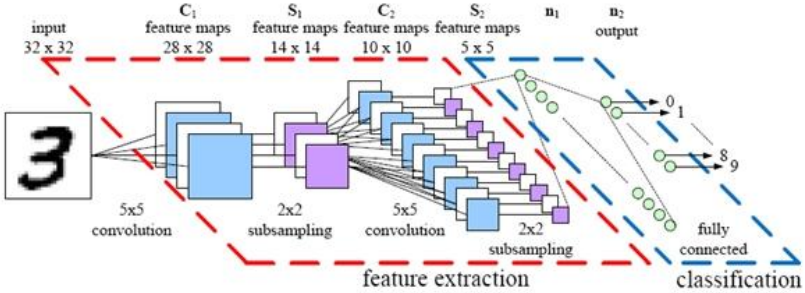
Frameworks for Distributed Machine Learning

Presented by John Alsop, Yasser Shalabi, and Oliver Melvin

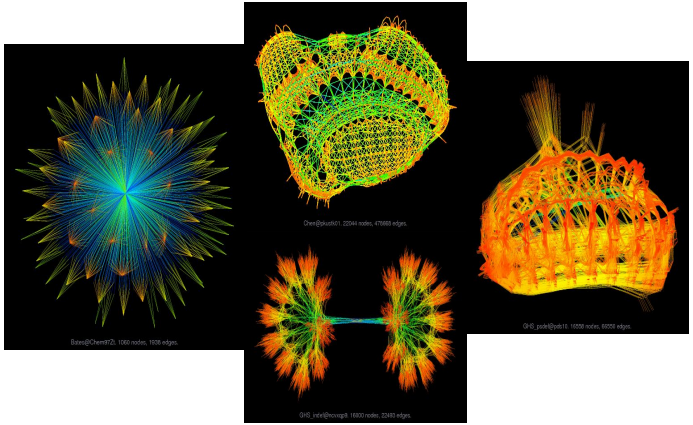
img source: https://www.docuvo.com/whatwedo.html

We present two ML frameworks targeting:

Deep Learning: TensorFlow



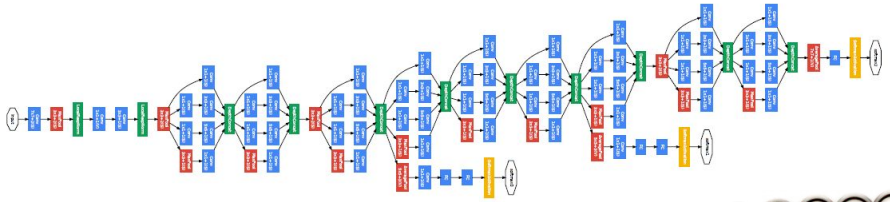
Graph Processing: **Faiter**

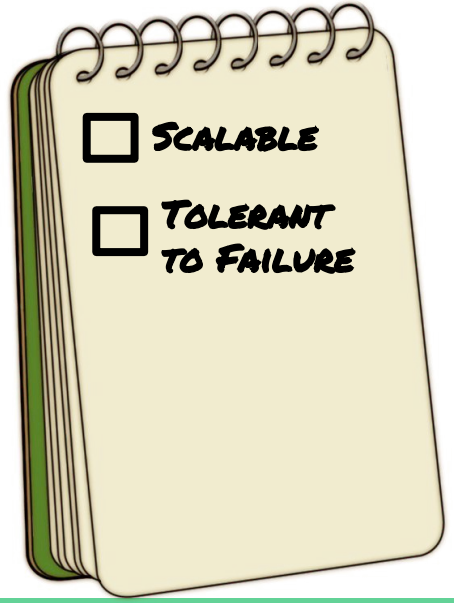


Common themes...

Distributed ML Frameworks: Common Themes

Trend towards **BIG DATA**





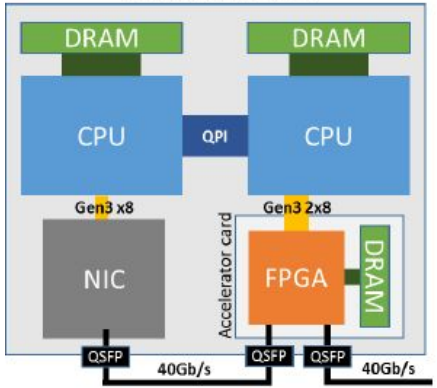
Big models, **big** graphs, **lots** of compute needed

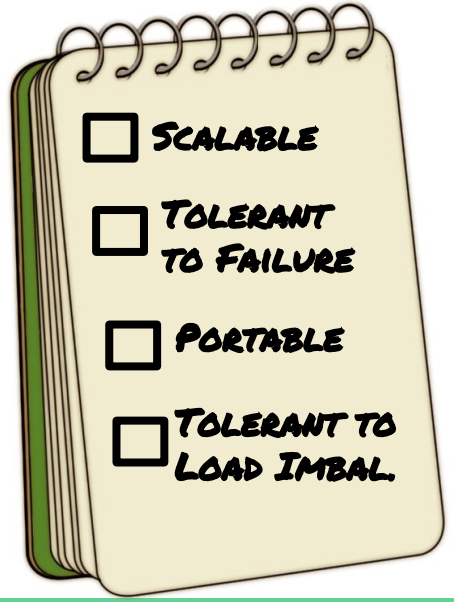
=> Any framework must be **Scalable**

=> Any framework must be **Tolerant to Failure**

Distributed ML Frameworks: Common Themes

Trend towards **Heterogeneity**





GPUs, **FPGAs**, other accelerators increasingly used to accelerate operations

=> Any framework must be **Portable**

=> Any framework must be **Tolerant to Load Imbalance**

TensorFlow: A System for Large-Scale Machine Learning

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, Google Brain

Introduction

Introduction

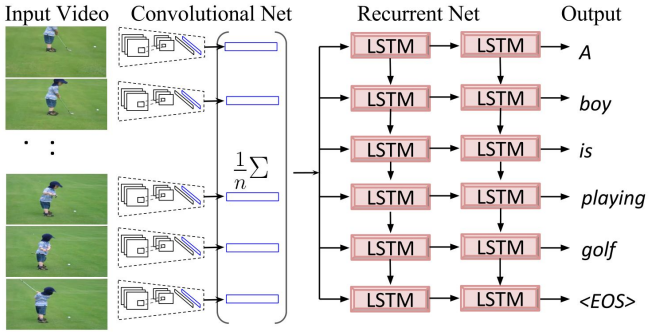
Availability of Data

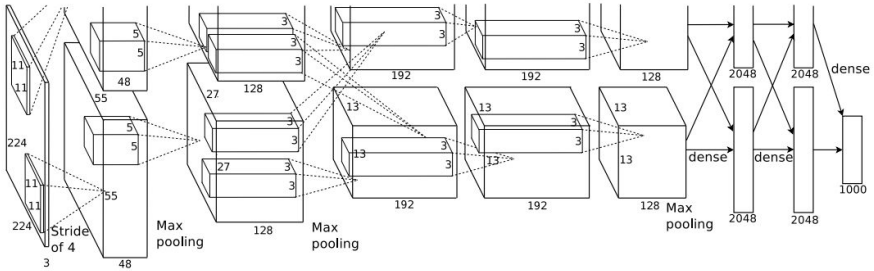
Introduction

Availability of Data

Software Platforms

Introduction





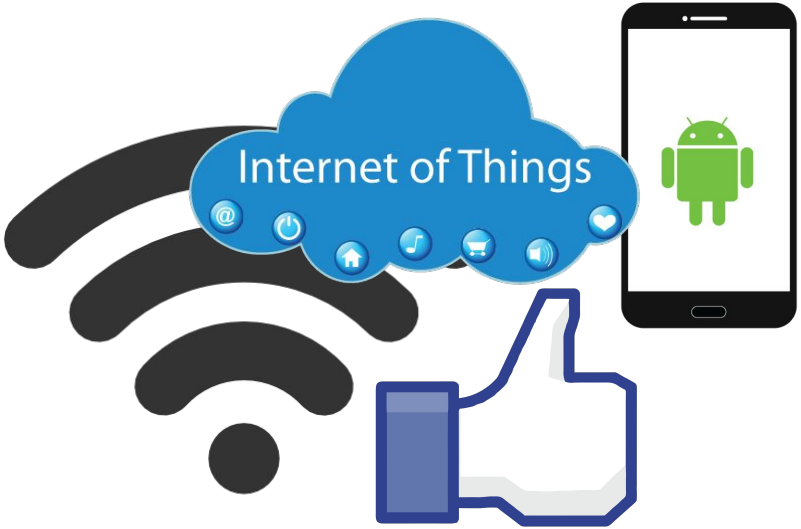
Availability of Data

ML Techniques

Software Platforms

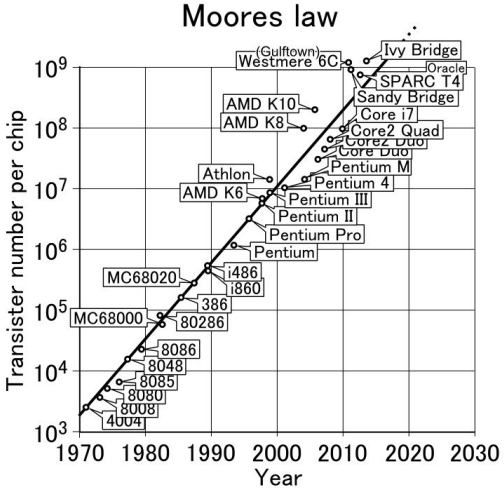
Large Scale Training

- Large data sets are available
- Compute resources are available
  - Warehouse Scale Computers
  - GPGPUs
  - ASIC Accelerators



Large Scale Training

- Large data sets are available
- Compute resources are available
  - Warehouse Scale Computers
  - GPGPUs
  - ASIC Accelerators



Large Scale Training

- Large data sets are available
- Compute resources are available
  - Warehouse Scale Computers
  - GPGPUs
  - ASIC Accelerators

Team	Year	Place	Error (top-5)
XRCE (pre-neural-net explosion)	2011	1st	25.8%
Supervision (AlexNet)	2012	1st	16.4%
Clarifai	2013	1st	11.7%
GoogLeNet (Inception)	2014	1st	6.66%
Andrej Karpathy (human)	2014	N/A	**5.1%**
BN-Inception (Arxiv)	2015	N/A	4.9%
Inception-v3 (Arxiv)	2015	N/A	3.46%

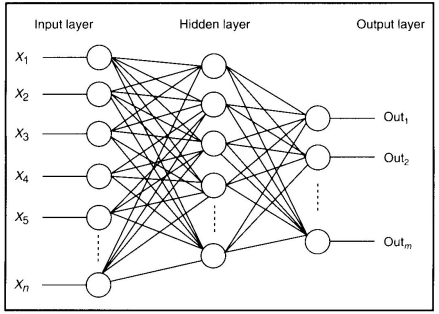
- Larger more models, complex techniques performing better
  - But they need more data and more time for convergence
- **So how can we scale training?**

Background: NN Training

Process:

Take input image

Compute loss function (forward pass)



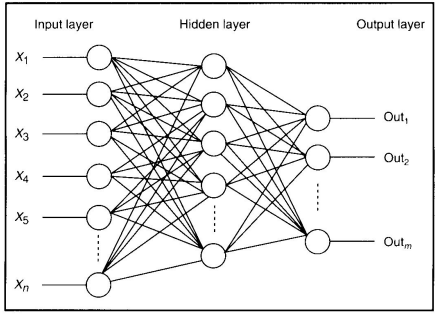
Background: NN Training

Process:

