HogWild on Steroids: SGD via Migrating Threads

Peter M. Kogge
McCourtney Prof. of CSE
Univ. of Notre Dame
IBM Fellow (retired)

Observations

- Modern systems becoming heterogeneous
  - Accelerators for high compute intensive apps
  - Need complex "glue" code for multi-node scaling
- But many apps no longer good fits
  - Computation often memory B/W limited
  - Intra-node parallelism often coherency limited
  - Inter-node parallelism no longer regular or predictable
- Result: poor efficiency, poor scalability

Project Funding

- NSF SPX project joint with Vivek Sarkar
  - "Scalable Heterogeneous Migrating Threads for Post-Moore Computing"
- Goal: can we fix the heterogeneous problem of combining different ISAs/core types with "migrating threads?
- Emu machine in CRNCH an excellent resource

This Study

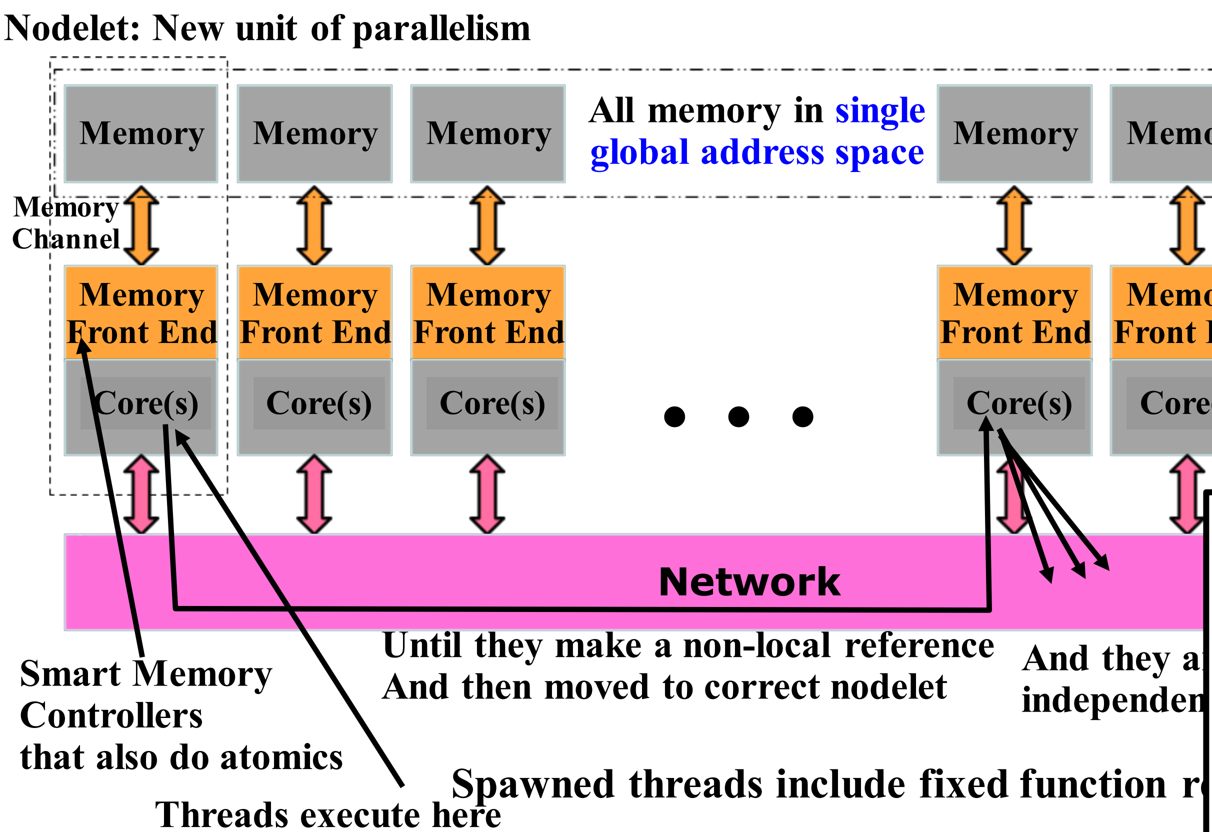
- Can a modern **Machine Learning** problem
- Benefit from architectures with **Migrating Threads**
- And what is likely scalability

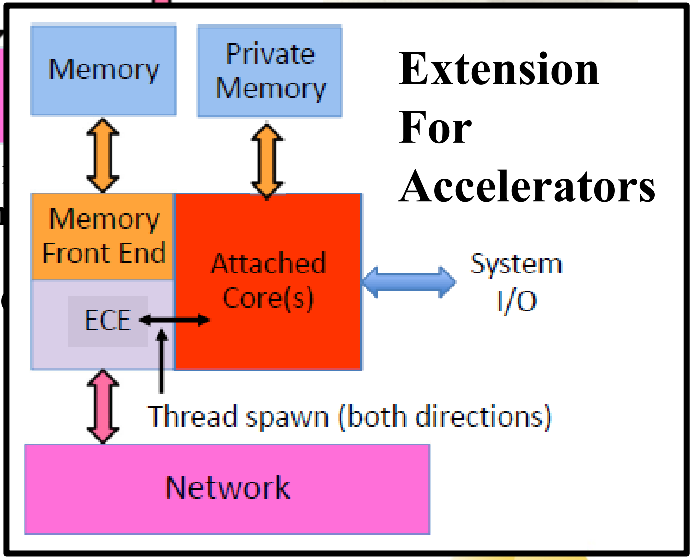
Today's talk: Very Preliminary Paper Analysis

A Funny Thing Happened…

- For this problem
  - Sparsity can dominate computational complexity
  - Inter-node communication is still the problem
  - Heterogeneity probably not efficient match
- Efficient Multi/Migrating threads turn out to solve ***both*** the intra and inter node problems

Thesis: Use “Migrating Threads” as Glue





Really good for large sparse, irregular data sets & memory bound apps

Machine Learning

- Given set E of "training samples"
- Find some model vector x*
- That minimizes $$\hat{f}(x) = \sum_{e \in E} f_e(x)$$
  - Where $f_e(x)$ is defined by e'th sample

