Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency

Qi Zhao    Jun (Jim) Xu
College of Computing
Georgia Institute of Technology

Zhen Liu
IBM T.J. Watson Research Center

SIGMETRICS 2006/PERFORMANCE 2006
To maintain a large array (say millions) of counters that need to be incremented (by 1) in an arbitrary fashion (i.e., $A[i_1] +=$, $A[i_2] +=$, ...)

- Increments may happen at very high speed (say one increment every 10ns) – has to use high-speed memory (SRAM)

- Values of some counters can be very large

- Fitting everything in an array of “long” (say 64-bit) SRAM counters can be expensive

- Possibly lack of locality in the index sequence (i.e., $i_1$, $i_2$, ...) – forget about caching

Figure 1: Hybrid SRAM/DRAM counter architecture

Counter Increments → small SRAM counters

Overflowing Counters → Counter Management Algorithm

Flush to DRAM → large DRAM counters

N
CMA used in [SIPM:2001]

- Implemented as a priority queue (fullest counter first)
- About 28 bits per counter (when S/D is 10)
- Need pipelined hardware implementation of a heap.

- SRAM counters are tagged when they are at least half full
- Scan from left to right to periodically flush (half-full)\(^+\) SRAM counters, and maintain a small priority queue to preemptively flush the SRAM counters that rapidly become completely full
- Pipelined hierarchical bitmap data structure to find out “Who’s the next (half-full)\(^+\)?” in log\((N)\) time
- 8 SRAM bits per counter for storage and 2 bits per counter for the bitmap control logic, when S/D is 10.
Our scheme

- Our scheme only needs 4 SRAM bits when S/D is 10.
- Flush only when an SRAM counter is “completely full” (e.g., when the SRAM counter value changes from 15 to 16 assuming 4-bit SRAM counters).
- Use a small (say hundreds of entries) SRAM FIFO buffer to hold the indices of counters to be flushed to DRAM.
- Key innovation: a simple randomized algorithm to ensure that counters do not overflow in a burst large enough to overflow the FIFO buffer, with overwhelming probability.
- Our scheme is provably space-optimal (e.g., 3 bits will never work when S/D is 10).
The randomized algorithm

• Set the initial values of the SRAM counters to independent random variables uniformly distributed in \( \{0, 1, 2, ..., 15\} \) (i.e., \( A[i] := \text{uniform}\{0, 1, 2, ..., 15\} \)).

• Set the initial value of the corresponding DRAM counter to the negative of the initial SRAM counter value (i.e., \( B[i] := -A[i] \)).

• Adversaries know our randomization scheme, but not the initial values of the SRAM counters

• We prove rigorously that a small FIFO queue can ensure that the queue overflows with very small probability
A numeric example

- One million 4-bit SRAM counters (512 KB) and 64-bit DRAM counters with SRAM/DRAM speed difference of 12
- 300 slots ($\approx$ 1 KB) in the FIFO queue for storing indices to be flushed
- After $10^{12}$ counter increments in an arbitrary fashion
- The probability of overflowing from the FIFO queue: less than $10^{-14}$ in the worst case
Timing diagram of the hardware operation

Figure 2: Hybrid SRAM/DRAM counter architecture

- : read SRAM counter value
- : increment SRAM counter value (+1 or reset to 0 if it overflows)
- : append the index of the counter to the queue.
Tail bound analysis – Intuition

- The average departure rate of the FIFO queue is the speed of DRAM (e.g., 1 departure every 12 cycles or with the rate $1/12$ when $S/D$ is 12)
- The average arrival rate to the FIFO queue is approximately $1/16$, as it takes 16 increments for a counter to become full – and hopefully the randomization makes the arrival process very smooth!
- Actually, our experimental result is very close to that of the Geom/D/1 queue
- However, we are NOT able to prove that our queueing process is stochastically comparable to (or bounded by) that of a Geom/D/1 queue – only able to prove much weaker tail bounds
Tail bound analysis (1st step)

