Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

PETUUM

System & algorithm co-design for distributed machine learning: theory and practice

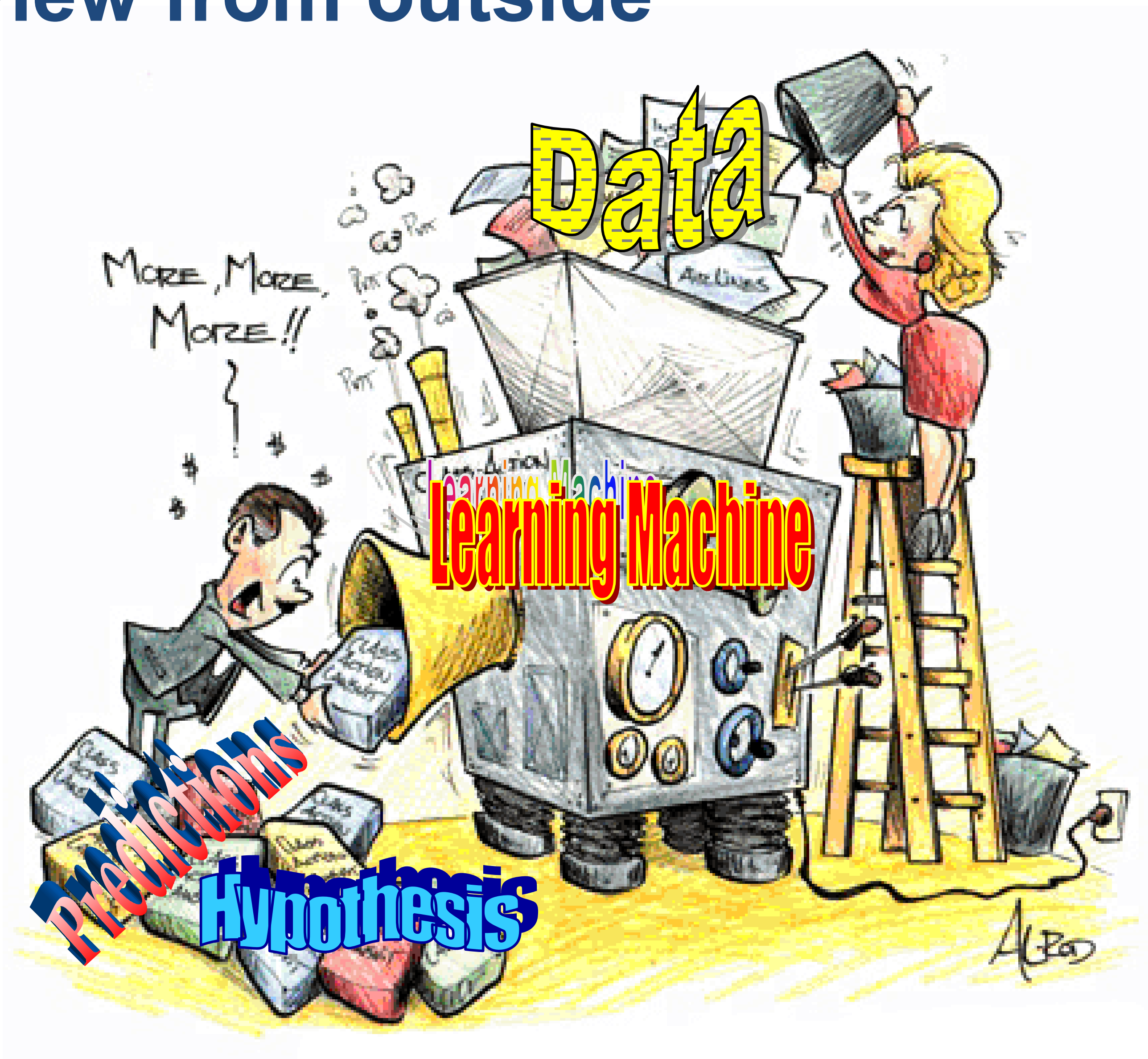
Eric Xing

epxing@cs.cmu.edu

School of Computer Science
Carnegie Mellon University

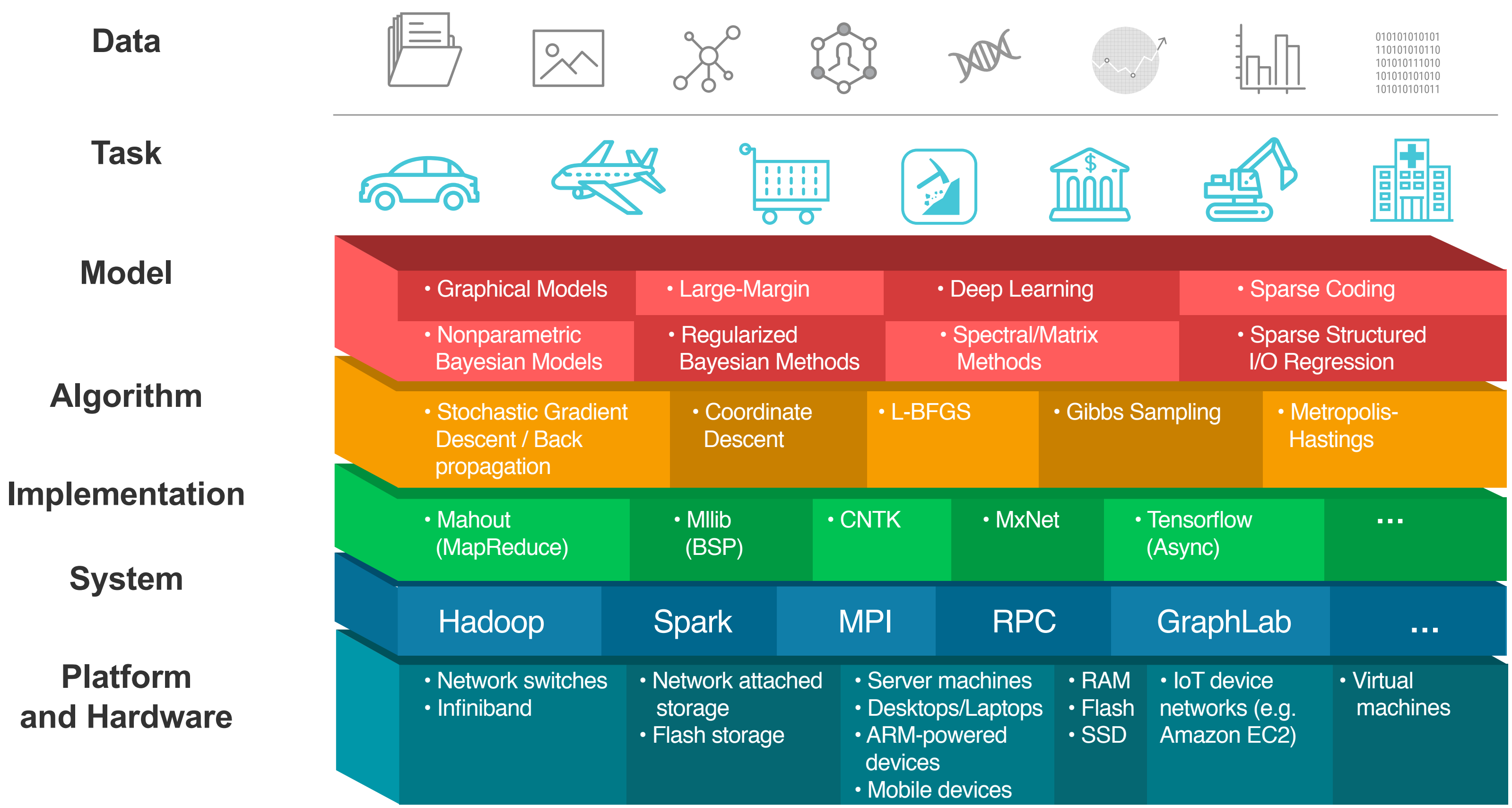
Acknowledgement:
Wei Dai, Qirong Ho, Jin Kyu Kim, Abhimanu Kumar, Seunghak Lee, Jinliang Wei, Pengtao Xie, Yaoliang Yu, Hao Zhang, Xun Zheng
James Cipar, Henggang Cui,
and, Phil Gibbons, Greg Ganger, Garth Gibson

