Contention and Backoff in Asynchronous Shared Memory

Naama Ben-David and Guy E. Blelloch
Carnegie Mellon University
Contention

- Hardware *sequentializes modifications* to the same memory location.
- Allows *read* operations to execute in *parallel*.
- Accessing a busy location takes much longer than an uncontested one.
Contention: Update Protocol

\textbf{Update}(m):
\begin{verbatim}
    while true:
        current = Read(m)
        new = Modify(current)
        if CAS(m, current, new) then return
\end{verbatim}
Contention: Update Protocol

**Update**<sub>(m)</sub>:

```python
while True:
    current = Read(m)
    new = Modify(current)
    if CAS(m, current, new):
        return
```

**Counter increment:**

Modify(current) = {return current+1}

**Stack push:**

Modify(head) = {return my_entry}
Contestation: Update Protocol

**Update**(m):

```python
while true:
    current = Read(m)
    new = Modify(current)
    if CAS(m, current, new) then return
```

memory location \( m \)
Contestation: Update Protocol

**Update**$(m)$:

```plaintext
while true:
    current = Read$(m)$
    new = Modify$(current)$
    if CAS$(m, current, new)$ then return
```

memory location $m$
Contention: Update Protocol

**Update**\((m)\):  
while true:  
    current = Read\((m)\)  
    new = Modify\((current)\)  
    if CAS\((m, current, new)\) then return
Contestion: Update Protocol

Update(m):
  while true:
    current = Read(m)
    new = Modify(current)
    if CAS(m, current, new) then return

memory location m
Contestion: Update Protocol

\[
\text{Update}(m): \\
\text{while } \text{true}: \\
\text{current} = \text{Read}(m) \\
\text{new} = \text{Modify}(\text{current}) \\
\text{if } \text{CAS}(m, \text{current}, \text{new}) \text{ then return}
\]
Contention: Update Protocol

Update\( (m) \):

\begin{align*}
\text{while} \text{ true:} \\
\text{current} &= \text{Read}(m) \\
\text{new} &= \text{Modify}(\text{current}) \\
\text{if} \ \text{CAS}(m, \text{current}, \text{new}) &\text{ then return}
\end{align*}
Contention: Update Protocol

\[ \text{Update}(m): \]

\begin{align*}
& \textbf{while} \text{ true:} \\
& \quad \text{current} = \text{Read}(m) \\
& \quad \text{new} = \text{Modify}(\text{current}) \\
& \textbf{if} \ \text{CAS}(m, \text{current}, \text{new}) \ \textbf{then return} \\
\end{align*}
Contention: Update Protocol

\textbf{Update}(m):

\textbf{while} true:

\hspace{1cm} current = \text{Read}(m)

\hspace{1cm} new = \text{Modify}(current)

\hspace{1cm} \textbf{if} \ \text{CAS}(m, \ current, \ new) \ \textbf{then return}
Contestion: Update Protocol

\textbf{Update}(m):
\begin{center}
\textbf{while} true:
\end{center}
\begin{center}
current = \text{Read}(m)
\end{center}
\begin{center}
new = \text{Modify}(current)
\end{center}
\begin{center}
\textbf{if CAS}(m, current, new) \textbf{then return}
\end{center}

\text{n}^2 \text{ time, } \text{n}^3 \text{ work!}
Exponential Backoff

- [Anderson ’90] Spin locks
- [Herlihy ’93] Lock-free data structures

**Intuition:**
If there are less processes in the “queue”, waiting times are shorter.

**Works in practice, but no theoretical results!**

```
maxDelay = 1
While true:
  success = tryUpdate
  if success then return
  maxDelay = 2*maxDelay
  d = rand(1, maxDelay)
  wait d steps
```
Contention in the Literature

- [GMR ’96] QRQW PRAM
- [DHW’97] Memory stalls
- [FHS ’05] Lower bounds in terms of memory stalls

Adversarial models don’t capture what happens in practice!
Where does asynchrony in shared memory come from?

- **System**: process swapped out, failed.
- **Hardware**: contention.
Our Approach

Where does asynchrony in shared memory come from?

- **System; process swapped out, failed.**

- **Hardware; contention.**
Our Results

**Model** that separates contention from system delays.

For n processes attempting to update the same location:

- $\Omega(n^3)$ work for **naïve protocol** (no backoff).
- $\Theta(n^2 \log n)$ work w.h.p. for **exponential backoff**.
- **New protocol** that takes $O(n^2)$ work w.h.p.
Our Model

In every \textbf{time step},
- \textbf{Adversary} chooses a \textbf{subset of ready} instructions to activate.
- Every \textbf{active} instruction that \textbf{does not conflict} with anything ahead of it \textbf{gets executed}.

\begin{center}
\begin{tikzpicture}
\node (waiting) at (0,0) {Waiting set};
\node (ready) at (3,0) {Ready set};
\node (active) at (6,0) {Active set};
\draw[->] (waiting) -- (ready);
\draw[->] (ready) -- (active);
\end{tikzpicture}
\end{center}

\textbf{Adversary is weak:}
Cannot see local values when making decisions.
Our Model

In every **time step**, 
- **Adversary** chooses a **subset of ready** instructions to activate.
- Every **active** instruction that **does not conflict** with anything ahead of it **gets executed**.