- Let $D$ be the event that the FIFO queue overflows after $n$ increments.
- Let $D_{s,t}$ be the event that the number of arrivals during the time interval $[s, t]$ is larger than the maximum possible number of departures from the FIFO queue (even if serving continuously), by more than the queue size $K$.
- Lemma 1: $D \subseteq \bigcup_{0 \leq s \leq t \leq n} D_{s,t}$ (proved using standard busy period arguments)
- Therefore

$$
\Pr[D] \leq \Pr[\bigcup_{0 \leq s \leq t \leq n} D_{s,t}] \leq \sum_{0 \leq s \leq t \leq n} \Pr[D_{s,t}]
$$
Bounding $\Pr[D_{s,t}]$ using Chernoff bound

- Let $c_j, j = 1, 2, \ldots, N$ be the number of increments to counter $j$ during time period $[s, t]$ – note our bound will be independent of these $c_j$ values (note $\sum_{j=1}^{N} = n$)
- Let $b_j$ be the number of “flush to DRAM” requests generated by the counter $j$ during the time interval $[s, t]$
- It can be shown that $b_j - E[b_j], j = 1, 2, \ldots, N$, are independent Bernoulli RV’s:

$$b_j = \begin{cases} \left\lfloor \frac{c_j}{2^l} \right\rfloor & \text{with probability } 1 - \{2^{-l}c_j\}, \\ \left\lfloor \frac{c_j}{2^l} \right\rfloor + 1 & \text{with probability } \{2^{-l}c_j\}. \end{cases} \quad (1)$$
• Lemma 3, Let $X_1, X_2, \ldots, X_m$ be mutually independent random
variable such that, for $1 \leq j \leq m$, $\Pr[X_j = 1 - p_j] = p_j$
and $\Pr[X_j = -p_j] = 1 - p_j$, where $0 < p_j < 1$. Then, for
$X = \sum_{j=1}^{m} X_j$ and $a > 0$,
$$\Pr[X > a] < e^{-2a^2/m}$$

• Applying to the sum of $b_j's$, we obtain Theorem 2:
For any $s < t$, let $\tau = t - s$.
$$\Pr[D_{s,t}] \equiv \Pr[b(s, t) - \mu \tau > K] < e^{-2(K+\mu \tau-2^{-l}\tau)^2/\min\{\tau,N\}}$$
(2)
Using 2nd Moment Information to Obtain a New Bound of $\Pr[D_{s,t}]$

$$VAR[b(s, t)] \leq \begin{cases} \frac{N}{4} \left(\frac{2^l - t-s}{N}(t-s)\right) & t - s \geq 2^{l-1}N, \\ \frac{(2^l - 1)(t-s)}{2^{2l}} & N \leq t - s < 2^{l-1}N, \\ 0 & 0 < t - s < N. \end{cases}$$

There is implicitly a quasi minimax analysis in it – imaging that the adversary has control over the increment index sequence
A New Tail Bound Theorem

• Given any $\theta > 0$ and $\epsilon > 0$, the following holds: Let $W_j, 1 \leq j \leq m, m$ arbitrary, be independent random variables with $\text{EXP}[W_j] = 0$, $|W_j| \leq \theta$ and $\text{VAR}[W_j] = \sigma_j^2$. Let $W = \sum_{j=1}^{m} W_j$ and $\sigma^2 = \sum_{i=1}^{m} \sigma_j^2$ so that $\text{VAR}[W] = \sigma^2$. Let $\delta = \ln(1 + \epsilon)/\theta$. Then for $0 < a \leq \delta \sigma$,

$$\Pr[W > a\sigma] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$$

• Mapping to our problem, it becomes

$$\text{maximize} \quad \frac{a^2}{2}(1 - \frac{\epsilon}{3})$$

$$\text{subject to} \quad 0 < a \leq \delta \sigma$$

$$e^\delta - 1 \leq \epsilon < 3$$

$$a\sigma \leq K + \mu \tau - 2^{-l} \tau$$
The Hybrid Tail Bound

- Recall that $\Pr[D] \leq \sum_{0 \leq s \leq t \leq n} \Pr[D_{s,t}]$
- We derived the first bound $\Pr[D_{s,t}] \leq \Omega_1(s, t)$ using Chernoff bound
- We derived the second bound $\Pr[D_{s,t}] \leq \Omega_2(s, t)$ using our new tail bound theorem
- The first bound is better for most of the $s, t$ values, BUT the second bound can be much better for some critical $s, t$ values
- We refer to $\Pr[D] \leq \sum_{0 \leq s \leq t \leq n} \min\{\Omega_1(s, t), \Omega_2(s, t)\}$ as the hybrid bound
**Numerical Examples**