Machine Learning:
-- a view from outside



SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Elements of Modern AI/ML

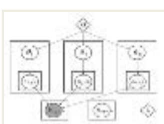


SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

From 1m to 100m Events (and more)

1m users

1 machine



Scaling up AI/ML programs: from workstation to production cluster

1-machine prototype, state-of-the-art code
=> supports 1m users in 6min

Want to run code on 100m users, in real-time
*=> 100m users = 100 * 1m users*

So if using 1000 Hadoop machines…
=> *should* support 100m users in 0.6min!

In fact, took >1 week to finish!

An ML Program

$$\arg\max_{\vec{\theta}} \equiv \mathcal{L}(\{\mathbf{x}_i, \mathbf{y}i\}_{i=1}^{N} \; ; \; \vec{\theta}) + \Omega(\vec{\theta})$$

Model **Data** **Parameter**

Solved by an iterative convergent algorithm

```
for (t = 1 to T) {
  doThings()
```

$$\vec{\theta}^{t+1} = g(\vec{\theta}^{t}, \; \underline{\Delta_f \vec{\theta}(\mathcal{D})})$$

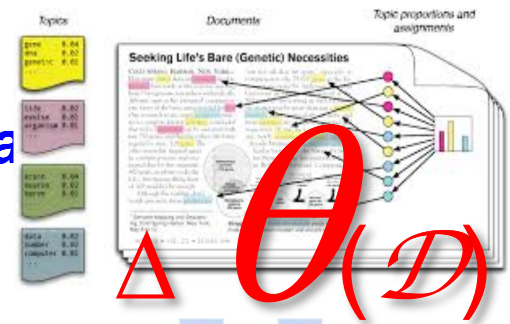
```
  doOtherThings()
}
```

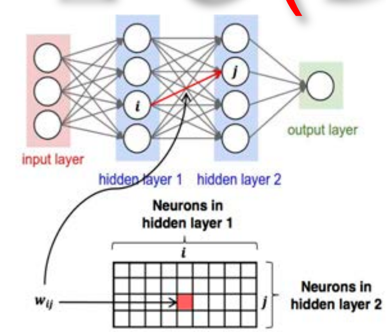
This computation needs to be parallelized!

Some Trends in AI & ML

Larger AI & ML Models are Better for Big Data

- Text Extraction: 1B to 1T params
- Deep Learning: 1B+ params
- Rec. Systems: 10M to 100M params
- **Today's Model Sizes: >GBs**





Efficiency & Correctness

- Need distributed computing
- Need to sync across cluster!