Take input image

Compute loss function (forward pass)

Compute error gradients (backward pass)



Background: NN Training

Process:

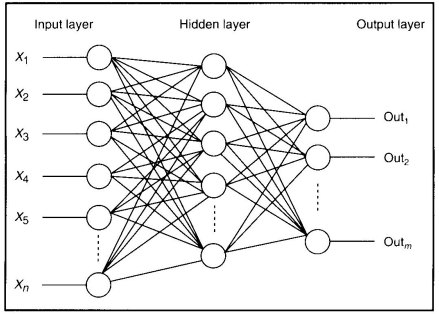
Take input image

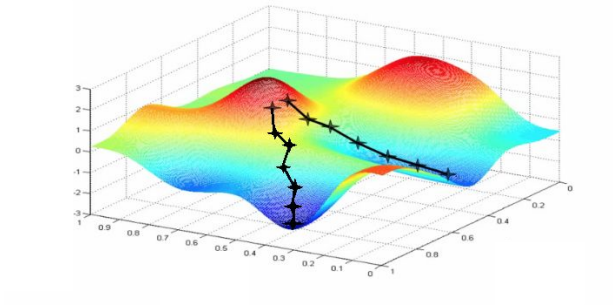
Compute loss function (forward pass)

Compute error gradients (backward pass)

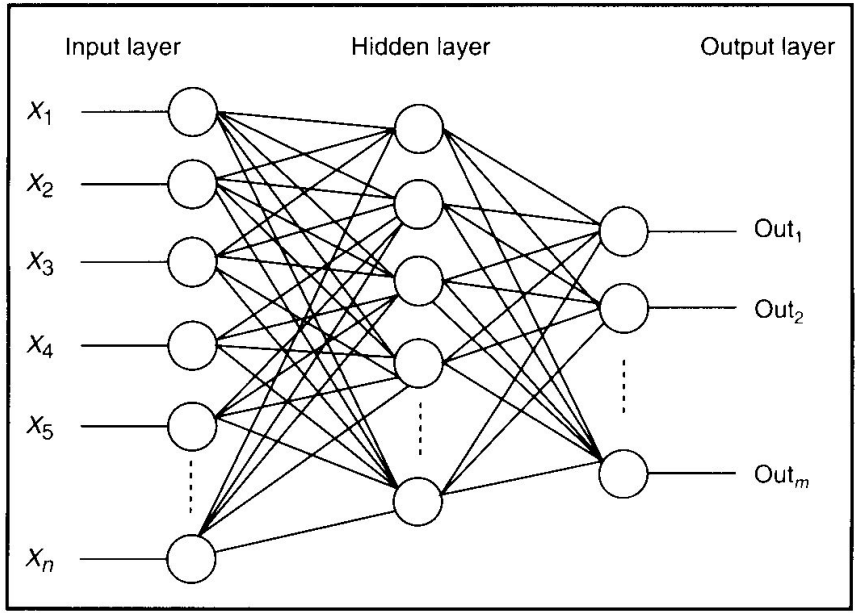
Update weights

Repeat



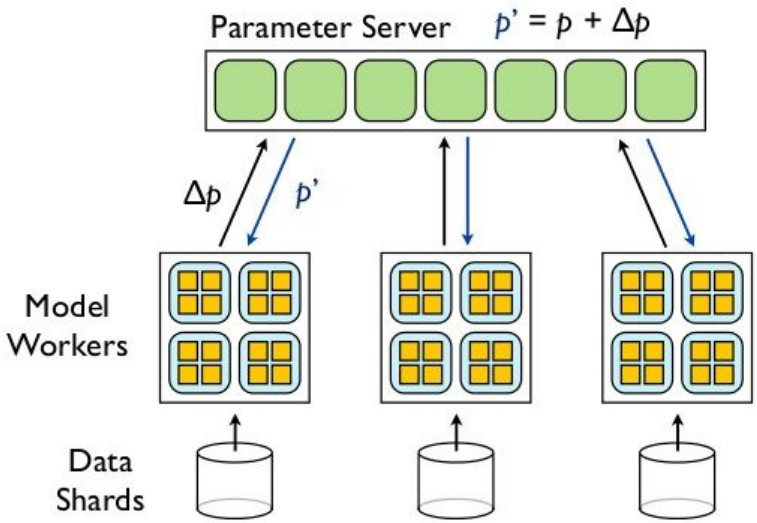


Background: Dist Belief, parameter-server arch



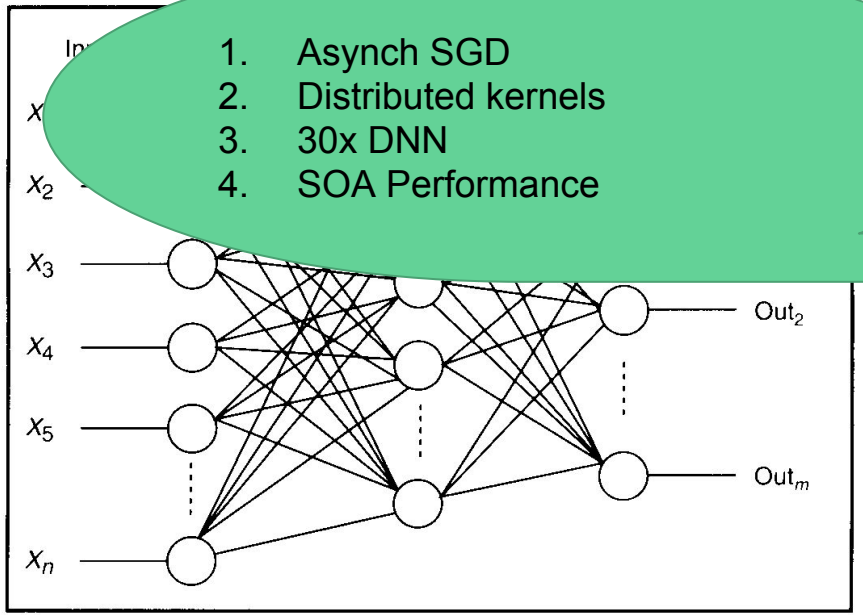
Data Parallelism:

Asynchronous Distributed Stochastic Gradient Descent



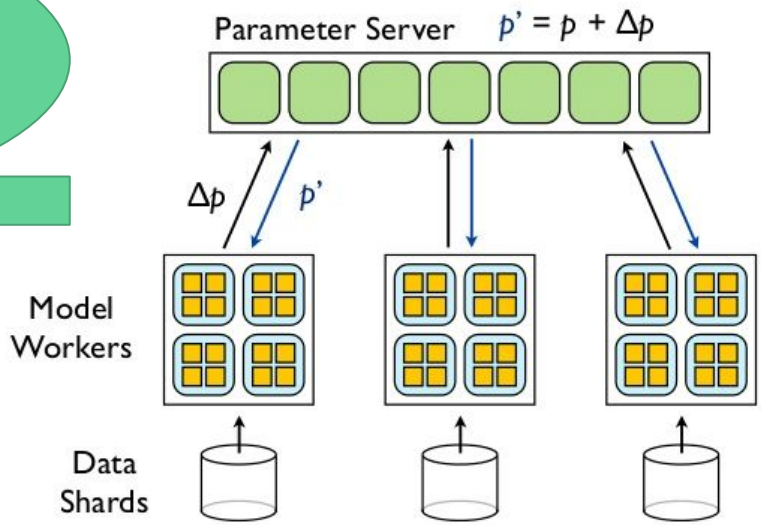
Background: Dist Belief, parameter-server arch

1. Asynch SGD
2. Distributed kernels
3. 30x DNN
4. SOA Performance



Data Parallelism:

Asynchronous Distributed Stochastic Gradient Descent



Background: Shortcomings of DistBelief

1. Difficulty of implementing new layers
    a. C++ classes implement layers
    b. Configuration file defines DNN architecture
    c. Not flexible enough for researchers

Background: Shortcomings of DistBelief

1. Difficulty of implementing new layers
    a. C++ classes implement layers
    b. Configuration file defines DNN architecture
    c. Not flexible enough for researchers
2. Refining Algorithms
    a. SGD is the heart of the training -- finalized in the parameter server
    b. Need atomicity for some techniques -- get/put interface cannot accommodate

Background: Shortcomings of DistBelief

1. Difficulty of implementing new layers
   a. C++ classes implement layers
   b. Configuration file defines DNN architecture
   c. Not flexible enough for researchers
2. Refining Algorithms
   a. SGD is the heart of the training -- finalized in the parameter server
   b. Need atomicity for some techniques -- get/put interface cannot accommodate
3. Supporting new algorithms
   a. If it doesn’t conform to feed-forward it doesnt map well to DistBelief
   b. EM, RF, RL, AdvMl,

Background: Shortcomings of DistBelief

1. Difficulty of implementing new layers
   a. C++ classes implement layers
   b. Configuration file defines DNN architecture
   c. Not flexible enough for researchers
2. Refining Algorithms
   a. SGD is the heart of the training -- finalized in the parameter server
   b. Need atomicity for some techniques -- get/put interface cannot accommodate
3. Supporting new algorithms
   a. If it doesn’t conform to feed-forward it doesnt map well to DistBelief
   b. EM, RF, RL, AdvMl,
4. Scaling down to other environments
   a. Designed to run on distributed cluster of multi-cores
   b. Augmented for GPGPU support for Conv NN

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism
2. Provides DFGs for primitive operators
    a. Softmax, convolution, MM, …
    b. Makes it easy to experiment with novel layers
    c. Automatic gradient calculation

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism
2. Provides DFGs for primitive operators
    a. Softmax, convolution, MM, ...
    b. Makes it easy to experiment with novel layers
    c. Automatic gradient calculation
3. Deferred execution
    a. Offload the larger chunks where possible...