Stochastic Gradient Descent

Iteratively improve an estimate of x*

- Randomly choose a new sample
- Update estimate: $x^{(k+1)} = x^{(k)} - \alpha \nabla \hat{f}(x^{(k)})$
- Where $\nabla \hat{f}(x^{(k)})$ is multidimensional gradient

Naïve Parallel Algorithm

- Obvious Algorithm
  - Break sample set into “mini-batches”
  - Process each independently
  - Combine updates
- If updates done asynchronously, results bad
- If updates done in single critical section, poor scalability

Hogwild Algorithms

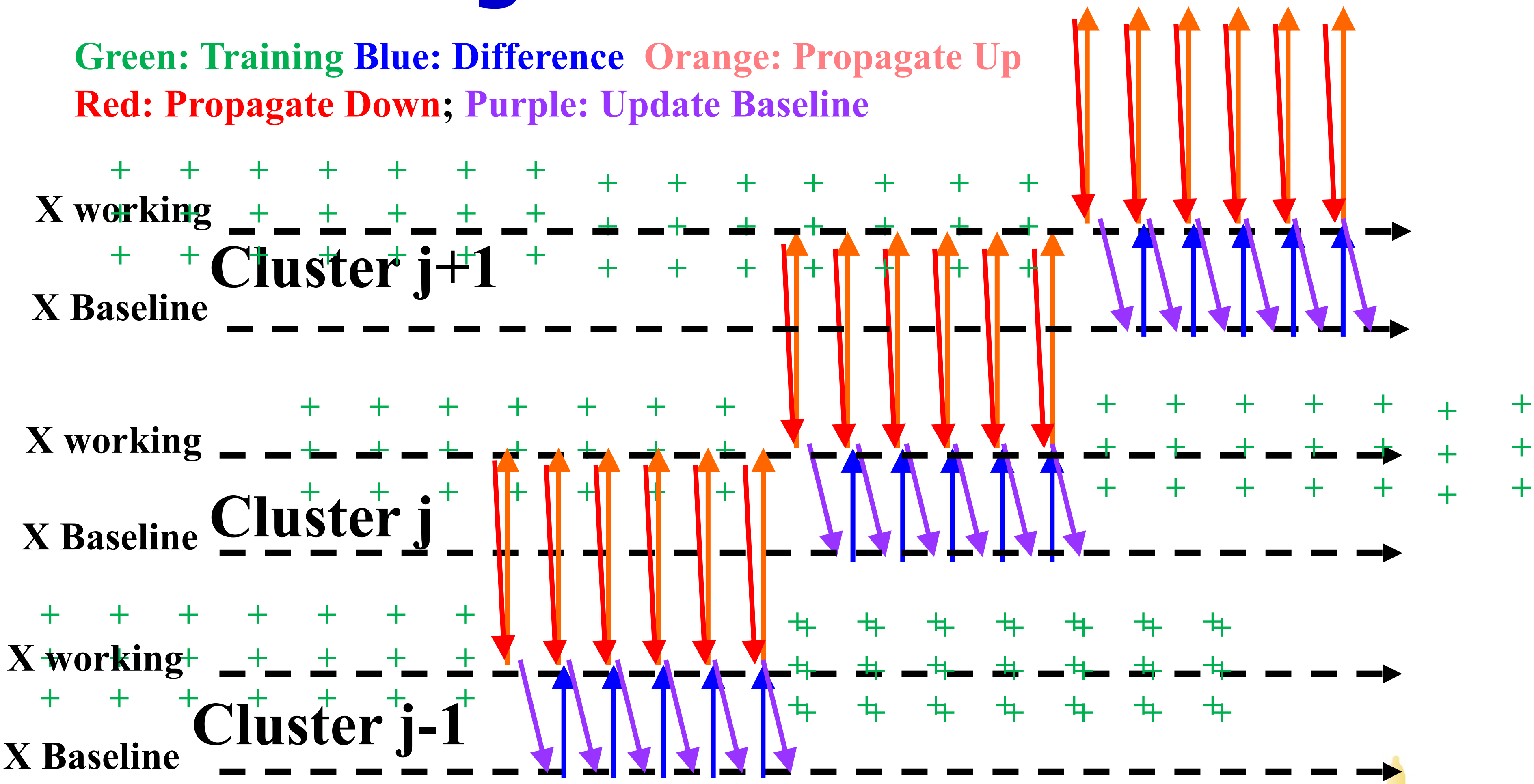
- Observation: if samples *sparse*, updates **can** be done *incrementally*, without locking
- **HogWild!**: use shared memory multi-threaded platform
  - One thread/mini-batch
  - Atomic updates to elements of shared model vector
  - Some scalability but suffers from coherency traffic
- **BuckWild**: use short precision
- **HogWild++**: Keep multiple local model vectors & perform cyclic updates
  - Much better scalability

Generic HogWild++

repeat
  for all threads in cluster j do
    Pick a random training sample e
    $atomic(\hat{x}_j^{(\tau+1)} = \hat{x}_j^{(\tau)} - \eta_0 \gamma^t \nabla f_e(\hat{x}^{(\tau)}))$
    $atomic(\tau = \tau + 1)$
    if Token is received **then**
      $\Delta x_j^{(k)} = \hat{x}_j^{(\tau)} - x_j^{(k)}$ {Compute local changes}
      $atomic(x_j^{(k+1)} = \lambda \hat{x}_{j+1}^{(\tau')} + (1-\lambda) x_j^{(k)} + \beta \gamma^t \Delta x_j^{(k)})$
      {Now update neighbor's $\hat{x}$}
        $atomic(\hat{x}_{j+1}^{(\tau')} = \hat{x}_{j+1}^{(\tau')} + \beta \gamma^t \Delta x_j^{(k)})$
      $\hat{x}_j^{(0)} = x_j^{(k+1)}$ {Update local model}
      $k = k + 1; \tau = 0$
      Pass token after at least τ_0 iterations
    end if
  end for
until $\tau \geq m$