Hadoop, Spark use joins (e.g. RDD join) to sync

- Parameter shuffle takes >90% of execution time

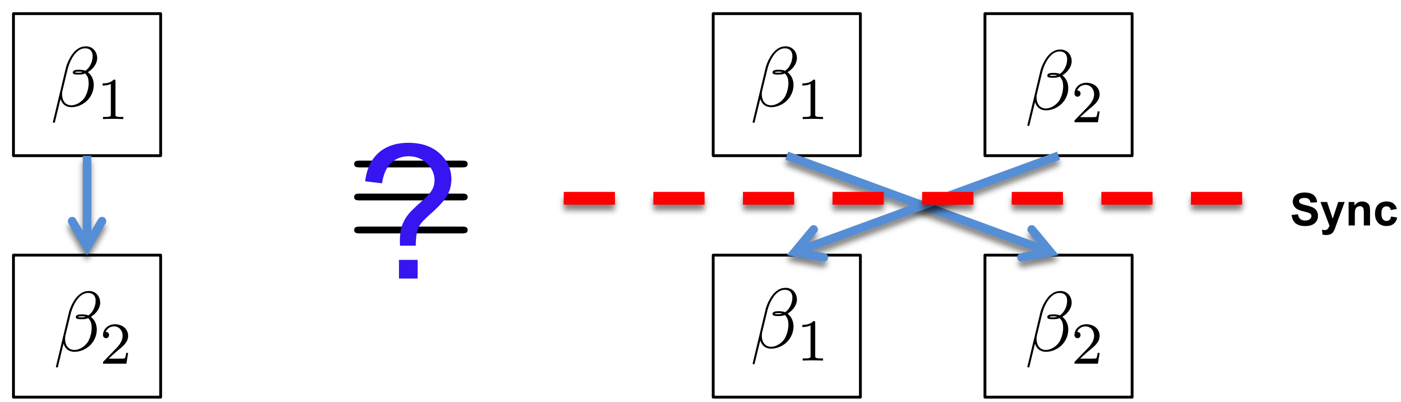
SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Parallelization Strategy

```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```

Usually, worry …

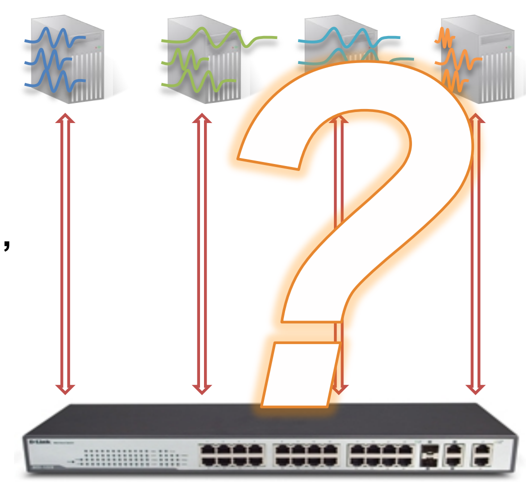
A sequential program **A parallel program**

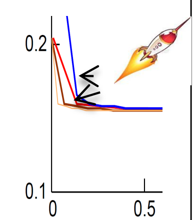


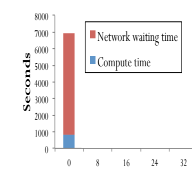
- but assuming an ideal system, e.g.,
  - zero-cost sync,
  - zero-cost fault recovery
  - uniform local progress
  - …

Unequal performance

Low bandwidth, High delay







Analysis of Efficiency …

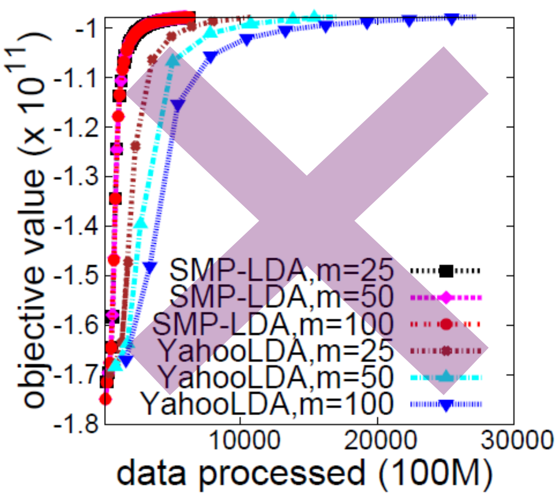
- Statistical, computation, data, optimization …

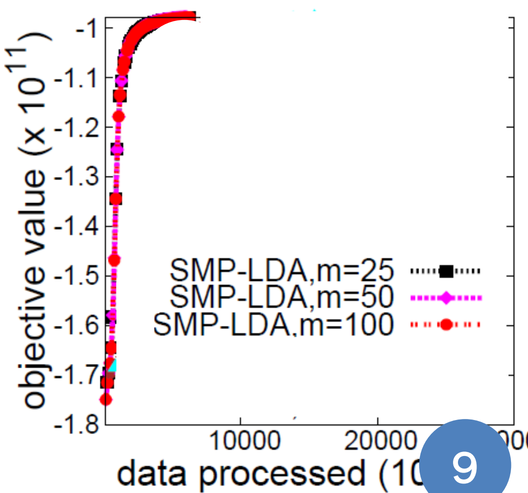
- A typical algorithmic behavioral analysis

$$(\ell + r)(\mathbf{w}^t) - (\ell + r)(\mathbf{w}) \leq \frac{\|\mathbf{w}^0 - \mathbf{w}\|^2}{2\eta t}$$

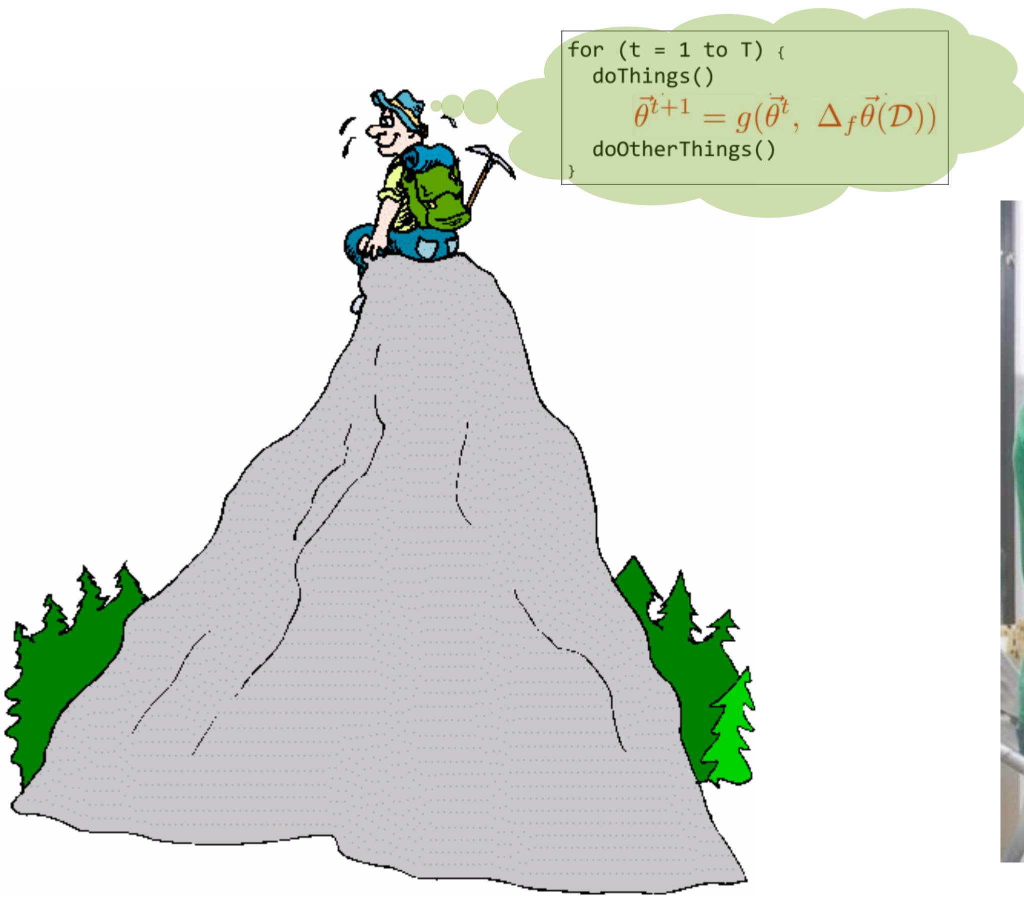
- A distributed implementation:

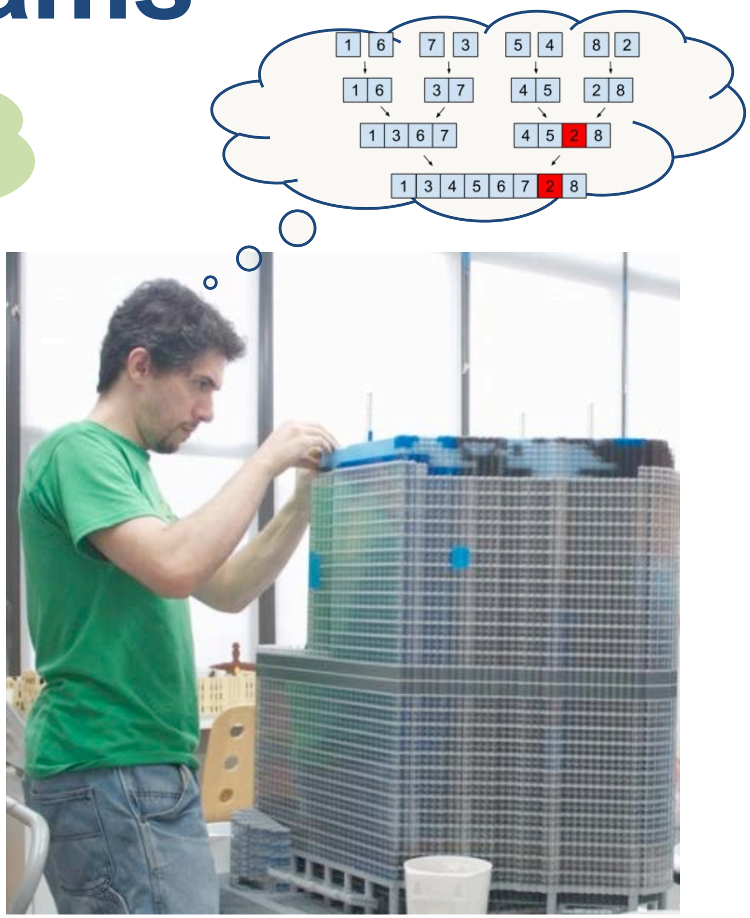
	YahooLDA data throughput
25 machines	39.7 M/s (1x)
50 machines	78 M/s (1.96x)
100 machines	151 M/s (3.8x)