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
   a. Makes it easy to extract the parallelism
2. Provides DFGs for primitive operators
   a. Softmax, convolution, MM, ...
   b. Makes it easy to experiment with novel layers
   c. Automatic gradient calculation
3. Deferred execution
   a. Offload the larger chunks where possible...
4. Common Abstraction for Accelerators
   a. Easy to integrate new accelerators into the fold
   b. The operators are specialized for different devices
5. Common data primitive : **Tensor**

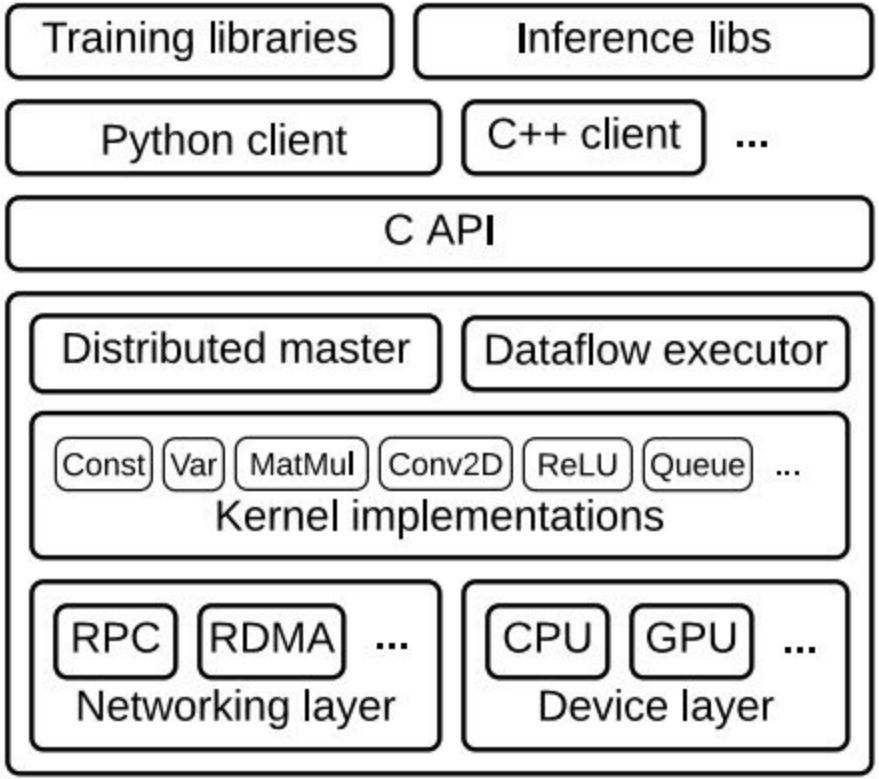
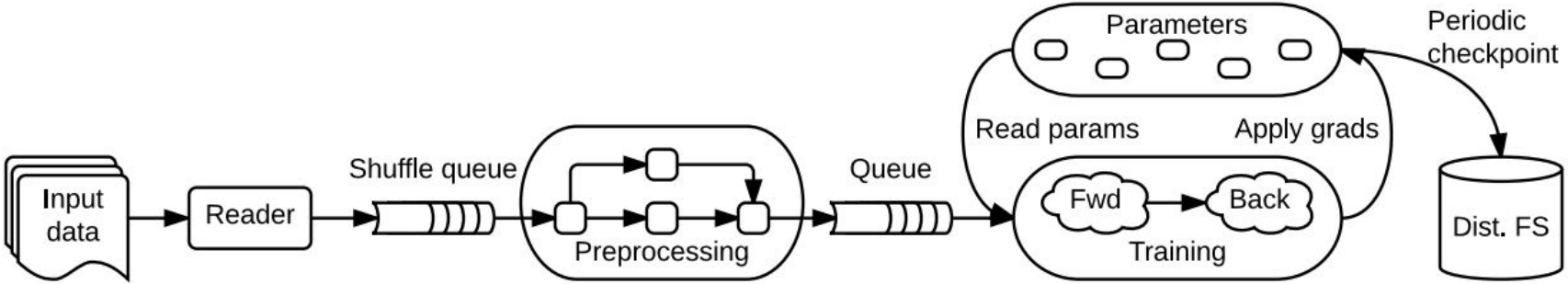


Figure 6: The layered TensorFlow architecture.

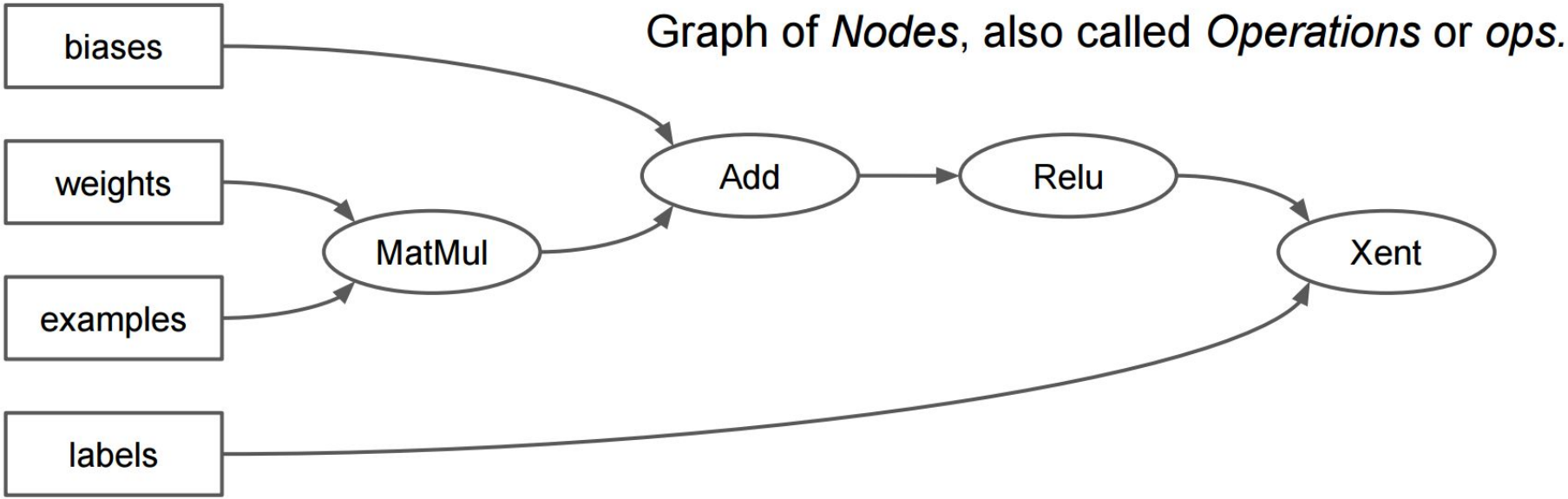
Execution Model

Execution Model

- Single DFG represents all computation and state for ML algorithm
  - Input preprocessing, mathematical operators, parameters, parameter update rules
  - Communication explicit, simplifying scheduling and partitioning

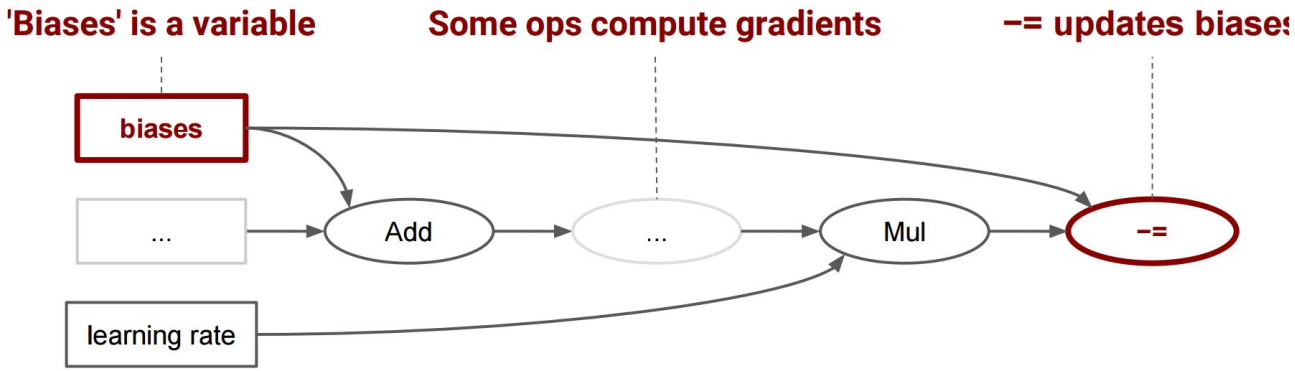


Computation is a DFG



Execution Model

- Single DFG represents all computation and state for ML algorithm
  - Input preprocessing, mathematical operators, parameters, parameter update rules
  - Communication explicit, simplifying scheduling and partitioning
- Differences with existing DF systems:
  - Concurrent execution on overlapping subgraphs supported
  - Individual vertices contain sharable, mutable state



Execution Model

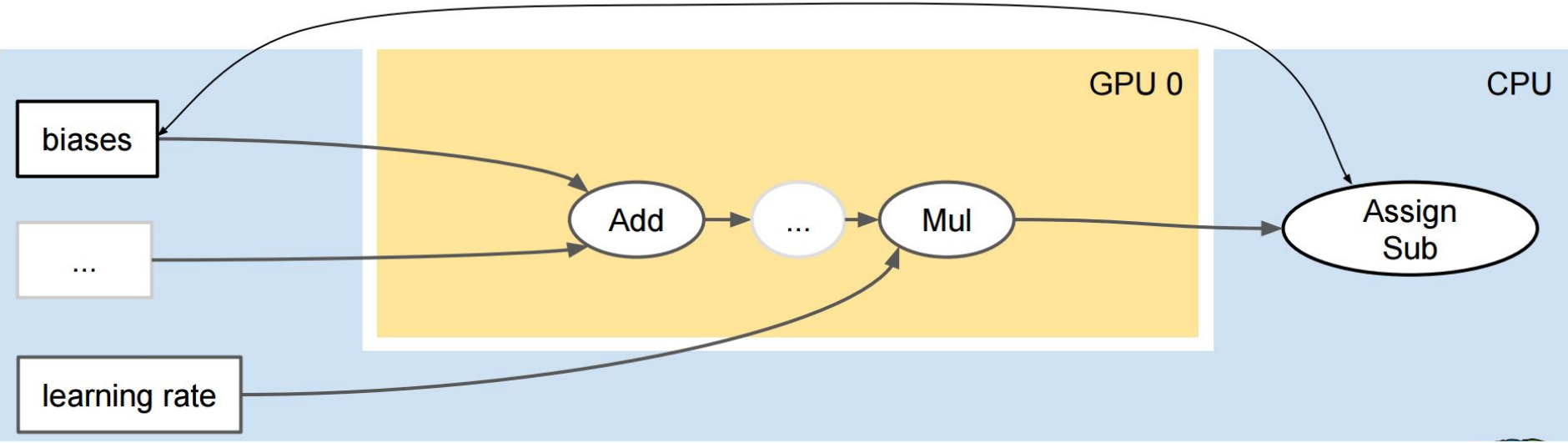
- Single DFG represents all computation and state for ML algorithm
  - Input preprocessing, mathematical operators, parameters, parameter update rules
  - Communication explicit, simplifying scheduling and partitioning
- Differences with existing DF systems:
  - Concurrent execution on overlapping subgraphs supported
  - Individual vertices contain sharable, mutable state