- Threads divided into clusters
- Each cluster has own model vector
- Local updates incrementally
- Token passed cyclically around
- Cluster j exchanges updates with cluster j+1

HogWild++ Traffic



Support Vector Machine (SVM)

- Each sample (z_e, y_e)
  - a vector z_e of F features
  - Member of one of two classes (y_e=+1 or -1)
- Sign of $x^T z_e$ determines class
- $f_e(x) = max(0,\ 1 - y_e x^T z_e)$ $\hat{f}(x) = (\sum_e max(0,\ 1 - y_e \boxed{x^T z_e})) + \lambda ||x||_2^2$ O(F)
- $\nabla \hat{f}(x)[i] = (\hat{f}(x + hb_i) - \hat{f}(x))/h$
- $\nabla f_e(x)[i] = (max(0, g_e - y_e h z_e[i]) - max(0, g_e))/h$
- If z_e is sparse: $\hat{f}(x) = \sum_e (max(0, 1 - y_e \boxed{\sum_{i \in Z_e} x[i] z_e[i]}) + \lambda \sum_{i=1}^{n} x[i]^2)$
  - ***Huge reduction*** if F' << F

O(F') where F'=# non-zeros

Reported Results

Data set	S:Training Samples	Sparsity	Per Sample F: Features	F':Non-Zeros	Max Speedup	Best Configuration Cores	Cores/Cluster	Clusters	τ_0	$log_{64}(S)$
news20	16,000	0.0336%	1,355,191	455	9.5	49	4	10	16	3.4
covtype	464,810	22.12%	54	12	30	40	1	40	16	1
webspam	280,000	33.52%	254	85	40	40	1	40	16	1.4
rcv1	677,399	0.155%	47,236	73	38	40	1	40	16	2.6
epsilon	400,000	100%	2,000	2,000					16	1.9

TABLE I

- **Webspam**: small feature set F, good speedup
- **News20**: large feature set, poor speedup

Issues to Consider

- Sparsity of samples: reduce training costs
  - But we know a priori which are non zero
- Sparsity of changes to handle during token passing updates
  - Not known a priori at either j or j+1
- Reducing inter-cluster traffic during token update
  - Avoid multiple large vector transfers
- Performing individual updates atomically

Single Cluster SVM- Training Only

Algorithm 2 Single Cluster SVM Epoch Computation
SY: vector of pairs (index into IZ, y_e) for sample e
IZ: vector of pairs (feature index,value) for samples
X: vector of current model values
$\lambda, \beta, \eta\gamma^t, h$: coefficients for update

for $\tau = 1$ to $|E|$ by 1 **do**
  {Multiple threads can execute asynvhronously}
  $e = 2 * random\_int(1, |E|)$ {Select sample}
  $iz2 = iz = SY[e]$ { Index to 1st non-zero}
  $y_e = SY[e+1]$; $iend = SY[e+2]$; $g = 0$
  {Compute $y_e \sum_{j \in Z_e} x[j] z_e[j]$}
  repeat
    $i = IZ[iz]$; $zei = IZ[iz+1]$
    $g = g + X[i] * zei$
  until $(iz++2) \geq iend$
  $g = 1 - y_e g$; $gmax1 = max(0, g)$;$yh = y_e * h$
  {Compute each $\nabla f_s(x)[i]$}
  repeat
    $i = IZ[iz2]$; $zei = IZ[iz2+1]$
    $dx2i = 2h * zei$
    $gmax2 = max(0, g - yh * zei - dx2i)$
    $gi = \eta_0 \gamma^t (gmax2 - gmax1)/h$
    $atomic(X[i] += gi)$
  until $(iz2++2) \geq iend$
end for

O(F')

O(F')

Features/Sample	No Cache 10	No Cache 100	No Cache 1000	64B Cache Lines 10	64B Cache Lines 100	64B Cache Lines 1000
Transfers/Sample	89	809	8009	49	409	4009
Flops/Transfer	1.04	1.12	1.12	2.02	2.22	2.35
Flops/Byte Accessed	0.13	0.14	0.14	0.03	0.03	0.04

- Modern architectures have flops/byte of 4-8

Single Token Update – Full Vector

Algorithm 3 Single Token Update - Full Vector
F: Number of features (zero or non-zero)
$\hat{x}_{j+1}^{(\tau')}[i]$ is on the next cluster