ML Computation vs. Classical Computing Programs





ML Program: optimization-centric and iterative convergent

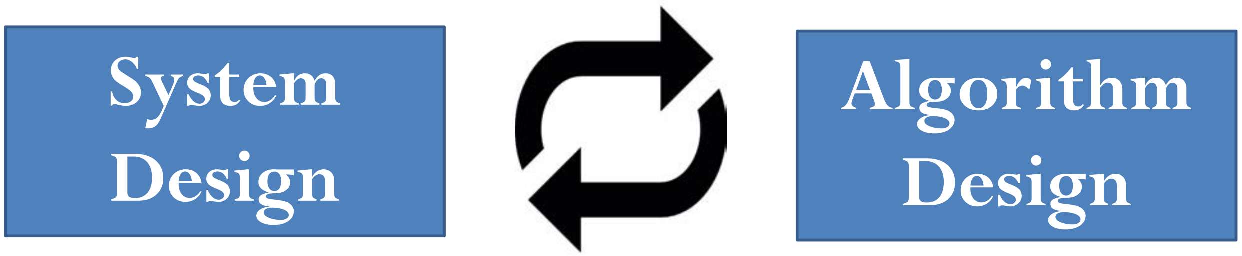
Traditional Program: operation-centric and deterministic

Properties of ML Programs [Xing et al., 2015]

- ML is ... gent algori... on
  - Err... erro...
  - Dyr... cha... criti...
  - Nor... can...

- Wher... only guara...
- How do design optimal architectures fit for the above?

System/Algorithm Co-design



- System design should be tailored to the unique mathematical properties of ML algorithms
- Algorithms can be re-designed to better exploit the system architectures

Toward a General Purpose Architecture via sys/alg co-design

ML program equations tell us “What to Compute”.

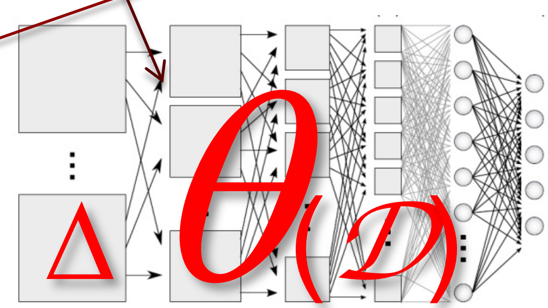
$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

But…

1. **How to Distribute?**
2. **How to Bridge Computation and Communication?**
3. **What to Communicate?**
4. **How to Communicate?**

Data- and Model-Parallel ML Programs

$$\vec{\theta}^{t+1} = \vec{\theta}^{t} + \Delta_f \vec{\theta}(\mathcal{D})$$

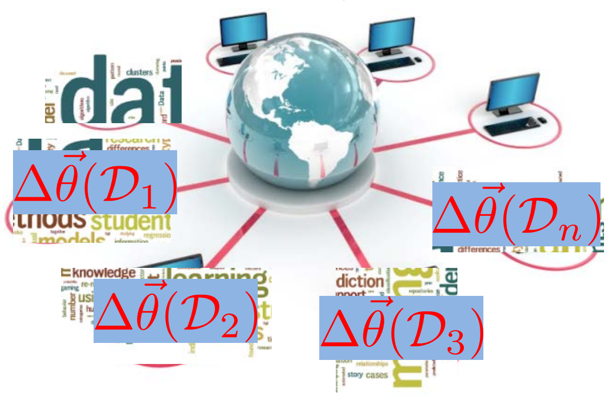


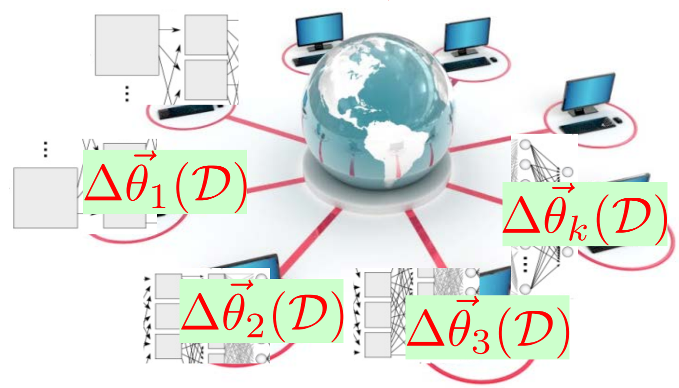