mutable state is **critical** when training large models

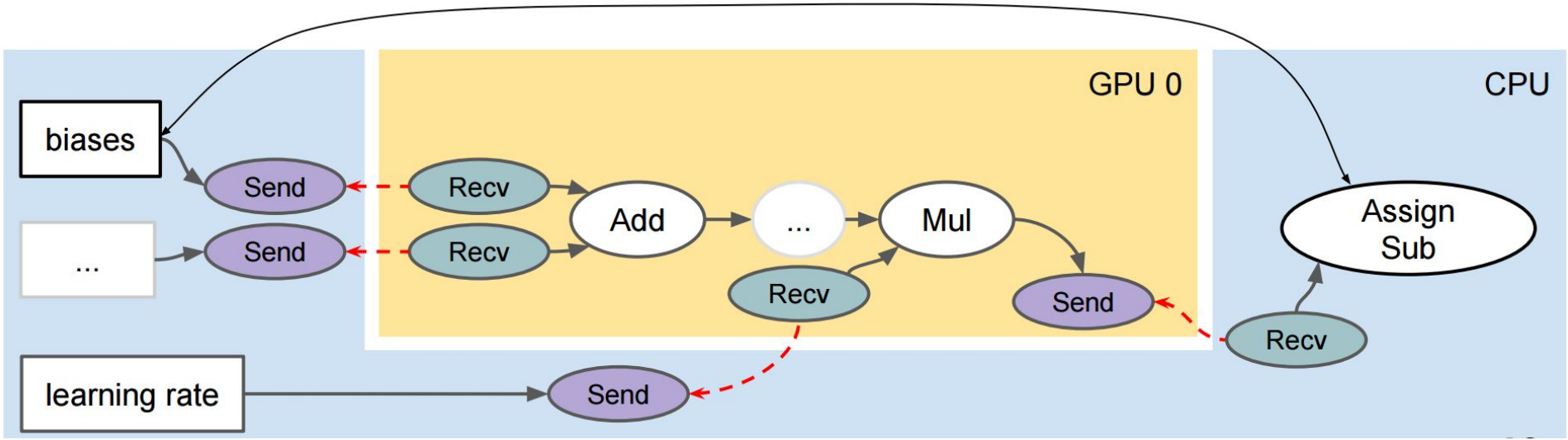
Compute + Mutable State = PS++

Distributed

Communication is explicit…

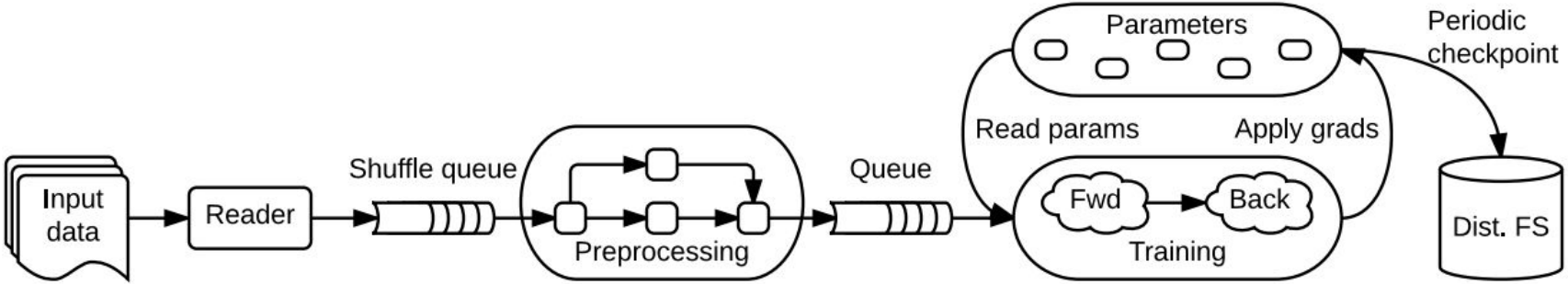


TensorFlow handles the glue



Fault Tolerance

- Days or many hours to train models -- fault tolerance is key
- RDD is overkill
  - Strong consistency is not needed for ML
- User-level checkpointing operations and client library for configuration
  - SAVE/RESTORE

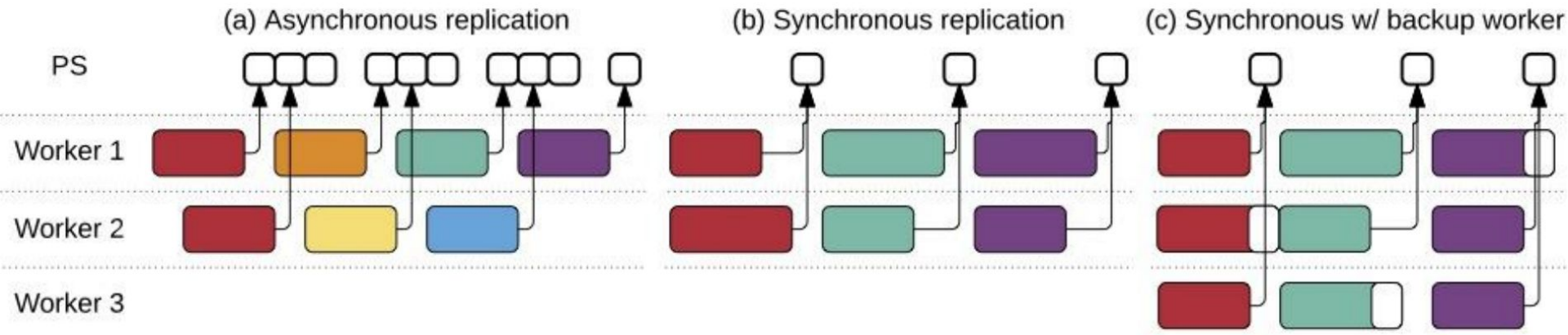


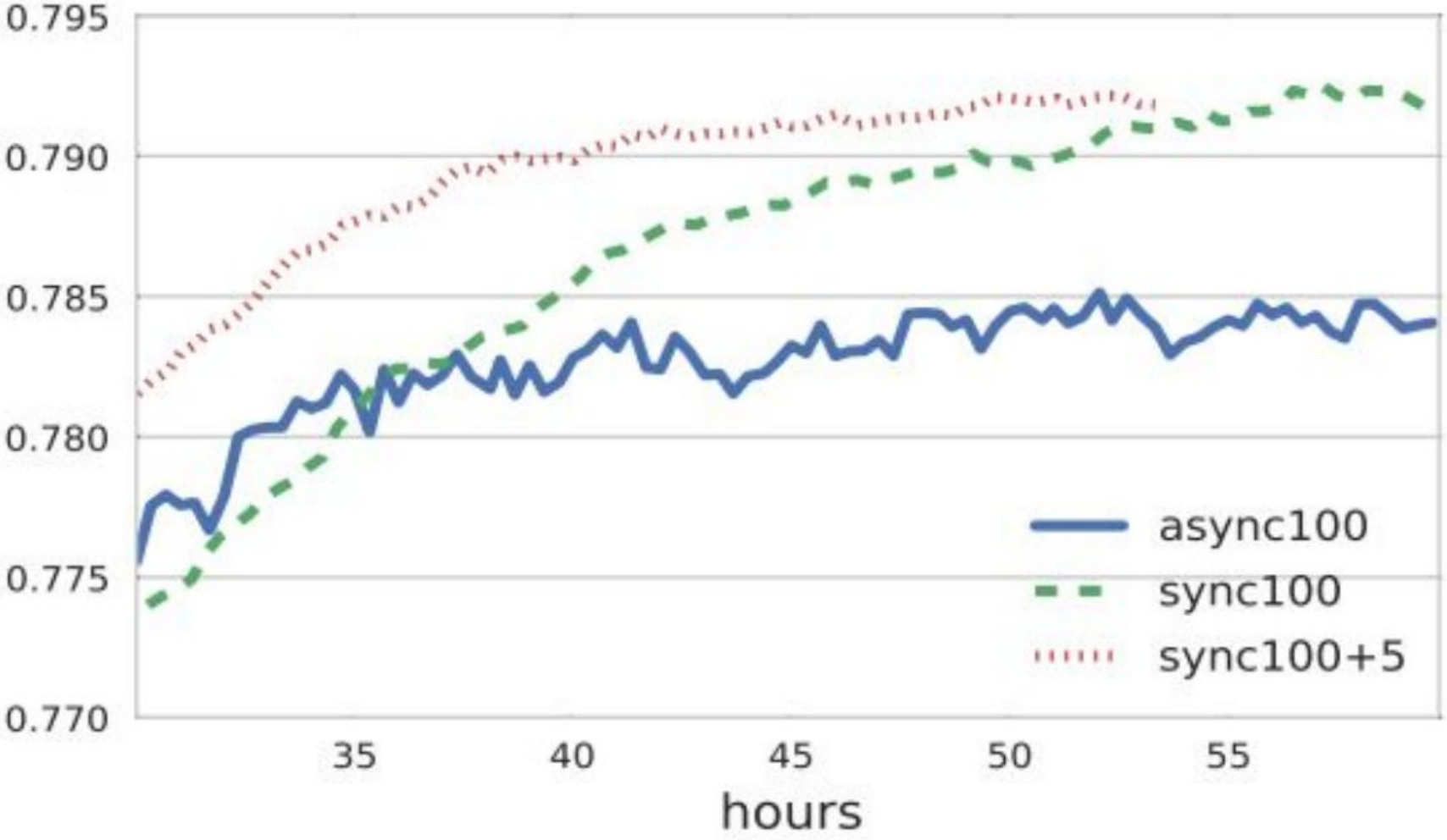
Synchronous!

Time-to-Quality vs Time-per-Update

- Recall earlier algorithm
  - Aggregate update delayed by stragglers
- SGD found to be robust to asynchrony
  - Asynchronous = better utilization…
  - But with GPGPUs…
- TensorFlow can handle Aynch or Synch updates…
  - Also, synchronous + **backup workers**
  - **Idea:** have K backup workers and N workers, then simply take updates from first N workers that complete

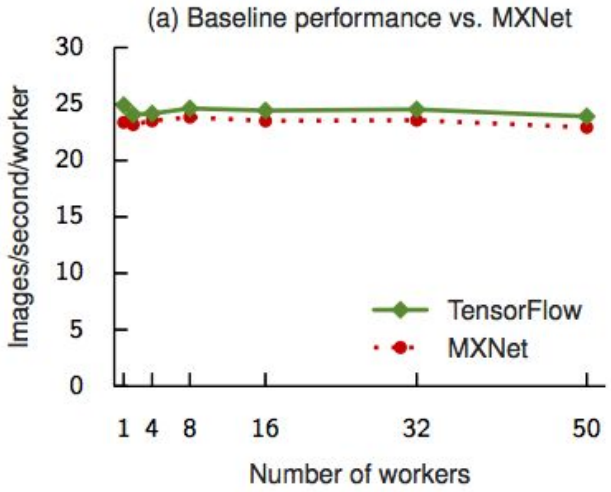
Time-to-Quality vs Time-per-Update




0.795
0.790
0.785
0.780
0.775
0.770
35
40
45
50
55
hours
async100
sync100
sync100+5