algorithm.3

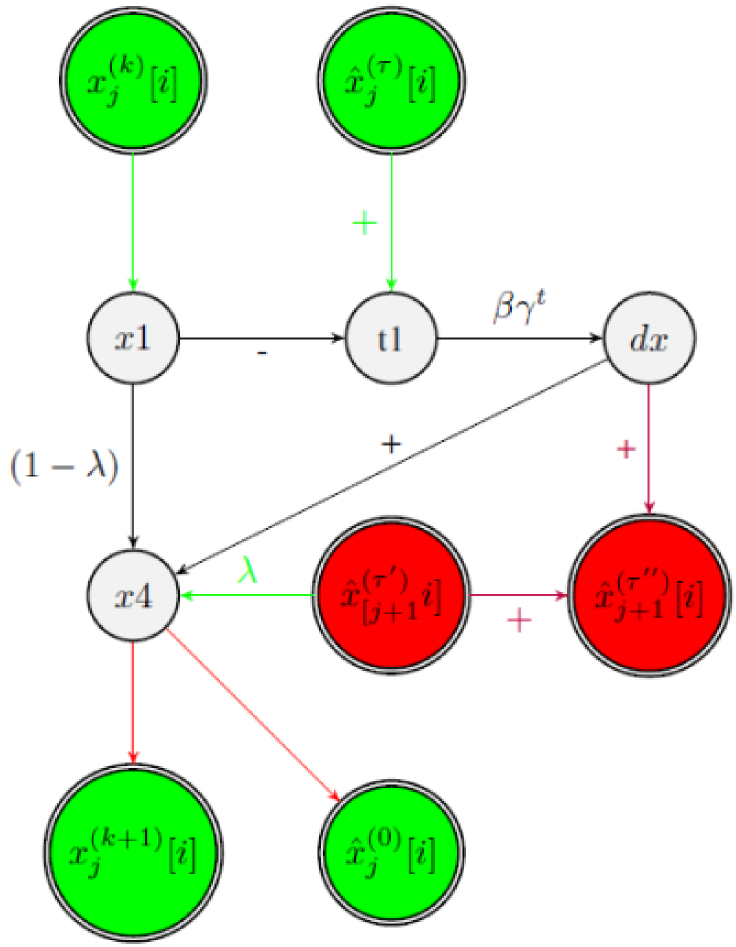
```
{Iterate over each feature in model vector}
for i = 1 to F by 1 do
    {Line 9: △x_j^(k) = x̂_j^(τ) − x_j^(k)}
      x1 = x_j^(k)[i] {Local read}
      x2 = x̂_j^(τ)[i] {Local read}
      dx = βγ^t(x2 − x1)
    {Line 14: x_j^(k+1) = λx̂_{j+1}^(τ') + (1 − λ)x_j^(k) + βγ^t △x_j^(k)}
      x3 = x̂_{j+1}^(τ')[i] {Read from cluster j + 1}
      x4 = λx3 + (1 − λ)x1 + dx
      x_j^(k+1)[i] = x4 {Local store}
    {Line 16: x̂_{j+1}^(τ') = x̂_{j+1}^(τ') + βγ^t △x_j^(k)}
      atomic(x̂_{j+1}^(τ')[i]+ = x4) {Remote update}
    {Line 17: x̂_j^(0) = x_j^(k+1)}
      x̂_j^(0)[i] = x4 {Local store}
    {The increment of i should be an AMO.}
end for
```

All these read full vector of F components, even if most are 0

Options for Tracking Dynamic Changes

- Keep list of changes
  - Add each time local model vector is updated
  - Cost of removing duplicates likely large
- Keep bit vector of changes
  - But still requires O(F) reads to scan
- Keep tree of bit vectors
  - Four deep of 64b words covers up to 16M samples
  - Requires up to 4 operations to record change
  - Must be done atomically if multi-threaded
- Also remember changes in cluster j+1 likely different from changes in cluster j

Complexity of Inter-Cluster Updates



Double circle: memory location
Green circle: local memory; Red circle: remote memory
Green line: read; Red line: write; Purple line: AMO

- Complex flow of data – in both directions
- Migrating Threads to rescue:
  - Separate cluster j and cluster j+1 processing
  - Spawn thread from cluster j to do processing at cluster j+1
  - Have cluster j+1 issue remote atomic adds for updates back to cluster j

Migrating Thread Pseudocode

Algorithm 4 Notional Migrating Thread Training Loop

```
repeat
    {Process samples using Algorithm 2}
    {Multiple threads can execute asynchronously}
    e = 2 * random_int(1, |E|) {Select sample}
    ix2 = ix = SY[e];
    y_e = SY[e+1]; iend = SY[e+2]; g = 0
    repeat
        g = g + X[IZ[ix]] * IZ[ix+1]
    until (ix++2) ≥ iend
    gmax1 = max(0, g = 1 − y_e g); yh = y_e * h
    repeat
        i = IZ[ix2]; zei = IZ[ix2+1]; dx2i = 2h * zei
        gmax2 = max(0, g − yh * zei − dx2i)
        atomic(X[ix] += η_0 γ^t (gmax2 − gmax1)/h)
        set_change_bit_j(i)
    until (ix2++2) ≥ iend
    {Test for token arrival}
    if atomic(τ += 1) > τ_0 then
        if CAS(token_j = 1, −1) then
            Break {1st thread to see it}
        else if token < 0 then
            Quit execution {Later threads}
        end if
    end if
until τ > m_j
{Token has arrived - drop to Token Processing}
```

Algorithm 5 Notional Migrating Thread Token Update

```
{Following code is executed on cluster j}
last_i = 1
for i = find_next_change_bit_j() do
    reset_change_bit_j(i)
    x1 = x_j[i] {Local read}
    dx = βγ^t (x̂_j[i] − x1)
    x4 = (1 − λ)x1 + dx {Partial update}
    x_j[i] = x4 {Local store of partial}
    x̂_j[i] = x4 {Local store}
    spawn remote_update(i, dx, last_i)
    last_i = i
end for
Restart sample training loop

{Following code is executed on cluster j+1}
procedure remote_update(i, dx, last_i)
atomic((x3 = x_{j+1}[i]) += dx) {Update local x}
remote_atomic(x_j[i] += λx3) {Complete x_j[i] update}
remote_atomic(x̂_j[i] += λx3) {Complete x̂_j[i] update}
set_change_bit_{j+1}(i)
for n = i − 1 down to last_i + 1 do
    if change_bit_{j+1}[n] == 1 then
        x3 = x̂_{j+1}^{(τ')} {Now a local read}
        remote_atomic(x̂_j[n] += x3)
        set_remote_change_bit_j(n)
    end if
end for
quit
```

Migrating Thread

Float atomic

Integer atomic

Key Model Parameters

- **F**: number of features per sample
- **F'**: average non-zero features per sample
- **F'':** average # of changes in x in one cluster between tokens
- **F''':** average # of changes in cluster j+1 that were not changes in cluster j