$\mathcal{D} \equiv \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$

$\vec{\theta} \equiv [\vec{\theta}_1^{\mathrm{T}}, \vec{\theta}_2^{\mathrm{T}}, \ldots, \vec{\theta}_k^{\mathrm{T}}]^{\mathrm{T}}$

Data Parallel

Model Parallel





$\mathcal{D}_i \perp \mathcal{D}_j \mid \theta, \ \forall i \neq j$

$\vec{\theta}_i \not\perp \vec{\theta}_j \mid \mathcal{D}, \ \exists (i,j)$

System/Algorithm Co-design

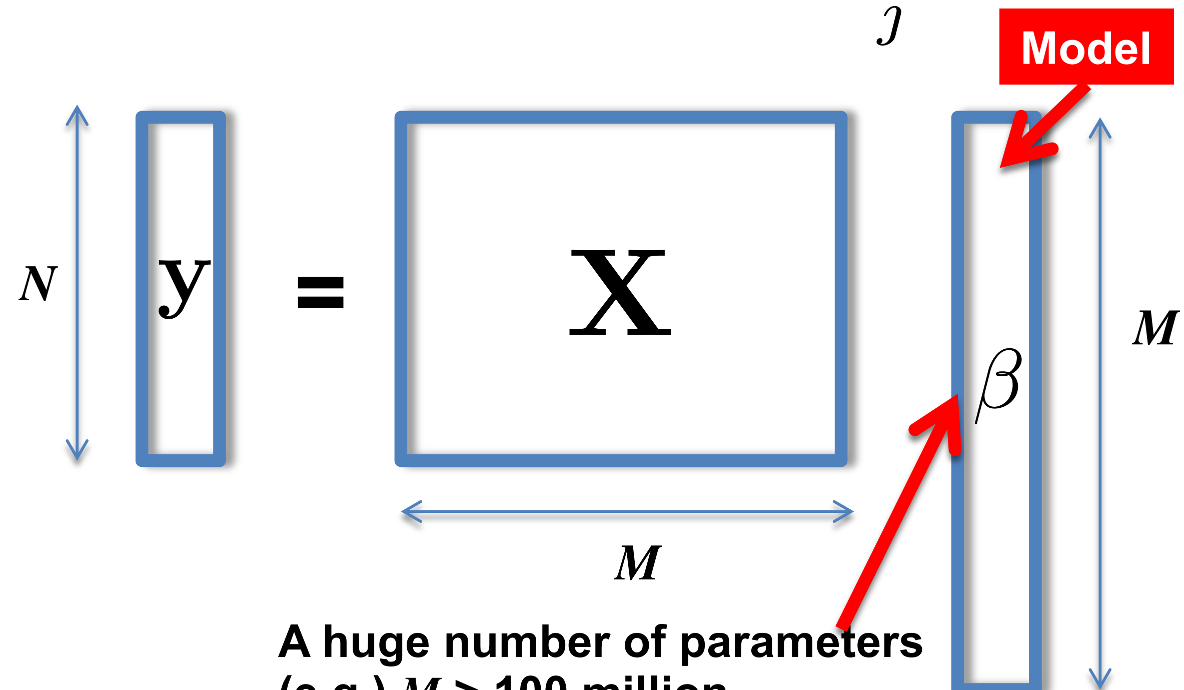
1. How to Distribute:

Scheduling and Balancing workloads

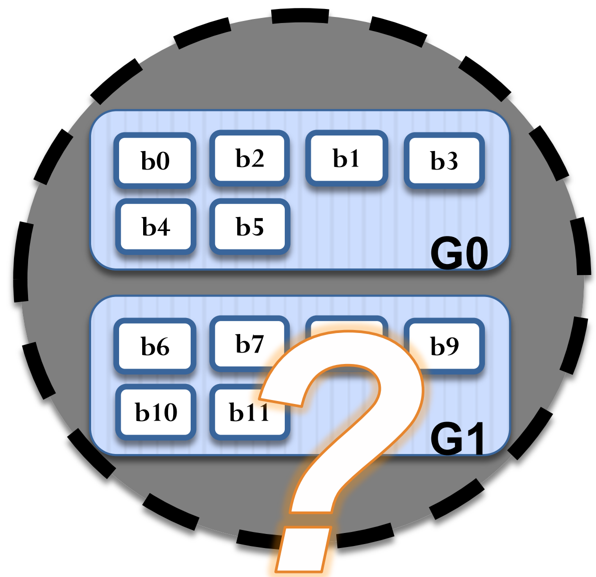
Example: Model Distribution

Lasso via coordinate descent:

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_j |\beta_j|$$



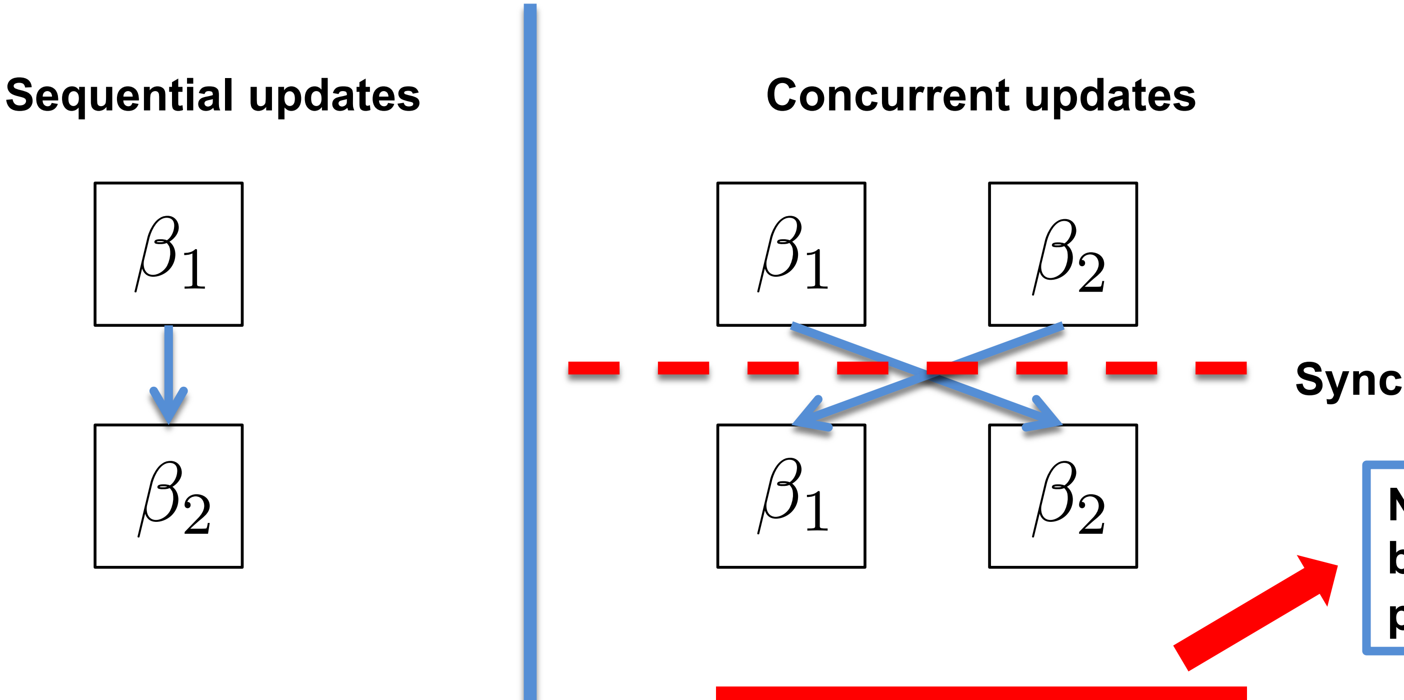
A huge number of parameters
(e.g.) M > 100 million



- How to correctly divide computational workload across workers?
- What is the best order to update parameters?

Model Dependencies

- Concurrent updates of β may induce errors

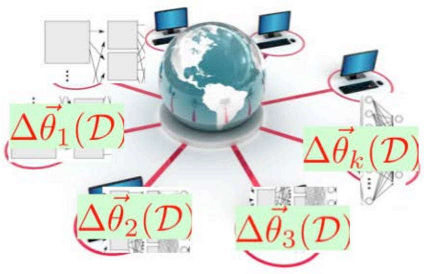


Need to check $\mathbf{x}_1^T\mathbf{x}_2$ before updating parameters

Decreases iteration progress

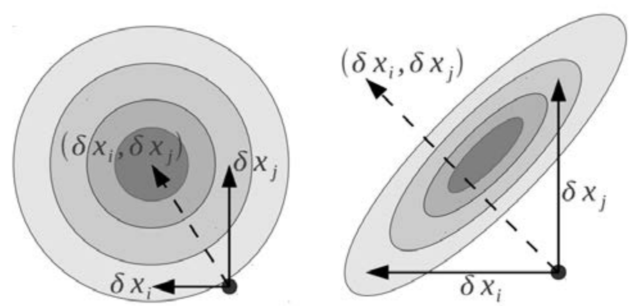
$$\beta_1^{(t)} \leftarrow S(\mathbf{x}_1^T\mathbf{y} - \boxed{\mathbf{x}_1^T\mathbf{x}_2\beta_2^{(t-1)}}, \lambda)$$