**Adversary is weak:**
Cannot see local values when making decisions.

Waiting set → Ready set → Active set

I won’t move anyone.

Adversary:

Read-Read
CAS-Read
CAS-CAS

C
R
R
R
Work and Time

We consider discrete time steps.

• Every time step corresponds to one “snapshot” of the queue from our examples.

Our analysis focuses on work.

• Work = $\sum_{t \in T} |Q_t|$

• Measures how much processing power is used.
New Protocol: Adaptive Probability

**Idea:** Gauge contention more accurately by using *reads*.

Allows **backoff** to happen more quickly when contention is high, but **recover quickly** when contention is low.

```
Update(m):
    prob = 1
    current = Read(m)
    while true:
        if flip(prob) == heads:
            if CAS(m, Modify(current), new): Return
        val = Read(m)
        if val != current then prob = prob / 2
        else prob = prob * 2
        current = val
```

**Instead of waiting, I’ll read!**
New Protocol: Adaptive Probability

**Idea:** Gauge contention more accurately by using *reads*.

Allows **backoff** to happen more quickly when contention is high, but **recovery** quickly when contention is low.

**Update**\((m)\):

\[
\text{prob} = 1 \\
\text{current} = \text{Read}(m) \\
\text{while true:} \\
\quad \text{if } \text{flip(prob)} == \text{heads:} \\
\qquad \text{if CAS}(m, \text{Modify(current)}, \text{new}): \text{Return} \\
\qquad \text{val} = \text{Read}(m) \\
\quad \text{if val} \neq \text{current} \text{ then prob} = \text{prob} / 2 \\
\quad \text{else} \quad \text{prob} = \text{prob} * 2 \\
\text{current} = \text{val}
\]
New Protocol: Adaptive Probability

Idea: Gauge contention more accurately by using reads.

Allows backoff to happen more quickly when contention is high, but recover quickly when contention is low.

Update(m):

\[
\begin{align*}
\text{prob} &= 1 \\
\text{current} &= \text{Read}(m) \\
\text{while} & \ \text{true}: \\
& \quad \text{if flip(prob) == heads:} \\
& \quad \quad \text{if CAS(m, Modify(current), new): Return} \\
& \quad \quad \text{val} = \text{Read}(m) \\
& \quad \text{if val != current then} \ \text{prob} = \text{prob} / 2 \\
& \quad \text{else} \quad \text{prob} = \text{prob} * 2 \\
& \quad \text{current} = \text{val}
\end{align*}
\]
New Protocol: Adaptive Probability

**Idea:** Gauge contention more accurately by using reads.

Allows **backoff** to happen more quickly when contention is high, but **recover quickly** when contention is low.

```plaintext
Update(m):
    prob = 1
    current = Read(m)
    while true:
        if flip(prob) == heads:
            if CAS(m, Modify(current), new): Return
        val = Read(m)
        if val != current then prob = prob / 2
        else prob = prob * 2
        current = val
```

Instead of waiting, I’ll read!
New Protocol: Adaptive Probability

Idea: Gauge contention more accurately by using reads.

Allows **backoff** to happen more quickly when contention is high, but **recover quickly** when contention is low.

```
Update(m):
    prob = 1
    current = Read(m)
    while true:
        if flip(prob) == heads:
            if CAS(m, Modify(current), new): Return
            val = Read(m)
        if val != current then prob = prob / 2
        else prob = prob * 2
        current = val
```
New Protocol: Adaptive Probability

Idea: Gauge contention more accurately by using reads.

Allows backoff to happen more quickly when contention is high, but recover quickly when contention is low.

Update(m):

prob = 1

while true:
    if flip(prob) == heads:
        if CAS(m, Modify(current), new): Return
    val = Read(m)
    if val != current then prob = prob / 2
    else prob = prob * 2
    current = val

If I read and see that nothing changed, I'll be more confident next time!
What requires work?

Protocol terminates when **all processes** have done **one successful CAS operation**.

Reads do not conflict; merge into 1 time step

All reads, one CAS

- 2n time steps;
- one successful CAS every other step.
- **n instructions in queue** at all times
  → O(n^2) work

All CASes

- Ω(n^2) time steps;
- one successful CAS every n steps.
- **n instructions in queue** at all times
  → O(n^3) work
What requires work?

Protocol terminates when all processes have done one successful CAS operation.