Highlights from Results

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445



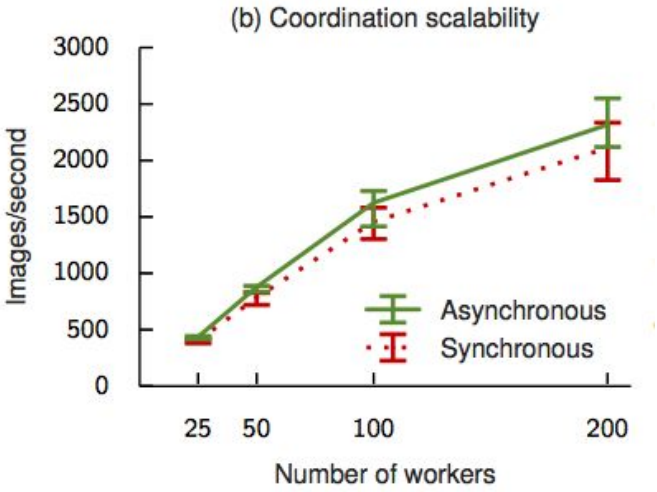
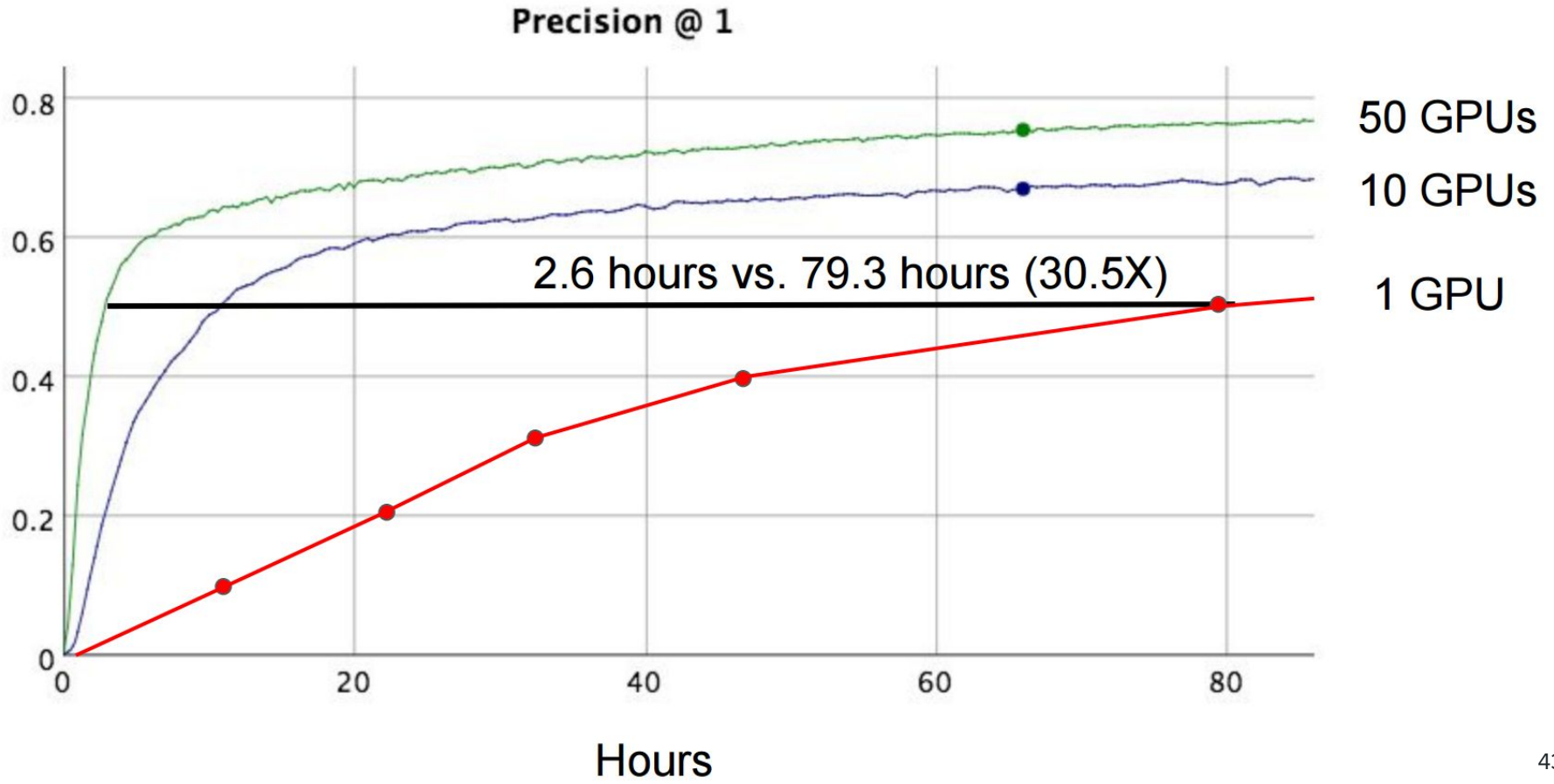


Image Model Training Time



Conclusion

- We didn’t cover everything
  - Dynamic control flow
  - Extensibility studies
- **Key lesson:** comprehensive look at problem necessary to design a good solution for it
  - Asynchrony was okay but throughput oriented GPGPUs made synchronous better…
  - RDD level fault tolerance was not necessary
  - Heterogeneity built into the design
- Focusing on the user
  - Design/experiment, train, and deploy

A Fault-Tolerant Framework for Asynchronous Iterative Computations in Cloud Environments

Zhigang Wang, Lixin Gao, Yu Gu, Yubin Bao, and Ge Yu

A Fault-Tolerant Framework for Asynchronous

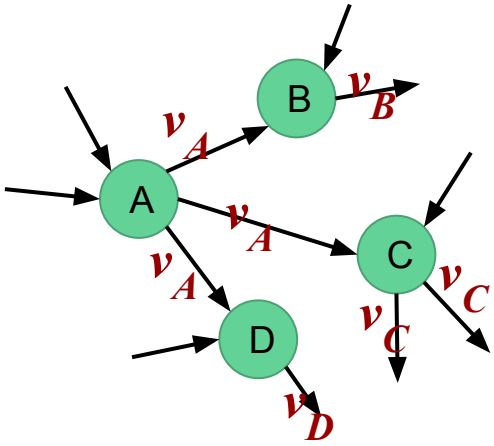
Iterative Computations in Cloud Environments

Iterative Computations = Graph Algorithms

Graph as input → iteratively extract meaning → output updated graph

Iterative update function:

$$v_j^{k+1} = c_j \oplus \sum_{i \in \Gamma^{in}(j)} \oplus g_{\{i,j\}}(v_i^k)$$



Converges to desired result

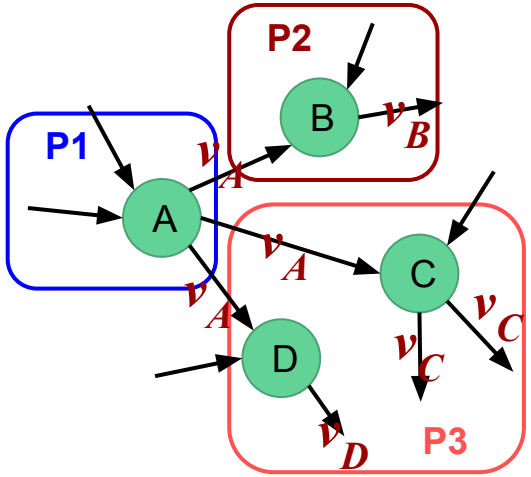
Examples:

- PageRank
- SSSP
- Adsorption
- Sparse Jacobi linear eq. solver

A Fault-Tolerant Framework for Asynchronous Iterative Computations **in Cloud Environments**

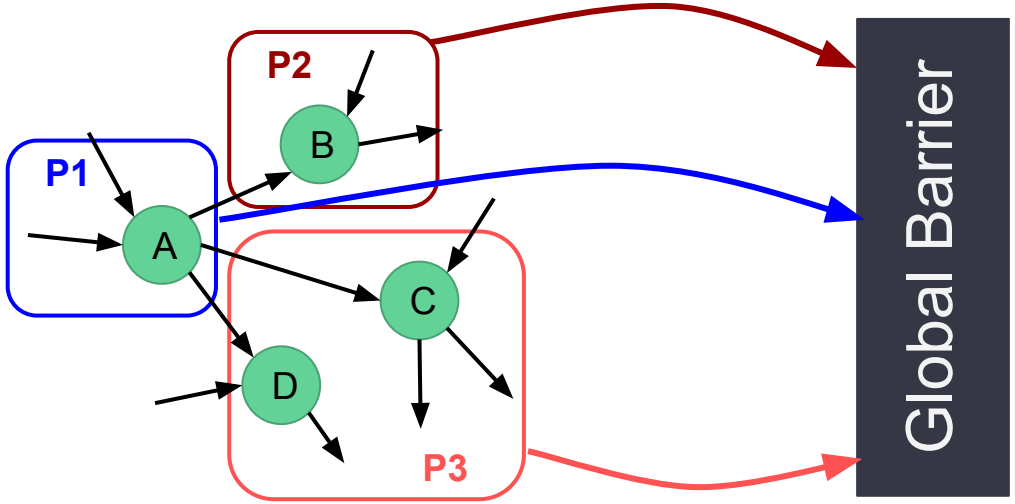
In Cloud Environments

Nodes are partitioned across processors, communicate via MPI



In Cloud Environments

Nodes are partitioned across processors, communicate via MPI



All processors synchronize after each iteration => **scales poorly**

Amount of work depends on connectivity => **load imbalance**

A Fault-Tolerant Framework for **Asynchronous** Iterative Computations in Cloud Environments

Asynchronous: Maiter

Can we avoid global barrier for some algorithms?

Observe that: $v_j^{k+1} = c_j \oplus \sum_{i \in \Gamma^{in}(j)}$

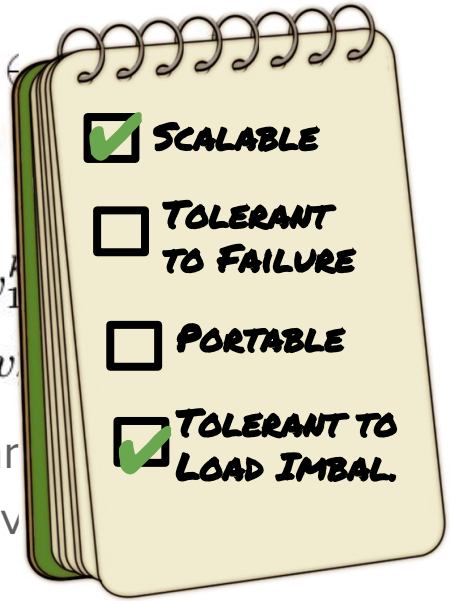
Can be rewritten using:

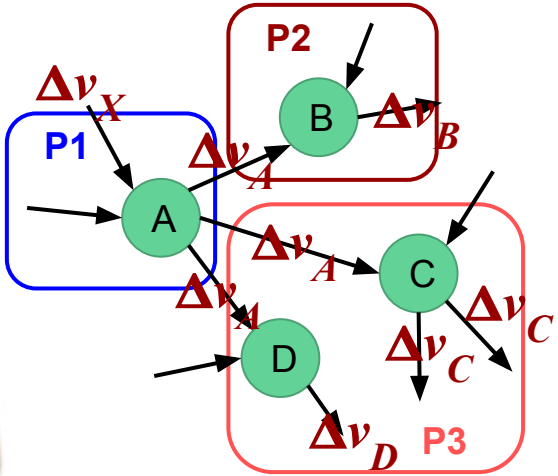
$$v_j^k = g_{\{1,j\}}(v_1^{k-2}) \oplus g_{\{1,j\}}(\Delta v_1^{k}$$
$$g_{\{n,j\}}(v_n^{k-2}) \oplus g_{\{n,j\}}(\Delta v$$

And $\Delta v_{1..n}^{1..k}$ can be applied asynchr

- $\oplus$ is commutative and associativ
- $g()$ is distributive over $\oplus$

(Just need to define initial Δv_i^1 for all i)