Parallel Coordinate Desce



[Bradley et al. 2011]

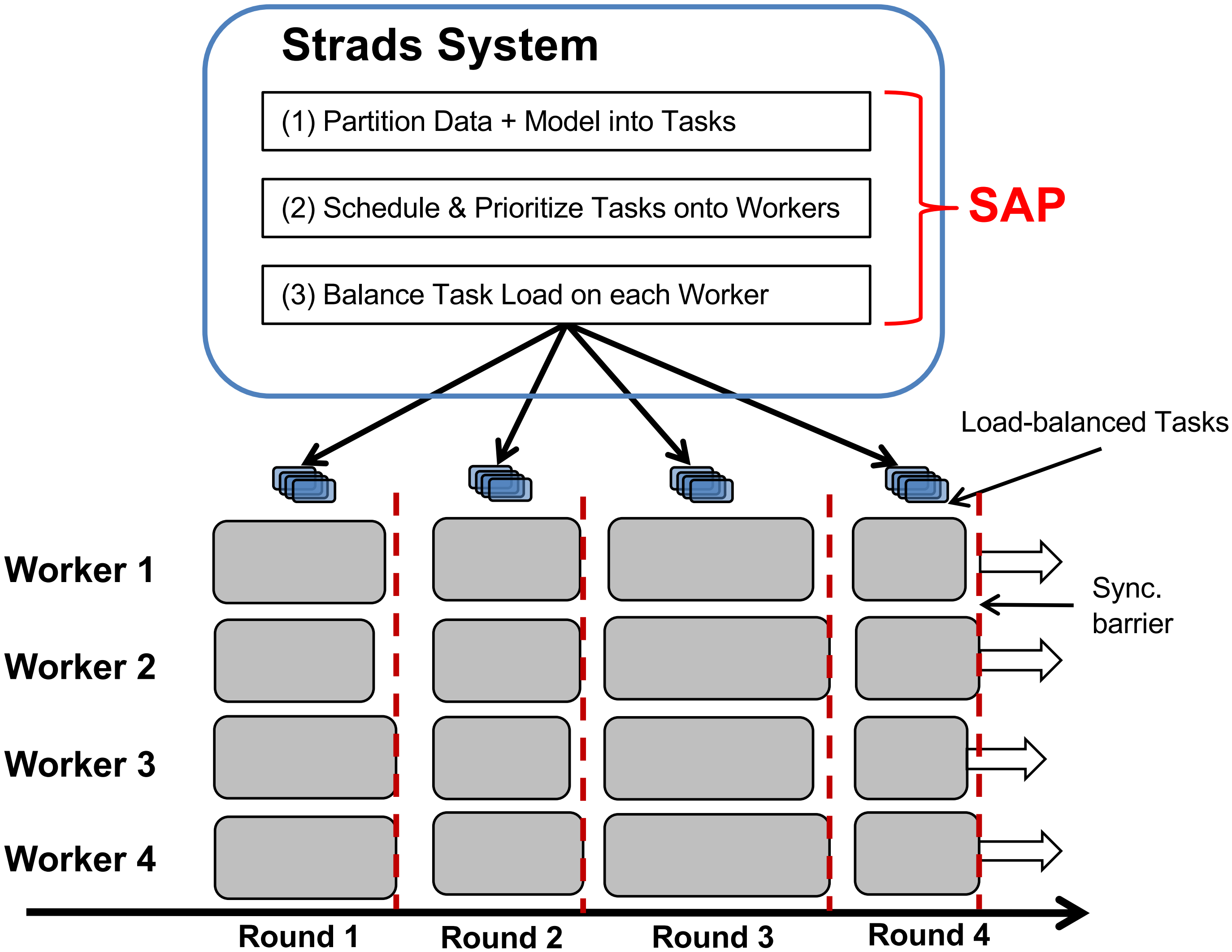
- Shotgun, a parallel coordinate descent algorithm
  - Choose parameters to update at random
  - Update the selected parameters in parallel
  - Iterate until convergence
- When features are nearly independent, Shotgun scales almost linearly
  - Shotgun scales linearly up to $P \leq \frac{d}{2\rho}$ workers, where ρ is spectral radius of A^TA
  - For uncorrelated features, ρ=1; for exactly correlated features ρ=d
  - No parallelism if features are exactly correlated!



Uncorrelated features **Correlated features**

Source:
[Bradley et al., 2011]

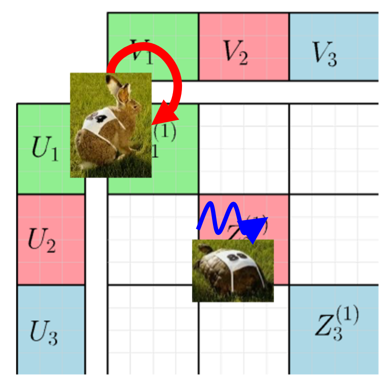
A Structure-aware Dynamic Scheduler (Strads) [Lee et al., 2014] [Kim et al, 2016]



- **Priority Scheduling**

$$\{\beta_j\} \sim \left(\delta\beta_j^{(t-1)}\right)^2 + \eta$$

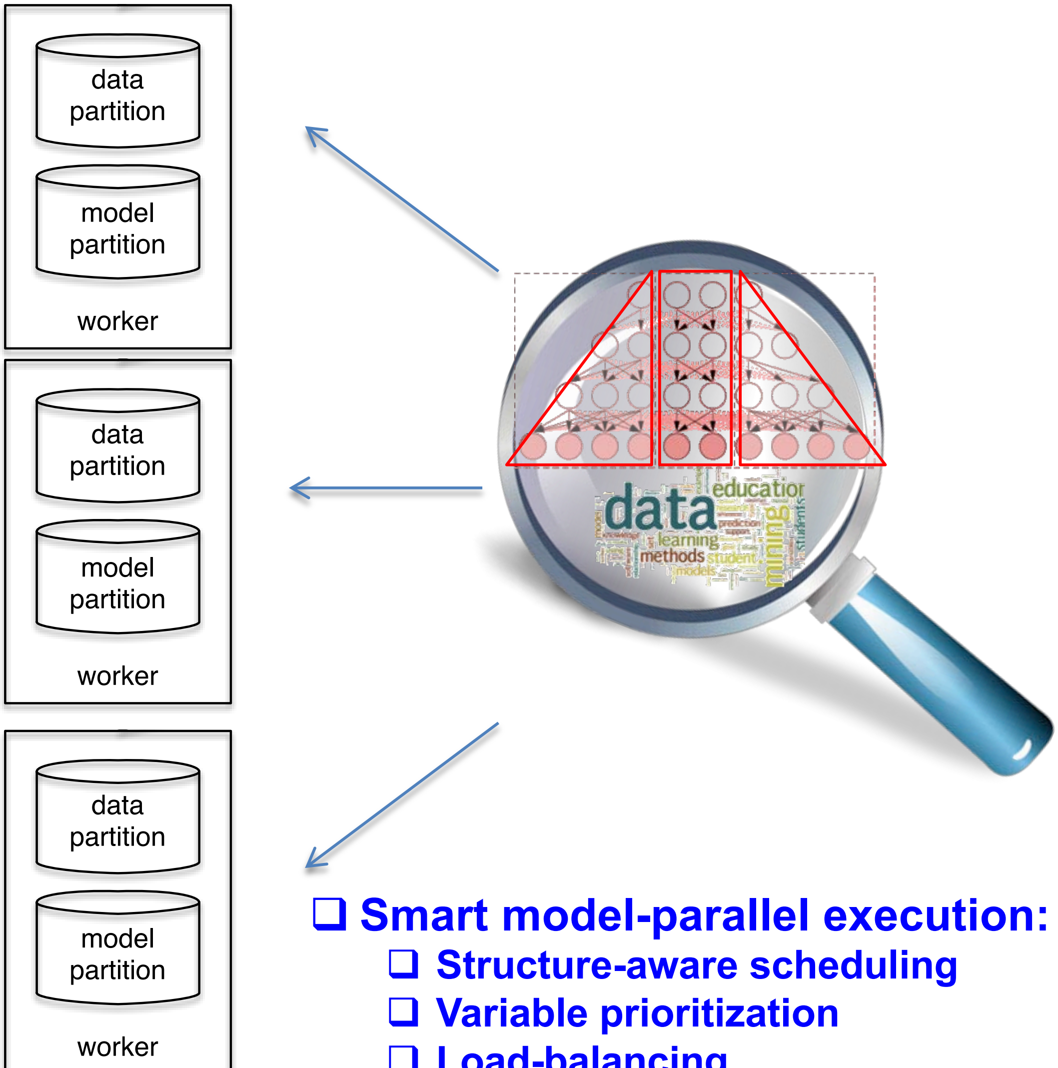
- **Block scheduling**



[*Kumar, Beutel, Ho and Xing*, ***Fugue: Slow-worker agnostic distributed learning***, *AISTATS 2014*]

Avoid Dependency Errors via Structure-Aware Parallelization (SAP)

[Lee et al., 2014] [Kim et al, 2016]