Our goal:
For all p, prob $\approx 1/n$

Adversary’s goal:
for all p, prob $\approx 1$

All reads, one CAS

2n time steps;
one successful CAS every other step.

$n$ instructions in queue at all times

$\Rightarrow O(n^2)$ work

All CASes

$\Omega(n^2)$ time steps;
one successful CAS every n steps.

$n$ instructions in queue at all times

$\Rightarrow O(n^3)$ work

That’s my plan!
Analyzing Adaptive Probability

• **Lemma**: *Every process makes at most 4 CAS attempts in expectation until it terminates.*

• That is, the adversary can only make us waste 3 CAS attempts in expectation!
Analyzing Adaptive Probability

- Consider a single process P
- The number of CASes P executes depends on its CAS-probability at every step.
- Its CAS-probability for the next step depends on what happens in this step.
Analyzing Adaptive Probability

- Consider a single process $P$
- The number of CASes $P$ executes depends on its CAS-probability at every step.
- Its CAS-probability for the next step depends on what happens in this step.

Adversary chooses which processes to place in front of $P$ in the queue
Analyzing Adaptive Probability

• Consider a single process P

• The number of CASes P executes depends on its CAS-probability at every step.

• Its CAS-probability for the next step depends on what happens in this step.

Adversary chooses which processes to place in front of P in the queue

prob = 1 CAS!

C
1/2
1/n
1/8

0
Analyzing Adaptive Probability

- Consider a single process P
- The number of CASes P executes depends on its CAS-probability at every step.
- Its CAS-probability for the next step depends on what happens in this step.
Analyzing Adaptive Probability

- If $P$ has **CAS-probability** $2^{-i}$, we say $P$ is in **state $i$**.

- We can model $P$'s state transitions:

![Diagram](Image)

- $i-1$ to $i$: $(1-\rho)(1-2^{-i})$
- $i$ to $i+1$: $\rho$
- $i$ to $Done$: $(1-\rho)2^{-i}$
- $i+1$ to $i$: $\rho$
- $i$ to $i-1$: $(1-\rho)(1-2^{-i})$
Analyzing Adaptive Probability

- If P has **CAS-probability** $2^{-i}$, we say P is in **state i**.

- We can model P’s state transitions:

  - $\rho$: Pr[value changed since P's last step]
Analyzing Adaptive Probability

- If $P$ has **CAS-probability** $2^{-i}$, we say $P$ is in **state $i$**.

- We can model $P$'s state transitions:

  - $\rho$: $\Pr[\text{value changed since } P\text{'s last step}]$
  - $2^{-i}$: $\Pr[P\text{ does a CAS}]$
  - $(1-\rho)2^{-i}$
  - $(1-\rho)(1-2^{-i})$
Analyzing Adaptive Probability

- If $P$ has **CAS-probability** $2^{-i}$, we say $P$ is in **state** $i$.

- We can model $P$'s state transitions:
  
  - **$\rho$:** $\Pr[\text{value changed since } P\text{'s last step}]$
  
  - **$2^{-i}$:** $\Pr[P \text{ does a CAS}]

  
  $i-1$ \quad \rightarrow \quad i \quad \rightarrow \quad i+1$

  - $\rho$: $(1-\rho)(1-2^{-i})$
  
  - Go Left
  
  - Go Right
Lemma: $E[CAS_p] \leq 4$.

- Consider process $P$ and the sequence $S$ of states it visits in its execution.

- $S_{\text{Init}}$ = First appearance of each state in $S$.

- $S_{\text{Rest}}$ = The rest of $S$, minus the last step.

- $S_{\text{Last}}$ = The last step in $S$. 
Analyzing Adaptive Probability

Lemma: $E[CAS_P] \leq 4$.

- Consider process $P$ and the sequence $S$ of states it visits in its execution.
- $S_{\text{Init}}$ = First appearance of each state in $S$.
  
  $E[CAS_{\text{Init}}] = \sum 2^{-i} \leq 2$

- $S_{\text{Rest}}$ = The rest of $S$, minus the last step.

- $S_{\text{Last}}$ = The last step in $S$. 
**Analyzing Adaptive Probability**

**Lemma:** \( E[CAS_p] \leq 4. \)

- Consider process \( P \) and the sequence \( S \) of states it visits in its execution.

- \( S_{\text{Init}} = \) First appearance of each state in \( S \).

  \[
  E[CAS_{\text{Init}}] = \sum 2^{-i} \leq 2
  \]

- \( S_{\text{Rest}} = \) The rest of \( S \), minus the last step.

- \( S_{\text{Last}} = \) The last step in \( S \).

  \[
  E[CAS_{\text{Last}}] = CAS_{\text{Last}} = 1
  \]
Analyzing Adaptive Probability

Lemma: $E[CAS_P] \leq 4$.

