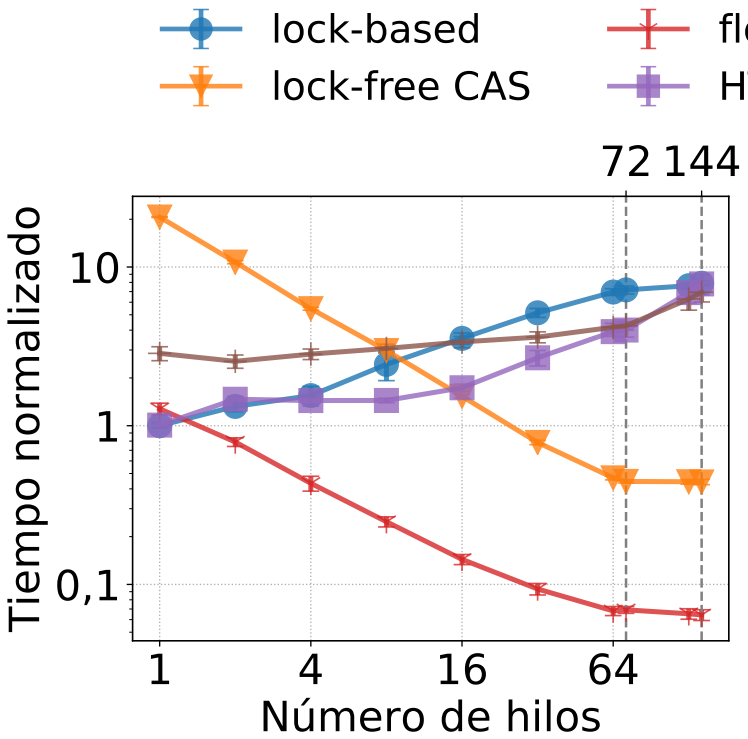
Configuración de los *benchmarks*

- 10^6 operaciones entre todos los hilos
- Pruebas con y sin *backoff* en HTM
- De 1 a 144 hilos
- Errores de estructuras originales
  - Conseguimos encontrar y resolver algunos
    - BST MCAS[3]
    - Estructuras *lock-based*
  - No arreglados impiden ejecutar *benchmarks* con MCAS de lista ordenada y *hash map*

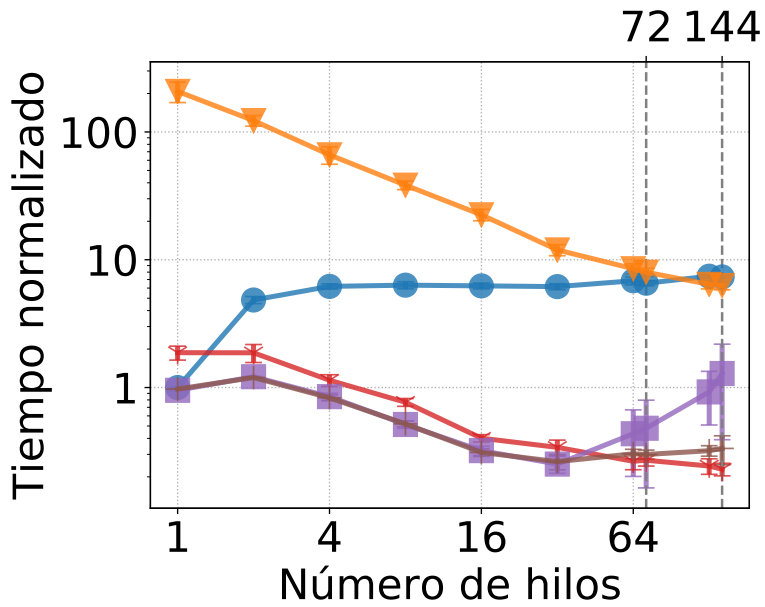
[3] Srishty Patel, Rajshekar Kalayappan, Ishani Mahajan, and Smruti R Sarangi. A hardware implementation of the mcas synchronization primitive. 2017.