```
schedule() {
  // Select U vars x[j] to be sent
  // to the workers for updating
  ...
  return (x[j_1], ..., x[j_U])
}
```

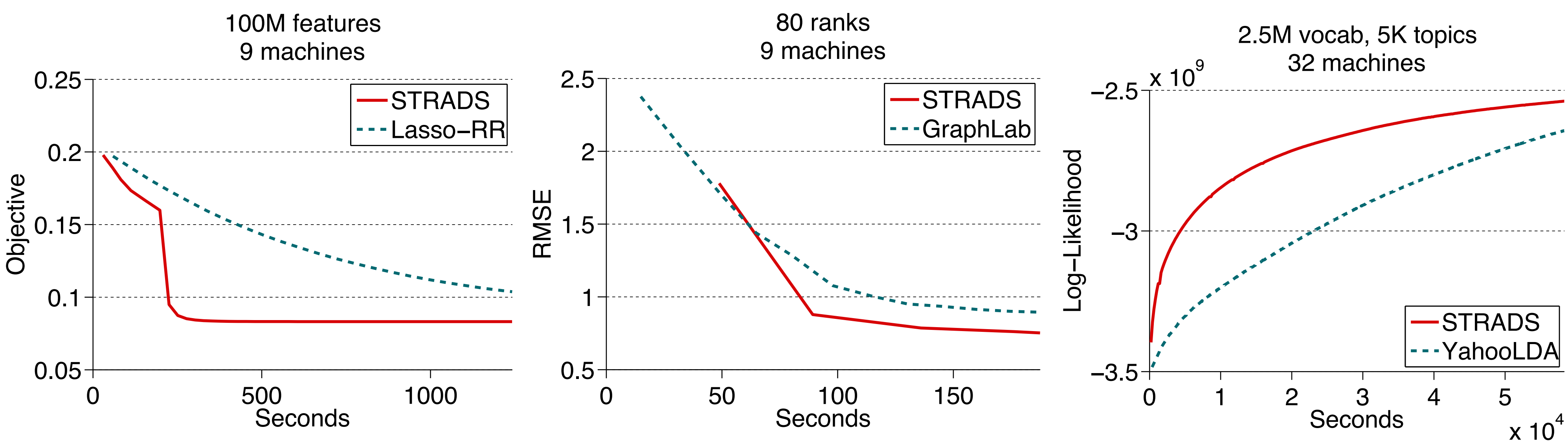
```
push(worker = p, vars = (x[j_1],...,x[j_U])) {
  // Compute partial update z for U vars x[j]
  // at worker p
  ...
  return z
}
```