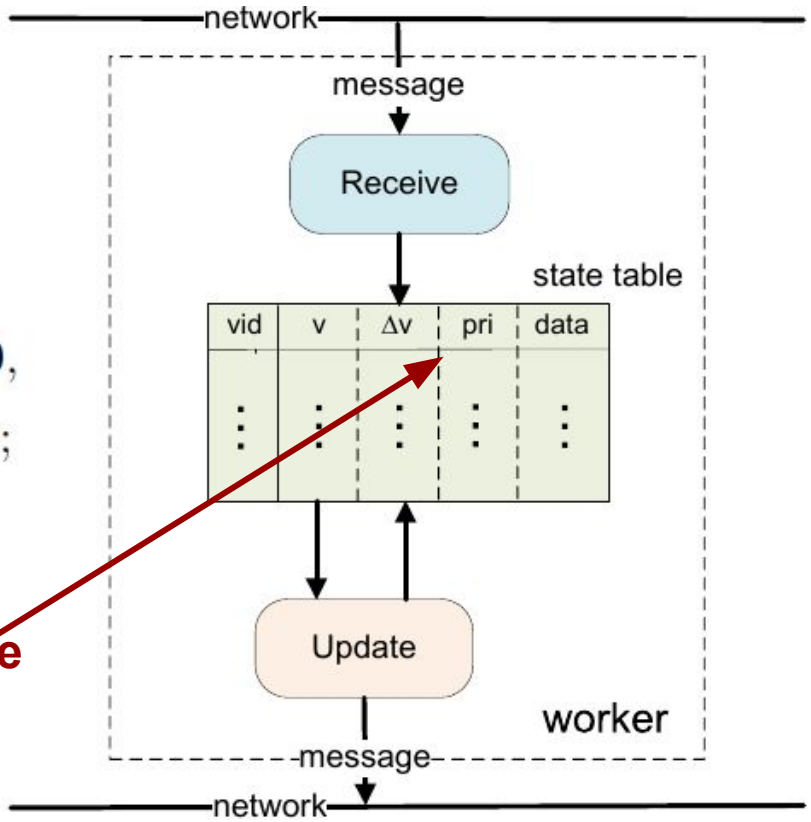


Maiter Implementation

$$\text{receive:} \begin{cases} \text{Whenever receiving } m_j, \\ \quad \Delta\check{v}_j \leftarrow \Delta\check{v}_j \oplus m_j. \end{cases}$$

$$\text{update:} \begin{cases} \check{v}_j \leftarrow \check{v}_j \oplus \Delta\check{v}_j; \\ \text{For any } h, \text{ if } g_{\{j,h\}}(\Delta\check{v}_j) \neq \mathbf{0}, \\ \quad \text{send value } g_{\{j,h\}}(\Delta\check{v}_j) \text{ to } h; \\ \Delta\check{v}_j \leftarrow \mathbf{0}, \end{cases}$$

May sort by priority to speed convergence



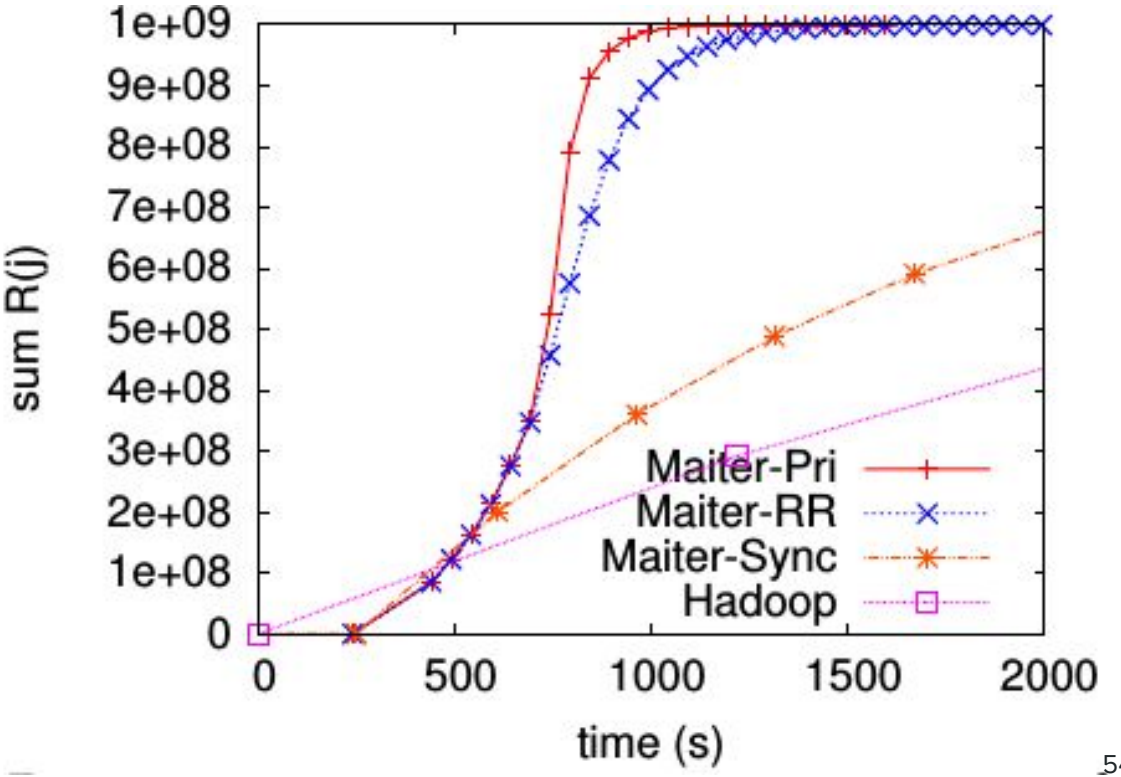
Maiter Evaluation

Hadoop: synchronous streaming framework

Maiter-Sync: Synchronous delta-based framework

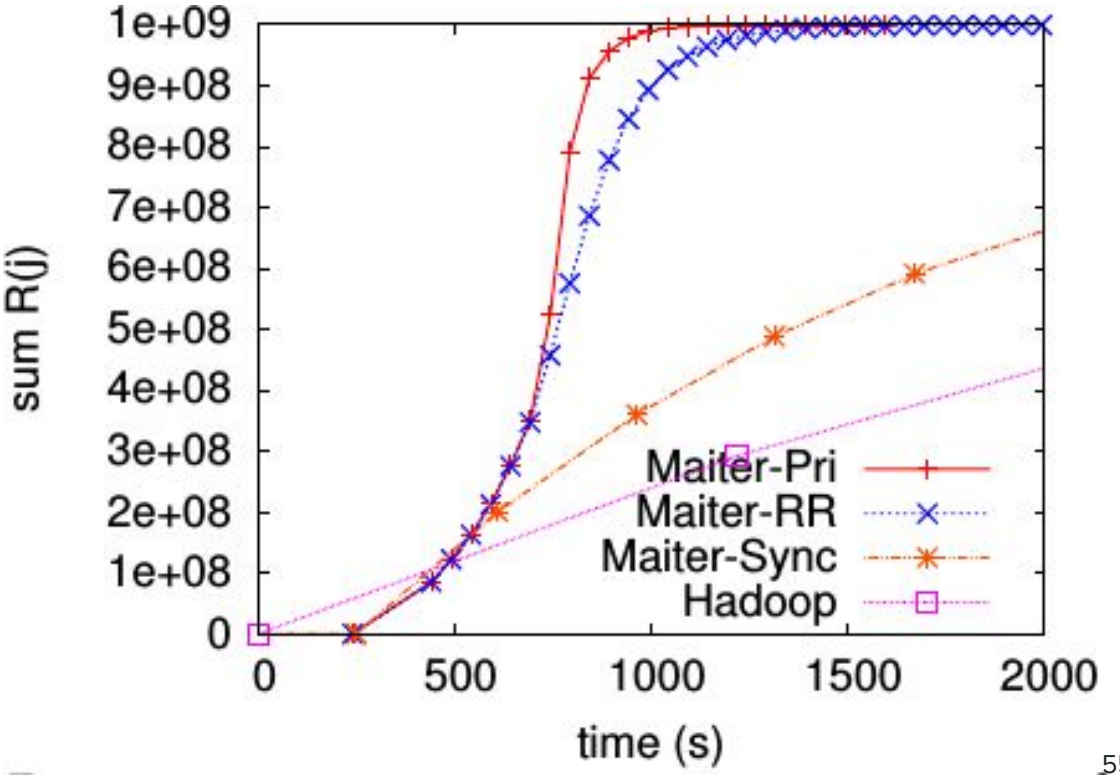
Maiter-RR: Asynchronous Maiter, process state table Round-Robin

Maiter-Pri: Asynchronous Maiter, process state table based on priority



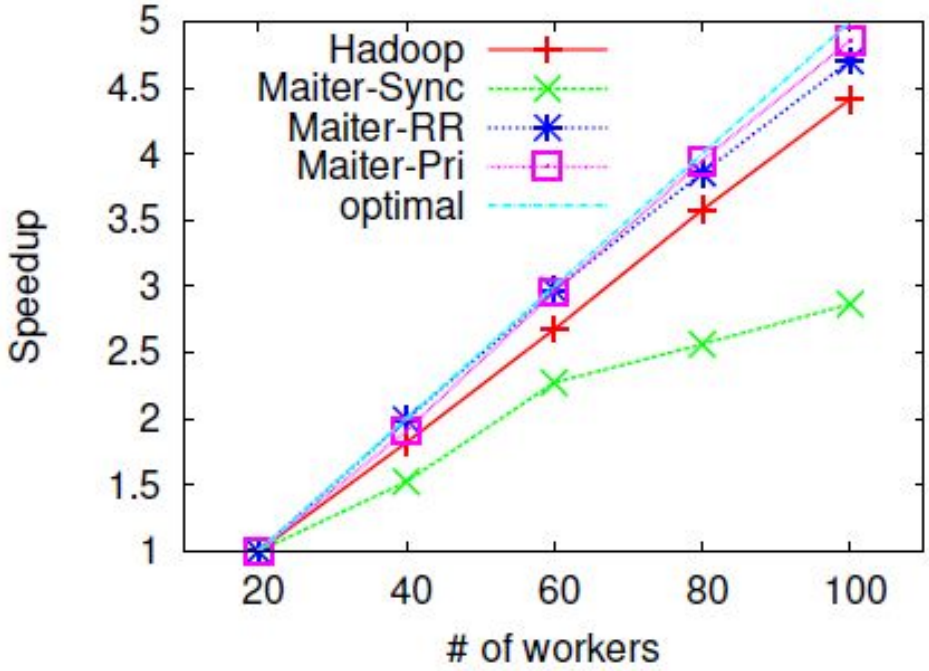
Maiter Evaluation

PageRank on billion node synthetic graph: Asynchrony and priority sorting help Maiter converge faster



Maiter Evaluation

Near optimal scaling

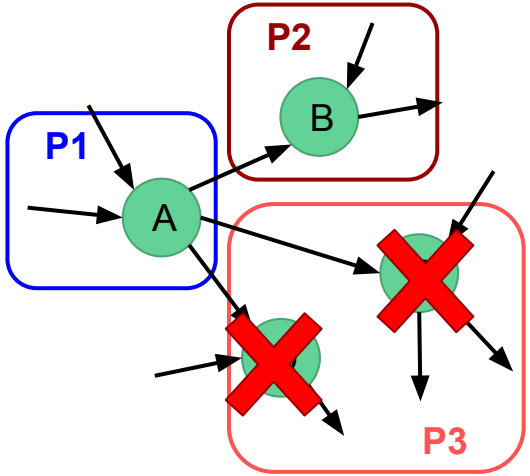


A **Fault-Tolerant** Framework for Asynchronous Iterative Computations in Cloud Environments

Fault-Tolerant: Faiter

What happens if some nodes fail?

Trivial solution: roll back all nodes to last checkpoint, or to initial state



Fault-Tolerant: Faiter

What happens if some nodes fail?