Pruebas y resultados

Resultados



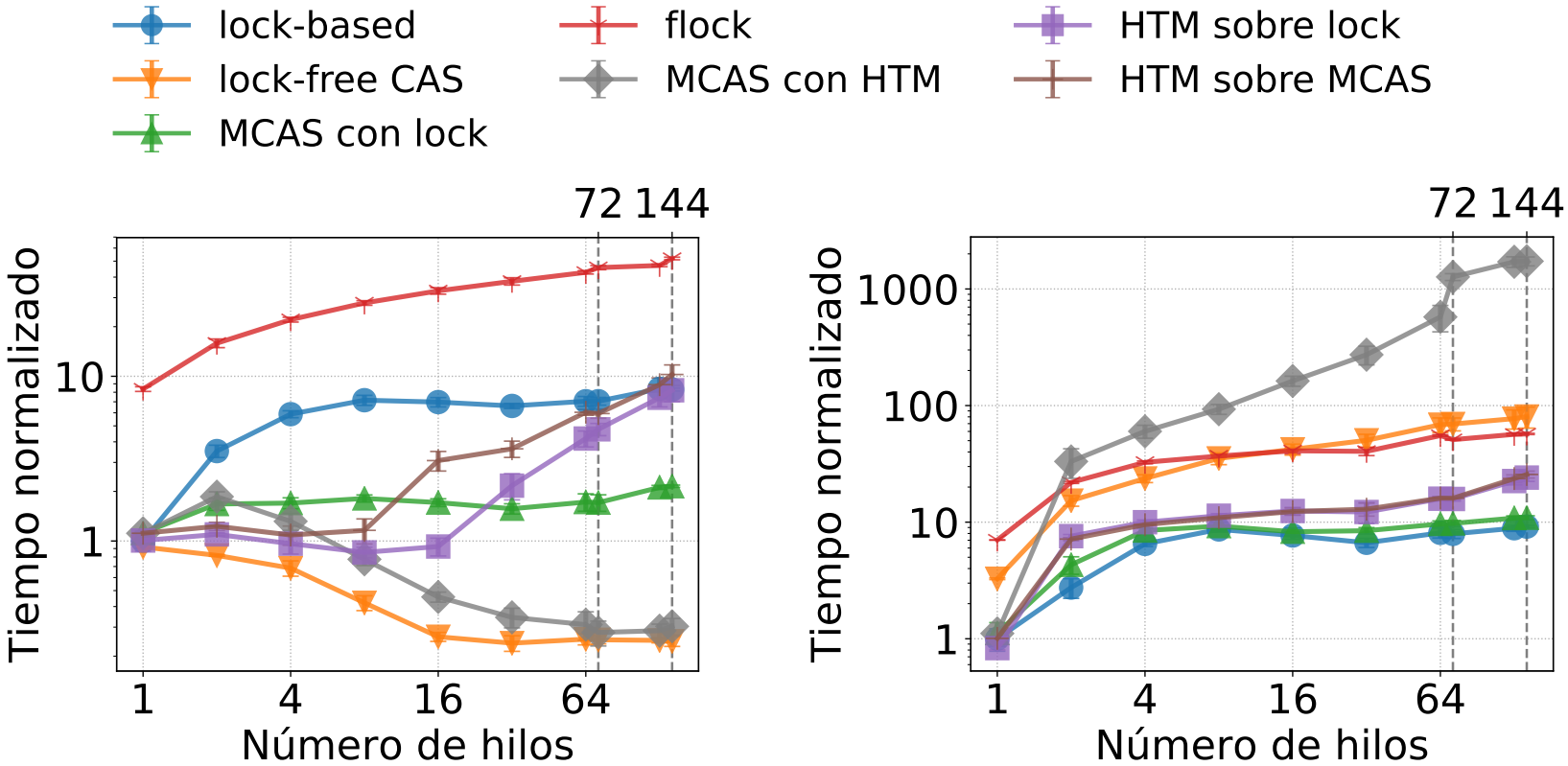
(a) Lista *mixed* sin *backoff*



(b) *Hash map mixed* sin *backoff*

Pruebas y resultados

Resultados



(a) BST *mixed* sin *backoff*

(b) Pila sin *backoff*

Conclusión

- Estudio de varios mecanismos de sincronización
  - `flock`: estructuras *lock-free* con cerrojos de grano fino frecuentemente mejores que *lock-free* con CAS
  - HTM: modelo de programación de grano grueso con mejor rendimiento que *lock-based*
  - Versiones de MCAS solamente preferibles para BST
    - Errores impiden pruebas con lista enlazada y *hash map*

Conclusión

- Estudio de varios mecanismos de sincronización
  - `flock`: estructuras *lock-free* con cerrojos de grano fino frecuentemente mejores que *lock-free* con CAS
  - HTM: modelo de programación de grano grueso con mejor rendimiento que *lock-based*
  - Versiones de MCAS solamente preferibles para BST
    - Errores impiden pruebas con lista enlazada y *hash map*
- En muchas aplicaciones se consiguen mejoras de rendimiento respecto a implementaciones con cerrojos y *lock-free* utilizando alternativas más sencillas