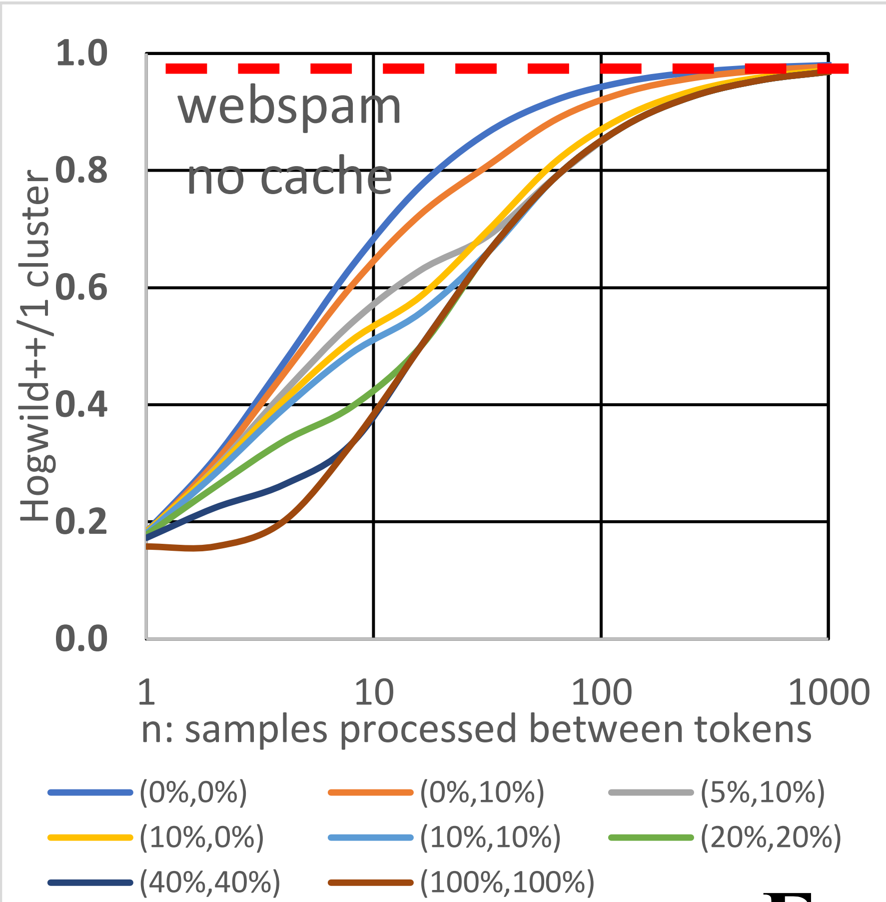
Traffic Counts

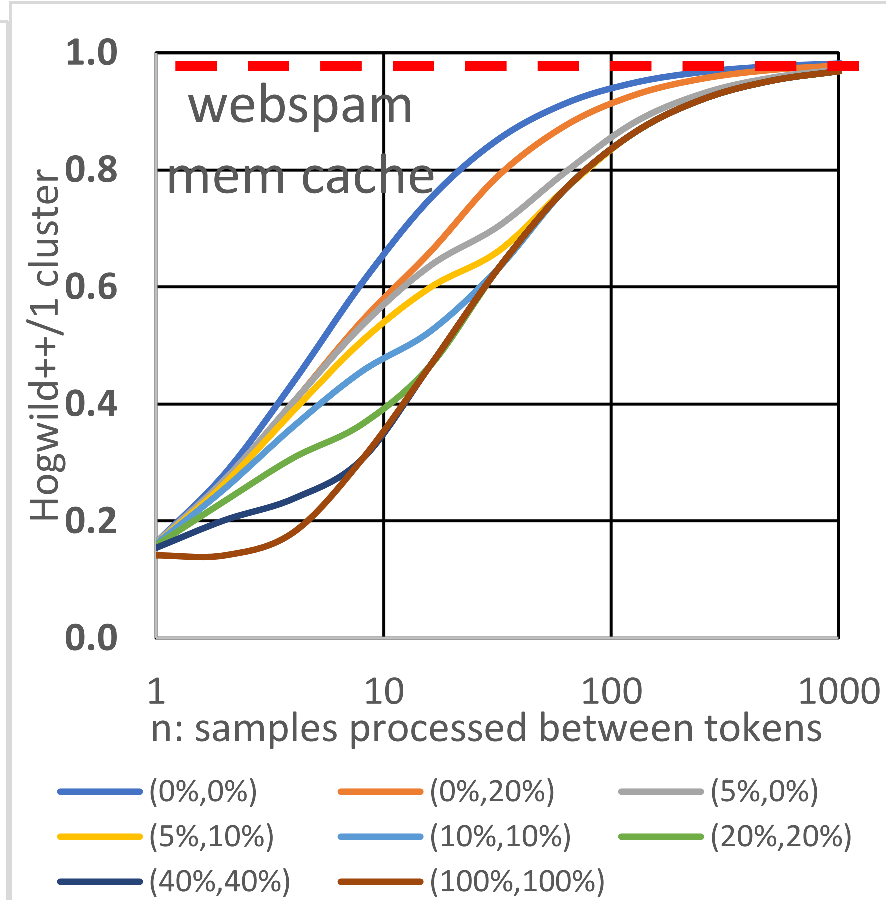
	Training	Local Update	Remote Update	Total Transfers: No Cache	Total Transfers: Cache
Local Random Reads	$n + 3nF'$	$2F''(n)$	$2F'''(n)$	$n + 3nF' + 2F''(n) + 2F'''(n)$	same
Local Sequential Reads	$n + 2nF'$			$n + 2nF'$	0
Local Random Writes		$2F''(n)$		$2F''(n)$	$4F''(n)$
Local Integer Atomics	$2n$	$F''(n)$		$6n + 3F''(n)$	$4n + 2F''(n)$
Local Float Atomics	nF'		$F''(n)$	$3nF' + 3F''(n) + 3F'''(n)$	$3nF' + 2F''(n) + 2F'''(n)$
Remote Float Atomics			$2F''(n) + F'''(n)$	$6F''(n) + 3F'''(n)$	$2nF' + 4F''(n) + 2F'''(n)$
random_int()	n			$6n$	$6n$
Find_next_change_bit()		$F''(n)$		$log_{64}(F)F''(n)$	$log_{64}(F)F''(n)$
Local *(re)set_change_bits*	n	$F''(n)$	$F''(n)$	$3log_{64}(F)(n + 2F''(n))$	$2log_{64}(F)(2n + F''(n))$
Remote *(re)set_change_bits*			$F'''(n)$	$3log_{64}(F)F'''(n)$	$2log_{64}(F)F'''(n)$
Full Thread Spawns		$F''(n)$			
Migrations from Cluster		$F''(n)$			
Float Adds	$n + 4nF'$	$2F''(n)$	$3F''(n) + F'''(n)$	$n + 4nF' + 5F''(n) + F'''(n)$	
Float Multiplies	$2n + 4nF'$	$2F''(n)$	$F''(n)$	$2n + 4nF' + 3F''(n)$	

n: number of samples processed between tokens. F': ave. non-zeros per sample.
F'' $F''(n)$: Ave. number of changes in cluster j during the processing of n samples.
F''' $F'''(n)$: Ave. number of changes in cluster $j+1$ during the processing of n samples by cluster j that are not changes by j.