Trivial solution: roll back all nodes to

r to initial state

Checkpoints are expensive in an a

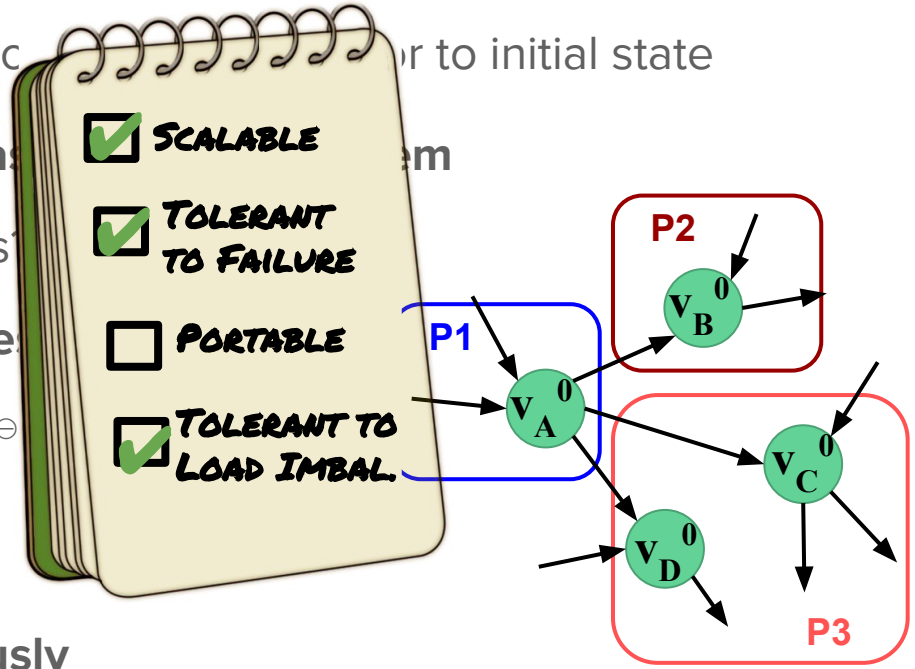
m

Can we avoid rolling back all nodes?

Yes! Can roll back only failed node

- $g()$ is distributive over new op $\ominus$
- $x \ominus x = 0$
- $(x \oplus y) \ominus z = x \oplus (y \ominus z)$

Also: can checkpoint asynchronously



Faiter Implementation

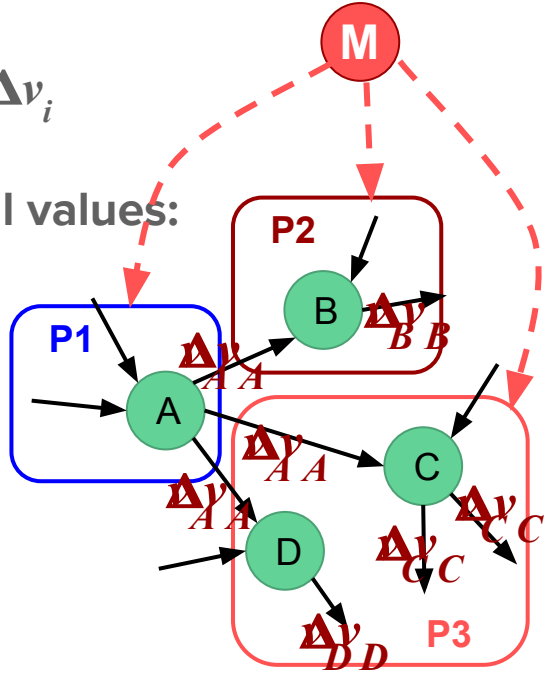
1) Master detects node failure, broadcasts recovery signal

2) Run a synchronous iteration using v_i rather than Δv_i

3) Resume asynchronous operation using new initial values:

$$\tilde{v}_j^0 = \begin{cases} \mathbf{0}, j \in \bigcup_{N \in \mathbb{N}_F} V_N \\ v_j^f, j \in \bigcup_{N \in \mathbb{N}_S} V_N \end{cases}$$

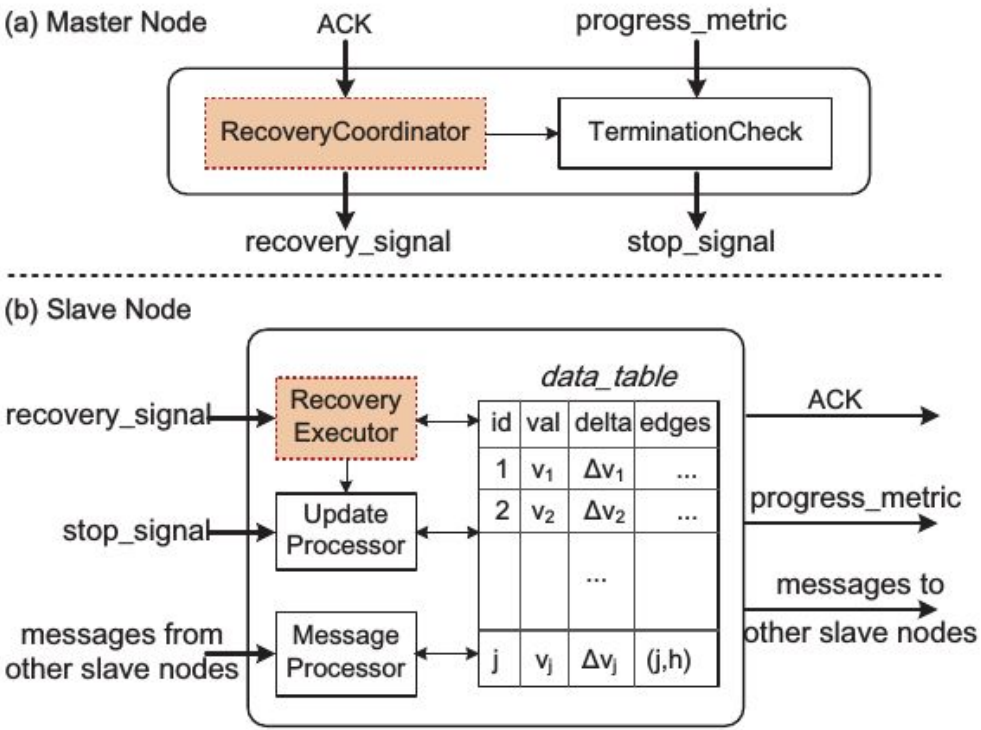
$$\Delta \tilde{v}_j^0 = \tilde{v}_j^1 \ominus \tilde{v}_j^0 = \left(c_j \oplus \left(\sum_{i \in \Gamma_j^{in}} \oplus g_{\{i,j\}}(\tilde{v}_i^0) \right) \right) \ominus \tilde{v}_j^0$$



Faiter Implementation

Algorithm 1: RecoveryCoordinator()

1 suspending *TerminationCheck*
2 sending *recovery_signal* of "load" to *RecoveryExecutor* on $N \in \mathbb{N}$
3 **while** *Number of ACKs of "load"* $\neq |\mathbb{N}|$ **do**
4 waiting for new *ACK*
5 sending *recovery_signal* of "flush" to *RecoveryExecutor* on $N \in \mathbb{N}$
6 **while** *Number of ACKs of "flush"* $\neq |\mathbb{N}|$ **do**
7 waiting for new *ACK*
8 sending *signal_recovery* of "construct" to *RecoveryExecutor* on $N \in \mathbb{N}$
9 **while** *Number of ACKs of "construct"* $\neq |\mathbb{N}|$ **do**
10 waiting for new *ACK*
11 waking *TerminationCheck*
12 sending *signal_recovery* of "restart" to *RecoveryExecutor* on $N \in \mathbb{N}$
13 **return**;



Faiter Evaluation

FR-Scratch: All nodes roll back to **0** on failure

FR-WORB: Failed nodes roll back to **0**

FR-WAC: Failed nodes roll back to last asynchronous checkpoint

T1 = 0.1 runtime, T2 = 0.5, T3 = 0.9

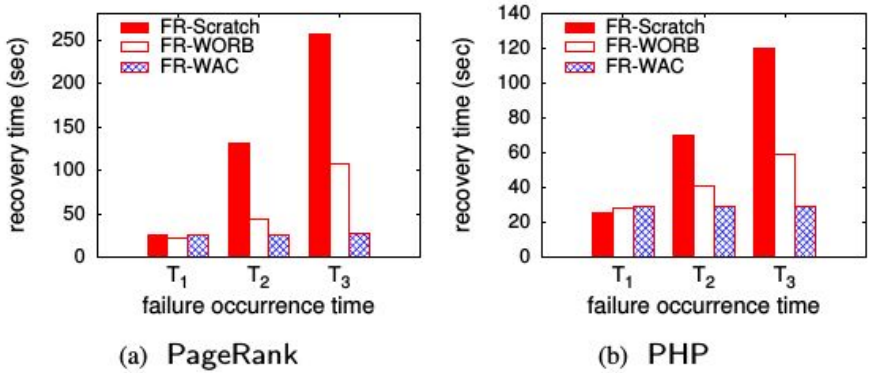


Figure 4. Recovery time on Orkut

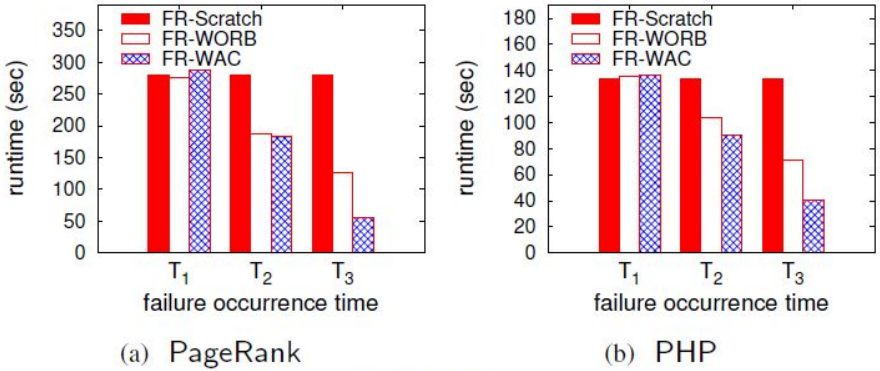


Figure 7. Runtime on Orkut

Faiter Evaluation

Failure recovery with and without asynchronous checkpointing is helpful at higher T

(with checkpointing is generally better)