Análisis de técnicas de sincronización en estructuras de datos concurrentes

Álvaro Rubira García

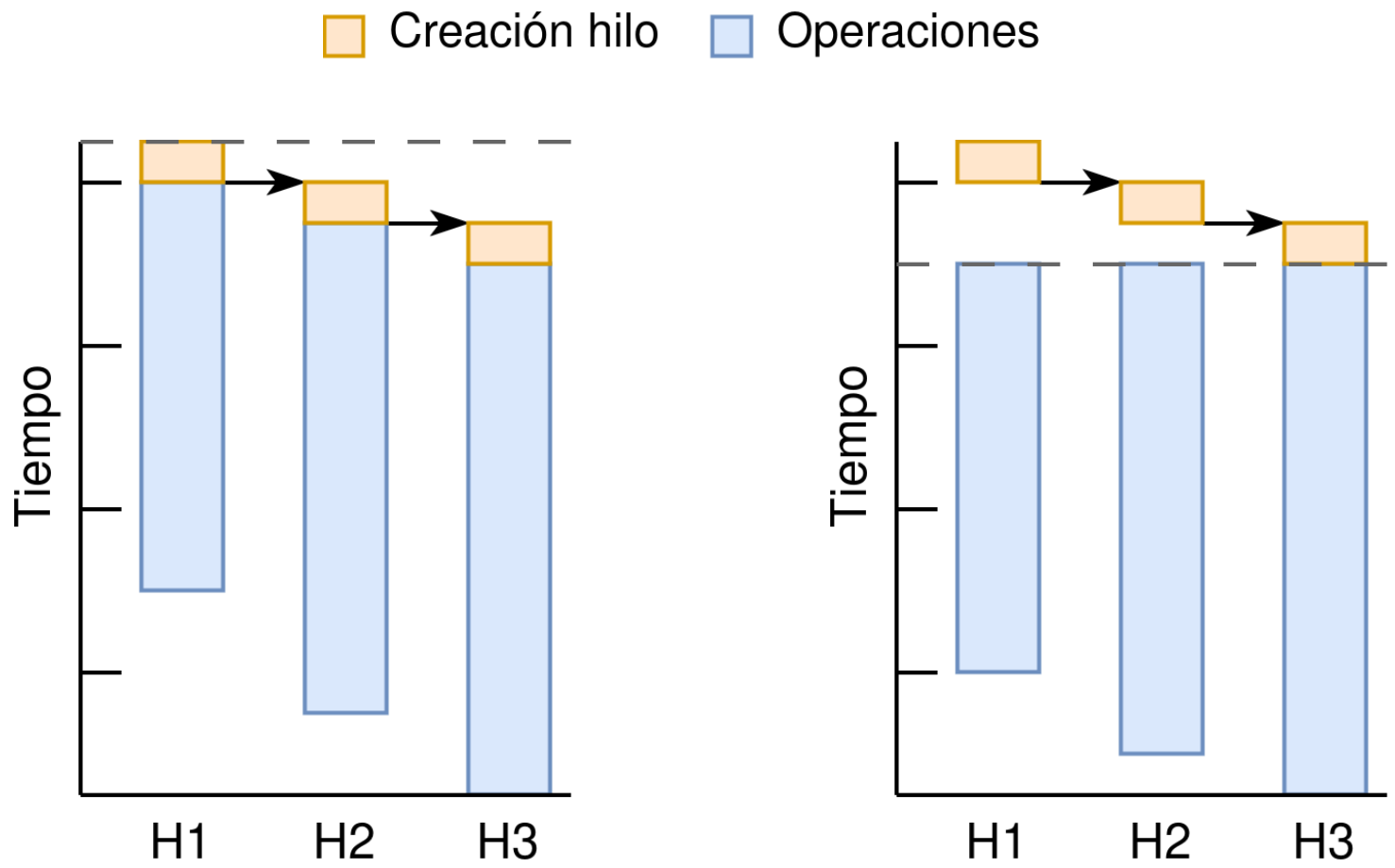
Junio de 2024

UNIVERSIDAD DE MURCIA

https://github.com/AlvaroRGum/data-structure-benchmarks

Apéndice

Barreras



(a) Medición del tiempo original (b) Medición del tiempo con barrera

Apéndice

TM_BEGIN_NO_CONFLICT

```
1   uint64_t dcas(uint64_t* addr0, uint64_t old0, uint64_t new0,
2                 uint64_t* addr1, uint64_t old1, uint64_t new1) {
3     if (!TM_BEGIN_NO_CONFLICT()) {
4       return false;
5     }
6
7     if ((*addr0 == old0) && (*addr1 == old1)) {
8       *addr0 = new0;
9       *addr1 = new1;
10
11      TM_END();
12      return true;
13    }
14
15    TM_END();
16    return false;
17  }
```