- Consider process $P$ and the sequence $S$ of states it visits in its execution.
- $S_{\text{Init}} = \text{First appearance of each state in } S$.
  \[ E[CAS_{\text{Init}}] = \sum 2^{-i} \leq 2 \]
- $S_{\text{Rest}} = \text{The rest of } S, \text{ minus the last step}$.
  \[ E[CAS_{\text{Rest}}] = ??? \]
- $S_{\text{Last}} = \text{The last step in } S$.
  \[ E[CAS_{\text{Last}}] = CAS_{\text{Last}} = 1 \]
Analyzing Adaptive Probability

$$E[CAS_{Rest}]$$

$$E[CAS_{Rest}, j] = 2^{-s(j)} + (1 - 2^{-s(j)})E[CAS_{Rest}, j+1]$$

$j = \text{step number}$

$s(j) = \text{state at } j$
Analyzing Adaptive Probability

\[ E[C_{\text{Rest}, j}] = 2^{-s(j)} + (1 - 2^{-s(j)})E[C_{\text{Rest}, j+1}] \]

Accounting for the future: Cost of going right from \( s(j) \)

\( j = \text{step number} \)

\( s(j) = \text{state at } j \)
Analyzing Adaptive Probability

\[ E[CAS_{Rest}, j] = 2^{-s(j)} + (1 - 2^{-s(j)})E[CAS_{Rest}, j+1] \]

Accounting for the future: Cost of going right from \( s(j) \)

\( j = \) step number

\( s(j) = \) state at \( j \)
Analyzing Adaptive Probability

$E[\text{CAS}_{\text{Rest}}]$  

Accounting for the future:
Cost of going right from $s(j)$

\[ E[\text{CAS}_{\text{Rest}, j}] = 2^{-s(j)} + (1 - 2^{-s(j)}) E[\text{CAS}_{\text{Rest}, j+1}] \]

\[ j = \text{step number} \]
\[ s(j) = \text{state at } j \]

Probability $P$ did not terminate in this step

Cost of the rest of the execution
Analyzing Adaptive Probability

$E[CAS_{Rest}]$

$E[CAS_{Rest, j}] = 2^{-s(j)} + (1 - 2^{-s(j)})E[CAS_{Rest, j+1}] \leq 1$

Accounting for the future:
Cost of going right from $s(j)$

$j = \text{step number}$

$s(j) = \text{state at } j$

Cost of the rest of the execution

Probability $P$ did not terminate in this step
Analyzing Adaptive Probability

**Lemma:** \( E[\text{CAS}_P] \leq 4. \)

- Consider process \( P \) and the sequence \( S \) of states it visits in its execution.

- \( S_{\text{Init}} \) = First appearance of each state in \( S \).
  \[
  E[\text{CAS}_{\text{Init}}] = \sum 2^{-i} \leq 2
  \]

- \( S_{\text{Rest}} \) = The rest of \( S \), minus the last step.
  \[
  E[\text{CAS}_{\text{Rest}}] \leq 1
  \]

- \( S_{\text{Last}} \) = The last step in \( S \).
  \[
  E[\text{CAS}_{\text{Last}}] = \text{CAS}_{\text{Last}} = 1
  \]
Analyzing Adaptive Probability

**Lemma:** $E[CAS_P] \leq 4$.

**Another Lemma:** No process takes more than $2n$ steps until terminating.

**Theorem:** The adaptive probability update protocol requires only $O(n^2)$ work in expectation for $n$ processes to update once.
Analyzing Adaptive Probability

Lemma: $\mathbb{E}[\text{CAS}_p] \leq 4$.

Another Lemma: No process takes more than $2n$ steps until terminating.

Theorem: The adaptive probability update protocol requires only $O(n^2)$ work in expectation for $n$ processes to update once.
Analyzing Adaptive Probability

**Lemma:** \( E[CAS_p] \leq 4 \).

**Another Lemma:** No process takes more than \( 2n \) steps until terminating.

**Theorem:** The adaptive probability update protocol requires only \( O(n^2) \) work in expectation for \( n \) processes to update once.
Analyzing Adaptive Probability

Lemma: $E[\text{CAS}_p] \leq 4$.

Another Lemma: No process takes more than $2n$ steps until terminating.

Theorem: The adaptive probability update protocol requires only $O(n^2)$ work in expectation for $n$ processes to update once.

Which means this is w.h.p.
Summary

- **Model** that separates delays due to contention

- Analysis of Update Protocol under our model:
  - **Naive**: $\Omega(n^3)$ work
  - **Exponential Backoff** Analysis: $\Theta(n^2 \log n)$ work in expectation and w.h.p.
  - **New Adaptive Probability** protocol: $O(n^2)$ work in expectation and w.h.p.

- Future Work
  - Experimental evaluation of Adaptive Probabilities
  - Apply analysis and techniques to more general use cases

Thank you!