- Transfer: movement of data to/from memory
- **No cache**: current Emu machine
- **Cache**: cache added to memory controller
- No cache AMOs: 3 transfers
- Cache AMOs: 2 transfers

Webspam Dataset





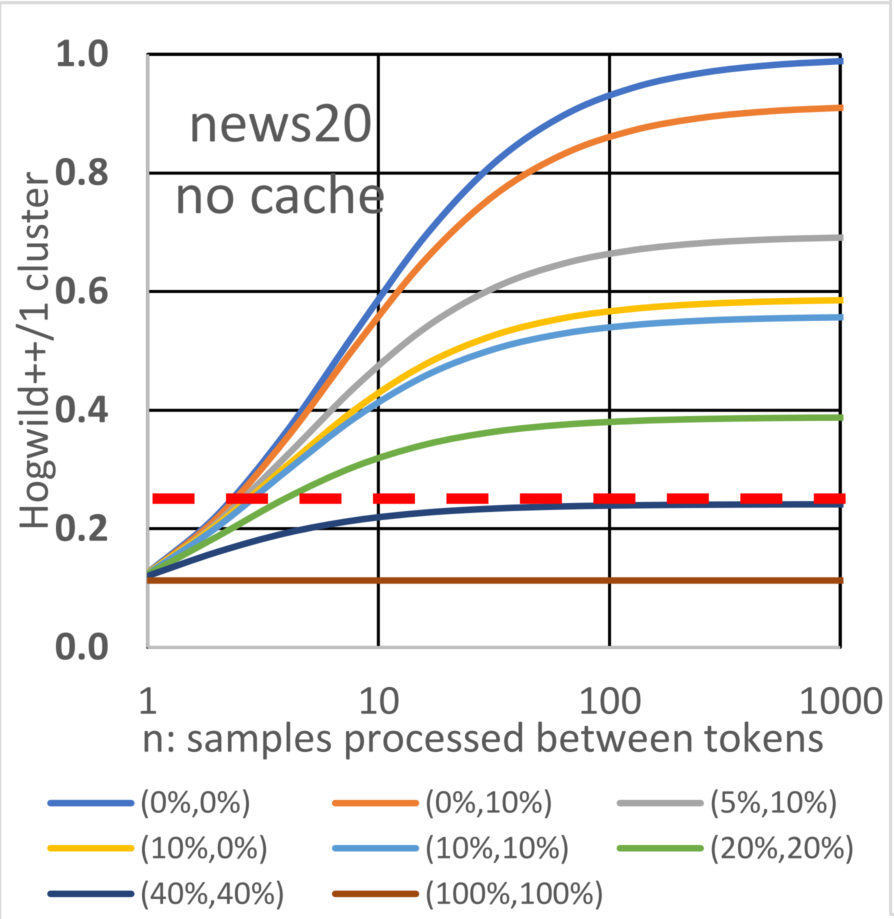
No Cache

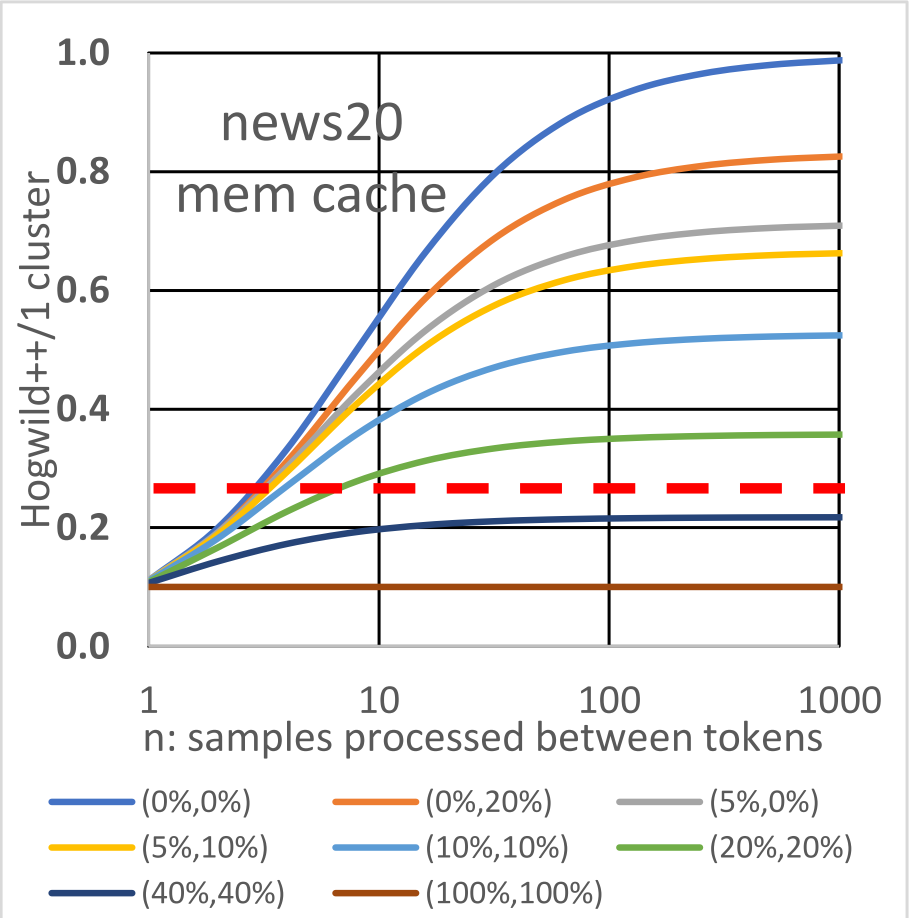
F = 254

F' = 85

Mem-Cache

news20 Dataset

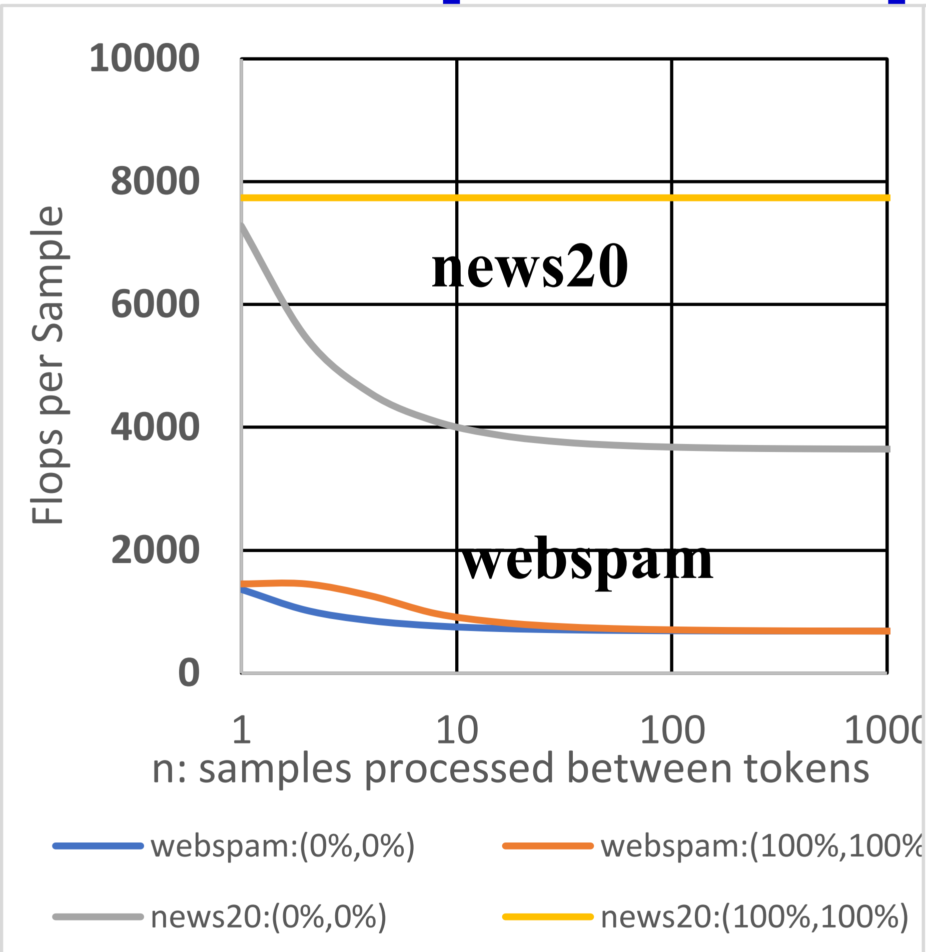


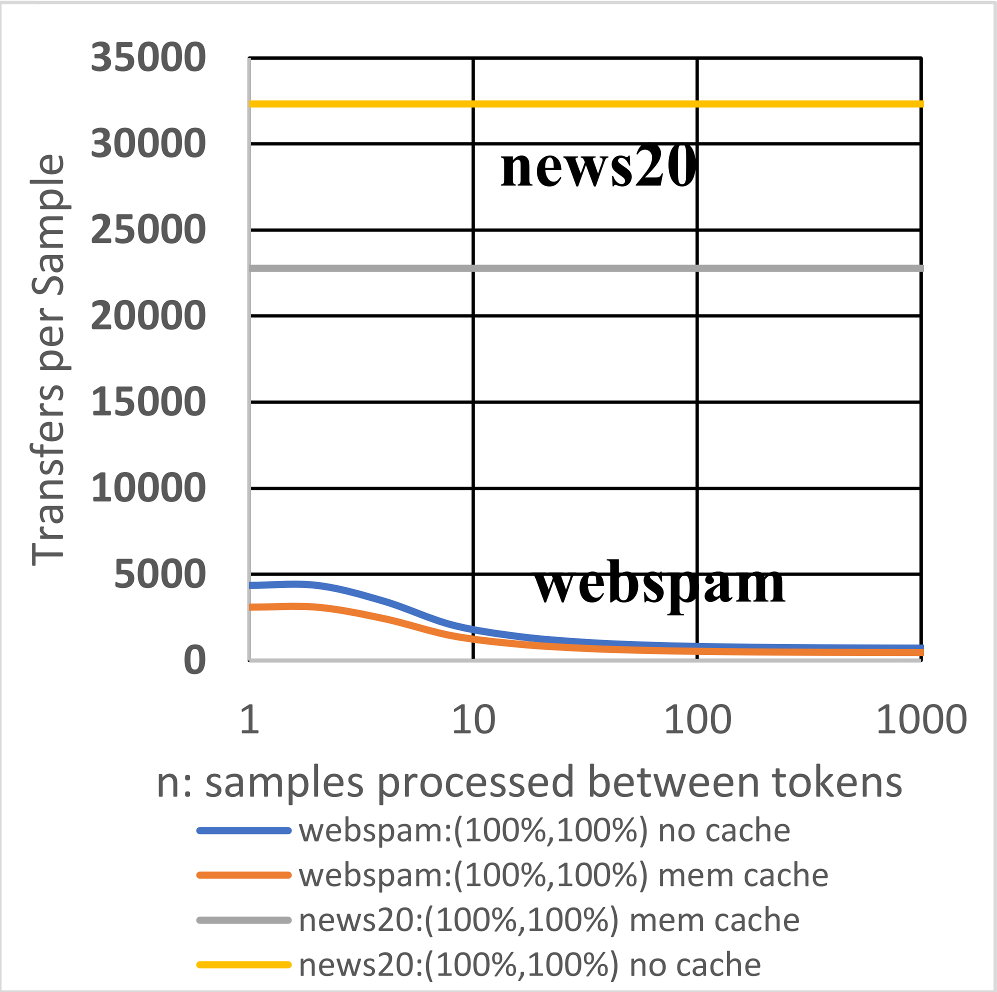


F = 1.3M

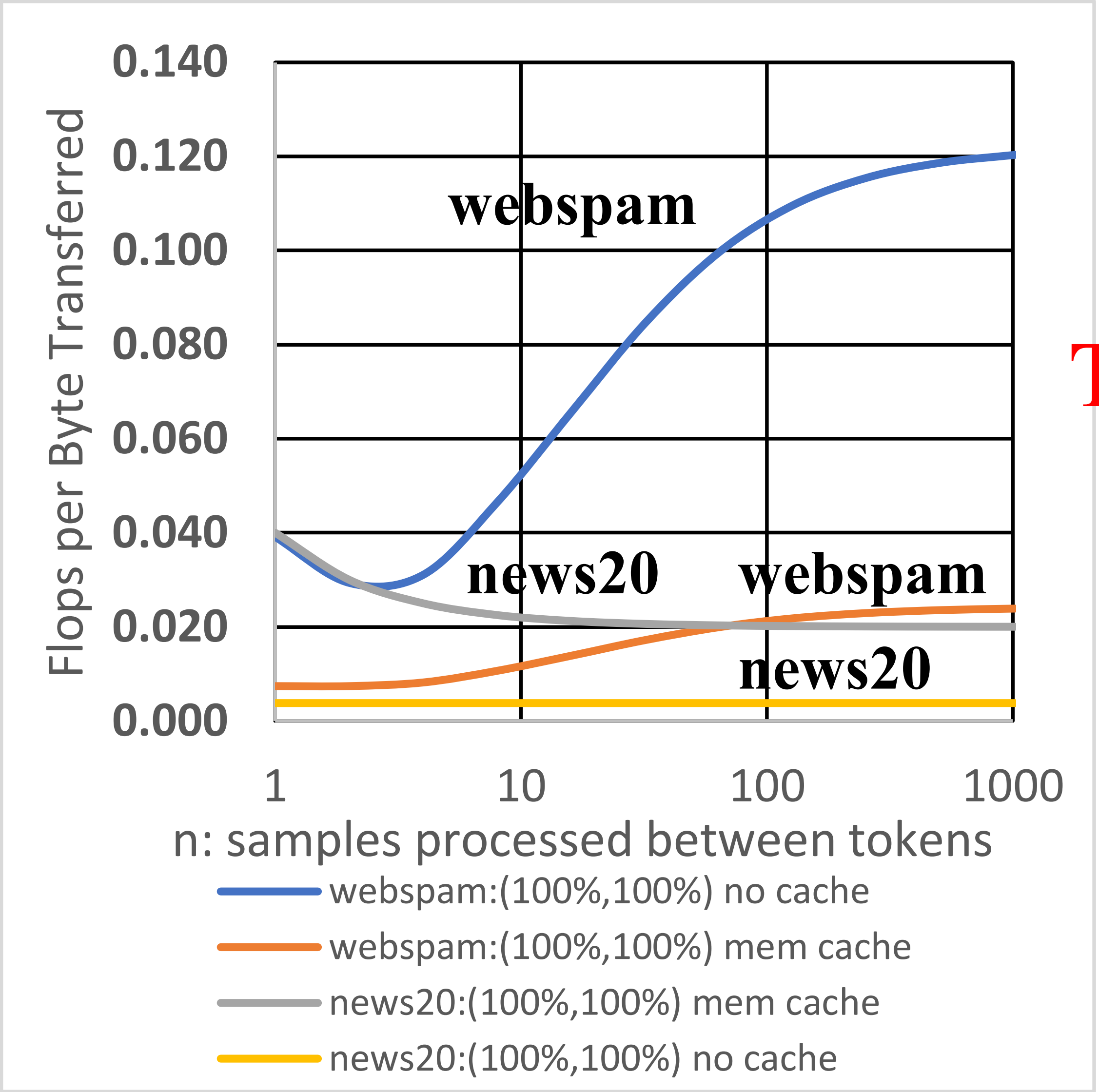
No Cache F' = 455 **Mem-Cache**

“per Sample” Metrics





Intensity



Remember: Today's Systems Are 4-8

Take-aways

- SVM very memory-bound
  - Very low flops/byte of memory bandwidth
  - No reason to have numeric-intensive accelerators
- Sparsity present in many data sets
  - And if recognized can have huge effect on time
- Multi-threading in cluster training a good match
  - Add enough threads to saturate memory
- Code complexity is in cross cluster updates
  - Ideal match to handle va migrating threads
- Efficient AMOs essential – esp. floating point
  - 40% of transfers are involved in AMOs
  - Would be much more for conventional architectures
- **All told: migrating thread good match**

Next Steps

- Consider integrating in code to track changes to objective function
- Consider effects of single thread token processing in more detail
- Develop demo code & look at news20 scaling
- Instrument to understand real-world F′′ and F′′′
- Look at other SGD applications

References

F. Niu, B. Recht, C. Re, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 693–701, USA, 2011. Curran Associates Inc.

C. D. Sa, C. Zhang, K. Olukotun, and C. Ré. Taming the wild: A unified analysis of hog wild! -style algorithms. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2674–2682, Cambridge, MA, USA, 2015. MIT Press.

H. Zhang, C. J. Hsieh, and V. Akella. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 629–638, Dec 2016.

Projections