Given $N = 10^6$, $n = 10^{12}$, $\mu = 1/30$ and $l = 5$ bits,

<table>
<thead>
<tr>
<th>$K$</th>
<th>First</th>
<th>Second</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>trivial ($\geq 1$)</td>
<td>trivial ($\geq 1$)</td>
<td>$1.1 \times 10^{-11}$</td>
</tr>
<tr>
<td>3033</td>
<td>$1.4 \times 10^{-6}$</td>
<td>trivial ($\geq 1$)</td>
<td>$8.7 \times 10^{-142}$</td>
</tr>
</tbody>
</table>
Cost-benefit Comparison

Given \( l = 64 \text{ bits} \), \( \mu = 1/30 \), and \( K = 500 \text{ slots} \)

<table>
<thead>
<tr>
<th></th>
<th>Naive</th>
<th>( LCF )</th>
<th>( LR(b) )</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter memory</td>
<td>64Mb SRAM</td>
<td>9Mb SRAM</td>
<td>9Mb SRAM</td>
<td>5Mb SRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64Mb DRAM</td>
<td>64Mb DRAM</td>
<td>65Mb DRAM</td>
</tr>
<tr>
<td>Control memory</td>
<td>None</td>
<td>20Mb SRAM</td>
<td>2Mb SRAM</td>
<td>10Kb SRAM</td>
</tr>
<tr>
<td>Control logic</td>
<td>None</td>
<td>Hardware heap</td>
<td>Aggregated bitmap</td>
<td>FIFO queue</td>
</tr>
<tr>
<td>Implementation Complexity</td>
<td>Very low</td>
<td>High</td>
<td>Low</td>
<td>Very Low</td>
</tr>
</tbody>
</table>
Simulation Using Real-world Internet Traffic

- Given $N = 1,000,000$, $n = 10^{12}$, $\mu = 1/30$ and $l = 5$ bits,

<table>
<thead>
<tr>
<th>Trace</th>
<th>SRAM counter size (in bits)</th>
<th>$\mu$</th>
<th>Queue Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max</td>
</tr>
<tr>
<td>USC</td>
<td>4</td>
<td>1/12</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1/30</td>
<td>61</td>
</tr>
<tr>
<td>UNC</td>
<td>4</td>
<td>1/12</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1/30</td>
<td>72</td>
</tr>
</tbody>
</table>

- Computing the hybrid bound, we need 228 slots for the bound to be nontrivial.

- The experimental result is in fact very close to that of Geom/D/1 queue (average is 1.6).

- The experimental result is much better than the bound because (1) The input is not adversarial, and (2) The union bound $Pr[\bigcup_{0 \leq s \leq t \leq n} D_{s,t}] \sum_{0 \leq s \leq t \leq n} Pr[D_{s,t}]$ is very lossy.
Conclusion

• A simple and efficient counter management algorithm for hybrid SRAM/DRAM counter architecture

• Statistical guarantee for queue overflow probability

• A new tail bound theorem for the sum of independent random variables that can take advantage of both their independence and their overall low variance
Future Work

- Further improve the theoretical bound by possibly ditching the union bound
- Allow for both increments and decrements – this algorithm won’t work since an adversary can create thrashing around 0.
- Apply the counter array work to other network applications (e.g., for implementing millions of token buckets).
Thank You!

ANY QUESTIONS?
Concern over heavy traffic through system bus

- Concern: shorter SRAM counter size means that much larger flushing traffic through the system bus, when the SRAM array is on the L1 cache of a network processor.
- "Victim of our own success": previous schemes are constrained by the lower efficiencies of their CMA algorithms, not by the concern that there will be too much bus traffic.
- We intend our scheme/algorithm to be generic and we do not want to bind it to any particular architecture choice just like in previous works.
- The heavy traffic over the bus may not be an issue in many scenarios: (a) a computer architecture can have a dedicated bus between CPU and memory (b) the system is built for network monitoring only (e.g., Sprint’s CMON)