At low T, barrier overheads can do more harm than good

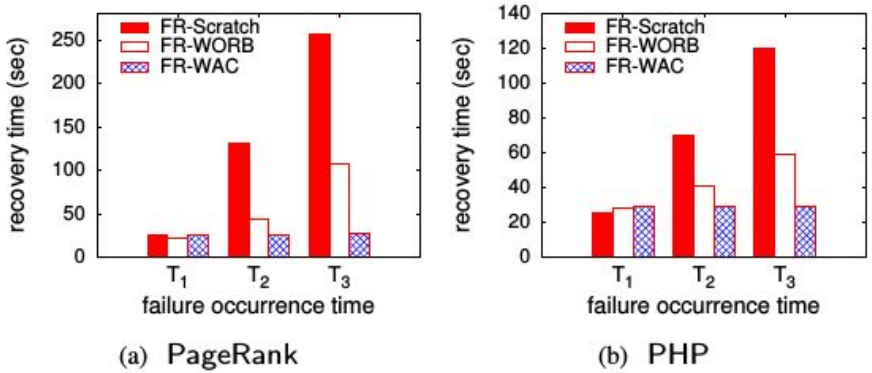


Figure 4. Recovery time on Orkut

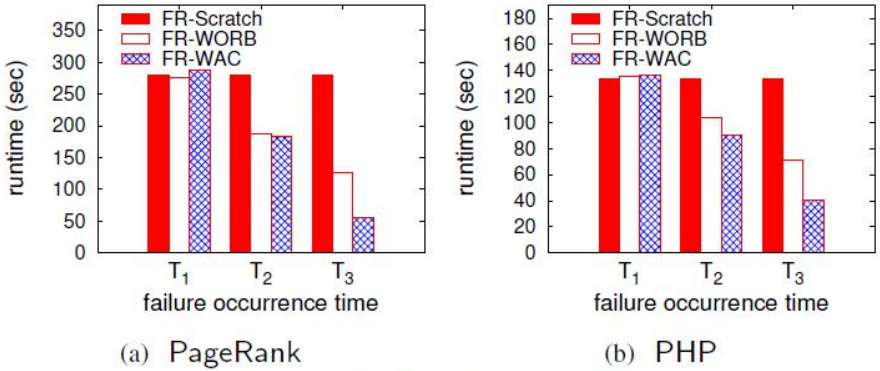
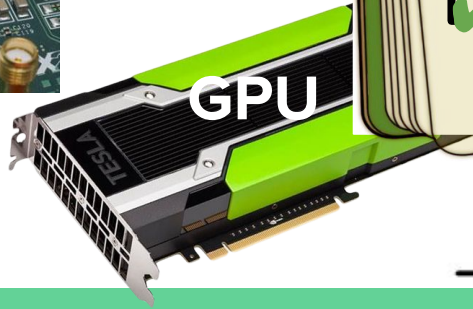
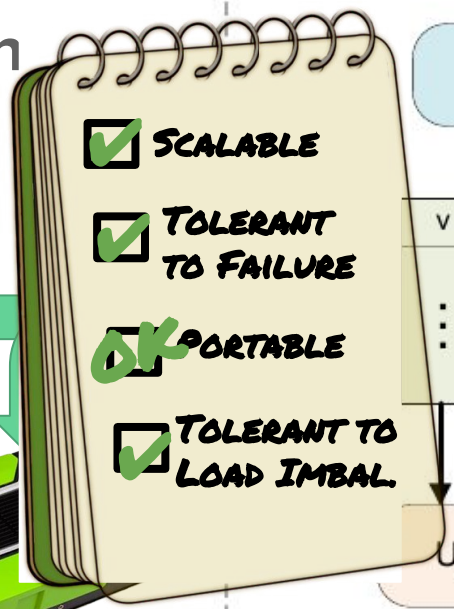


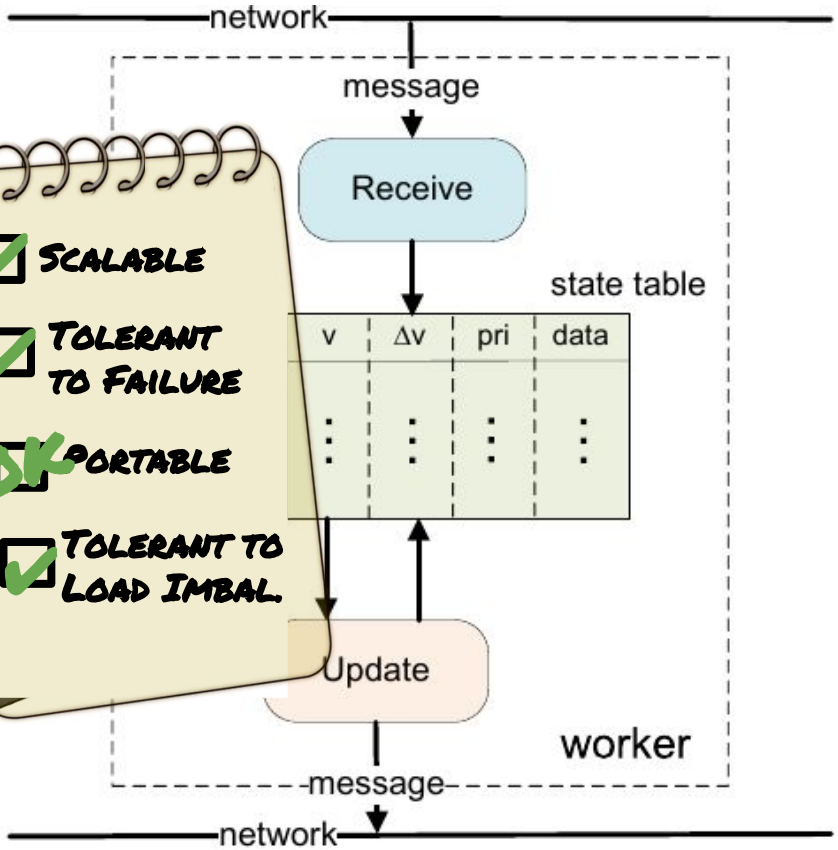
Figure 7. Runtime on Orkut

Portability

Independent computation based on vid







Conclusion

Maiter:

- **Define when asynchrony is allowed in delta-based graph processing**
- **Demonstrate performance and scalability benefits of asynchrony**

Faiter:

- **Identify weakness in asynchronous fault recovery**
- **Define when complete synchronous rollback is unnecessary**
- **Demonstrate performance benefits of efficient fault tolerance**

TensorFlow Discussion

- Can the data-parallel execution model be extended to other systems discussed in the course?
- How to handle reallocation of pre-empted nodes?
  - Load balancing not discussed
- Are stronger consistency guarantees worth the overhead?
  - Mutable state speeds up large scale model training, removes need for parameter server
- How to balance company vs. individual scale trade offs
  - Lacks some optimizations (i.e. hand-optimized kernels similar to Neon)
  - Fault tolerance - user dependent and checkpoint based
  - Few defaults - requires domain knowledge to create performant systems

Device Clarification

- GPUs, CPUs, other specialized hardware for training
- TPUs provide high performance/Watt for server side inference
- Cellphone GPUs aren’t incorporated into training systems. Rather they enable offline inference, i.e. offline translation

Faiter Discussion

- Single point of failure
- Fault Tolerance
  - FR-WORB vs FR-WAC
  - Is it really guaranteed to perform better than checkpointing? Should comparisons be included, perhaps with carefully placed failures in Faiter?
  - No comparison to other asynchronous checkpointing methods, e.g. the mentioned Chandy-Lamport method in GraphLab, nor Lineage recovery
- Scalable to real-world sized corporate clusters?
  - Tests run on t2.micro instances, these provide burst performance
- Unclear why the graph algorithms & datasets were chosen for evaluation

Faiter Discussion

- Can framework be applied to non-iterative graph algorithms? Or even stream processing?
- Are the distributed, communicative and associate property assumptions realistic for most desired computation?

Under what conditions are these systems not suitable for use?

Backup

Table 1: A list of DAIC algorithms

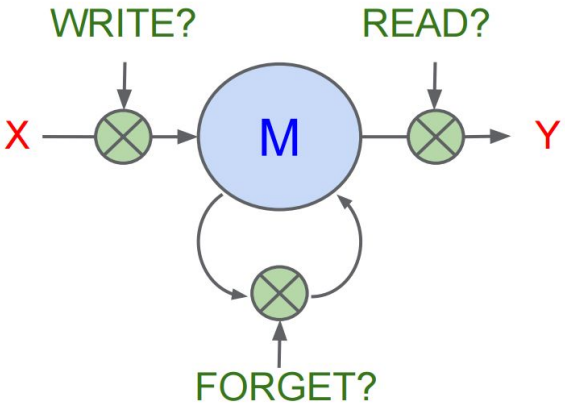
algorithm	$g_{\{i,j\}}(x)$	$\oplus$	v_j^0	Δv_j^1
SSSP	$x + A(i,j)$	min	∞	$0\ (j = s)$ or $\infty\ (j \neq s)$
Connected Components	$A(i,j) \cdot x$	max	-1	j
PageRank	$d \cdot A(i,j) \cdot \frac{x}{\lvert N(j)\rvert}$	$+$	0	$1 - d$
Adsorption	$p_i^{cont} \cdot A(i,j) \cdot x$	$+$	0	$p_j^{inj} \cdot I_j$
HITS (*authority*)	$d \cdot A(i,j) \cdot x$	$+$	0	1
Katz metric	$\beta \cdot A(i,j) \cdot x$	$+$	0	$1\ (j = s)$ or $0\ (j \neq s)$
Jacobi method	$-\frac{A_{ji}}{A_{jj}} \cdot x$	$+$	0	$\frac{b_j}{A_{jj}}$
SimRank	$\frac{C \cdot A(i,j)}{\lvert I(a)\rvert\lvert I(b)\rvert} \cdot x$	$+$	$\lvert I(a) \cap I(b)\rvert\ (a \neq b)$ or $1\ (a = b)$	$\frac{\lvert I(a)\rvert\lvert I(b)\rvert}{C}\ (a \neq b)$ or $0\ (a = b)$
Rooted PageRank	$A(j,i) \cdot x$	$+$	0	$1\ (j = s)$ or $0\ (j \neq s)$

Table 1. Example Graph Algorithms

Algorithms	c_j	$g_{\{i,j\}}(x)$	$\oplus$	$\ominus$	**0**
PageRank	(1-d)	$d \cdot \frac{x}{\lvert\Gamma^{out}(j)\rvert}$	$+$	$-$	0
PHP	1 (j=s) or 0 ($j \neq s$)	d$\cdot x \cdot w(i,j)$ ($j \neq s$) or 0 (j=s)	$+$	$-$	0
Katz	1 (j=s) or 0 ($j \neq s$)	$\beta \cdot x$	$+$	$-$	0
Adsorption	$p_j^{inj} \cdot I_j$	$p_j^{cont} \cdot w(i,j) \cdot x$	$+$	$-$	0

Dynamic Control Flow

Long Short-Term Memory (LSTMs):
Make Your Memory Cells Differentiable
[Hochreiter & Schmidhuber, 1997]



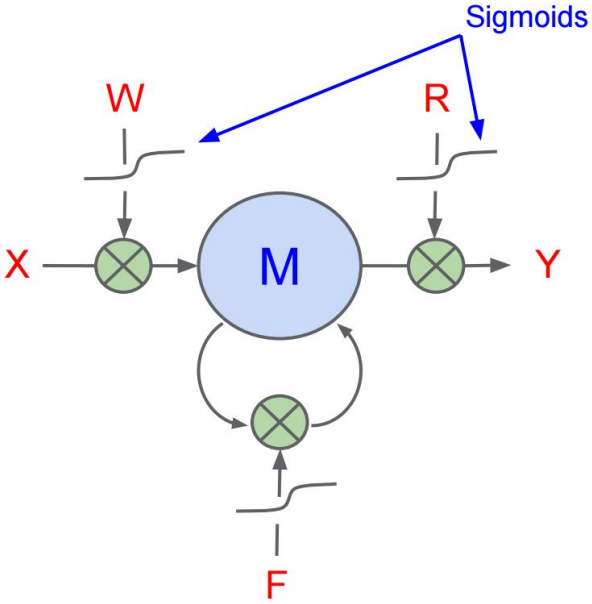


Image sources

Graphs source: http://www.cise.ufl.edu/research/sparse/matrices/

CNN source: http://parse.ele.tue.nl/education/cluster2

Brain img source: https://www.docuvo.com/whatwedo.html

Social net source: https://www.quora.com/How-do-social-networking-sites-and-search-engines-use-the-concept-of-graph-Please-use-layman-terms

GoogLeNet source: Christian Szegedy et al,"Going Deeper with Convolutions", CVPR2015

Notepad source: https://clipartfest.com/categories/view/bf3ad8f22b3d818eee77c01504f976009ecefebf/clipart-notepad.html

FPGA source: Adrian Caulfield et al. "A Cloud-Scale acceleration architecture", MICRO 2016

Blue waters source: http://www.cray.com/enabling-scientific-breakthroughs-petascale

2.3 Related work

1. Single Machine Frameworks
    a. Theano, Caffe, Torch
2. Batch Dataflow Frameworks
    a. MapReduce, Spark
3. Parameter Server Architectures
    a. DistBelief, Project Adam, MXNet