```
pull(workers = [p], vars = (x[j_1],...,x[j_U]),
     updates = [z]) {
  // Use partial updates z from workers p to
  // update U vars x[j]. sync() is automatic.
  ...
}
```

- ❑ **Smart model-parallel execution:**
  - ❑ **Structure-aware scheduling**
  - ❑ **Variable prioritization**
  - ❑ **Load-balancing**
- ❑ **Simple programming:**
  - ❑ **Schedule()**
  - ❑ **Push()**
  - ❑ **Pull()**

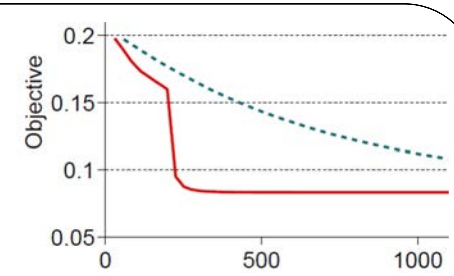
SAP Scheduling: Faster, Better Convergence across algorithms

- SAP on Strads achieves better speed and objective



Lasso **MF** **LDA**

SAP gives Near-Ideal Convergence Speed [Xing et al., 2015]



- **Goal:** solve sparse regression problem
  - Via coordinate descent over "SAP blocks" $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$
    - $X^{(b)}$ are data columns (features) in block (b)
  - P parallel workers, M-dimensional data
  - ρ = Spectral Radius[BlockDiag[$(X^{(1)})^T X^{(1)}, \ldots, (X^{(t)})^T X^{(t)}$]]; this block-diagonal matrix quantifies max level of correlation within all SAP blocks $X^{(1)}, X^{(2)}, \ldots, X^{(t)}$

- **SAP converges according to**

$$\mathbb{E}\left[\overbrace{f(X^{(t)}) - f(X^*)}^{\text{Gap between current parameter estimate and optimum}}\right] \leq \frac{\overbrace{\mathcal{O}(M)}^{\text{SAP explicitly minimizes } \rho\text{, ensuring as close to } 1/P \text{ convergence as possible}}}{P - \boxed{\frac{\mathcal{O}(P^2\rho)}{M}}} \frac{1}{t} = \mathcal{O}\left(\frac{1}{Pt}\right)$$

where t is # of iterations

- **Take-away:** SAP minimizes ρ by searching for feature subsets $X^{(1)}, X^{(2)}, \ldots, X^{(B)}$ w/o cross-correlation => as close to P-fold speedup as possible

System/Algorithm Co-design

2. How to Bridge Computation and Communication:
Bridging Models and Bounded Asynchrony

Data-Parallel Proximal Gradient under SSP

- Model (e.g. SVM, Lasso …):

$$\min_{\mathbf{a}\in\mathbb{R}^d} \mathcal{L}(\boxed{\mathbf{a}}, \boxed{D}), \quad \text{where } \mathcal{L}(\mathbf{a}, D) = f(\mathbf{a}, D) + g(\mathbf{a})$$

data D, model a

- Algorithm:
  - Update

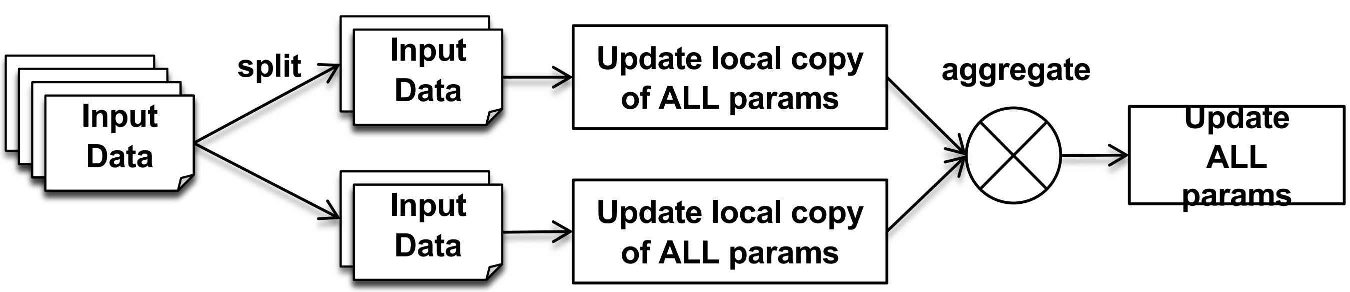
$$\mathbf{a}(t) := \underbrace{\mathrm{prox}_g}_{\text{proximal step wrt } g}\left(\mathbf{a}^p(t) - \eta(t)\sum_{(p',t')\in \boxed{Recv^p(t)}} \overbrace{\Delta(\mathbf{a}^{p'}(t'), D_{p'})}^{\text{sub-update}}\right)$$

stale sub-updates Δ() received by worker *p* at iteration *t*

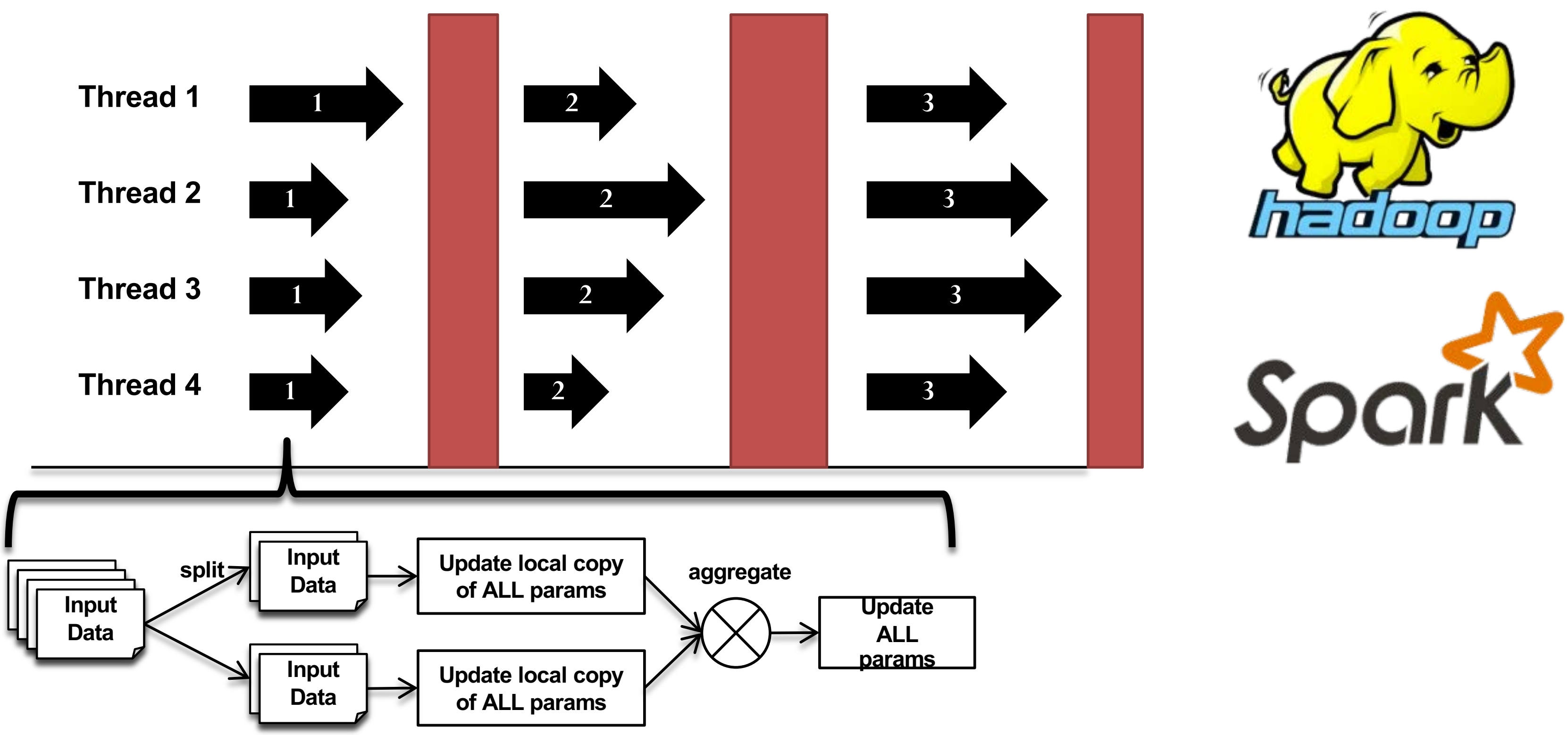
  - sub-update

$$\Delta(\mathbf{a}^p(t), D_p) := \underbrace{\nabla f(\mathbf{a}^p(t), D_p)}_{\text{gradient step wrt } f}$$

- Data parallel:
  - Data D too large to fit in a single worker, divide among P workers

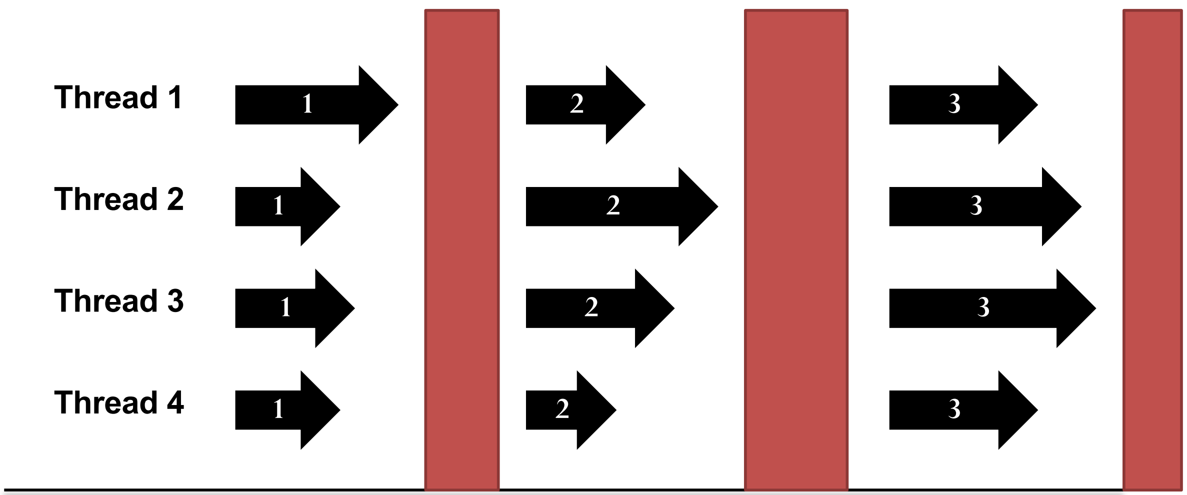


The *Bulk Synchronous Parallel* Bridging Model [Valiant & McColl]



- Perform barrier in order to communicate parameters
- Mimics sequential computation – "serializable" property
- Enjoys same theoretical guarantees as sequential execution

The *Bulk Synchronous Parallel* Bridging Model [Valiant & McColl]



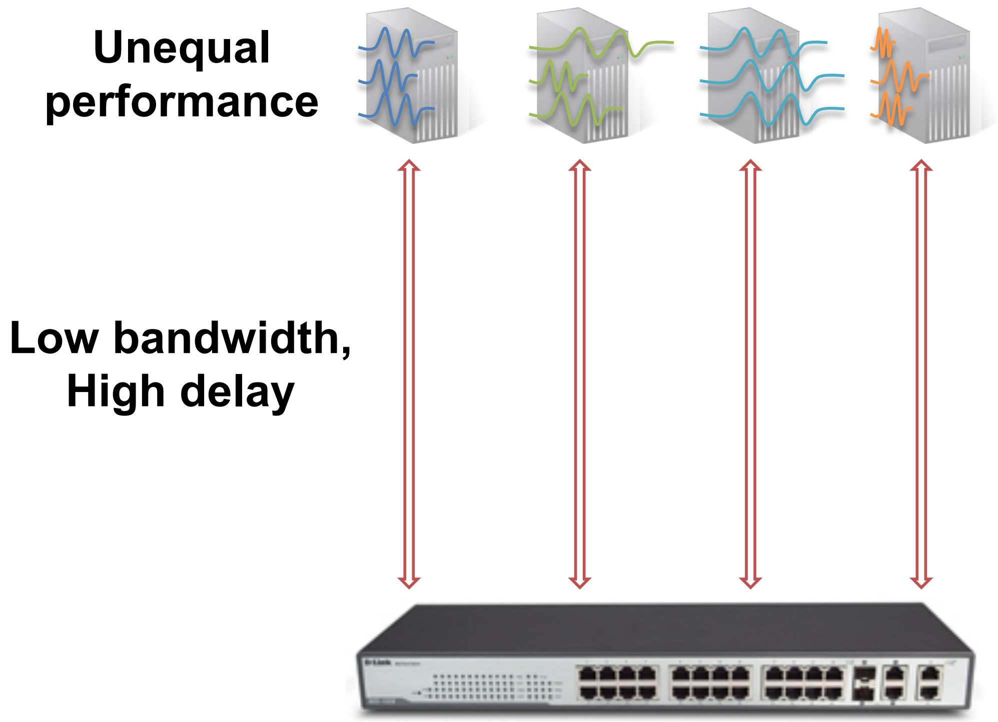
The success of the von Neumann model of sequential computation is attributable to the fact it is an efficient bridge between software and hardware… an analogous bridge is required for parallel computation if that is to become as widely used – **Leslie G. Valiant**

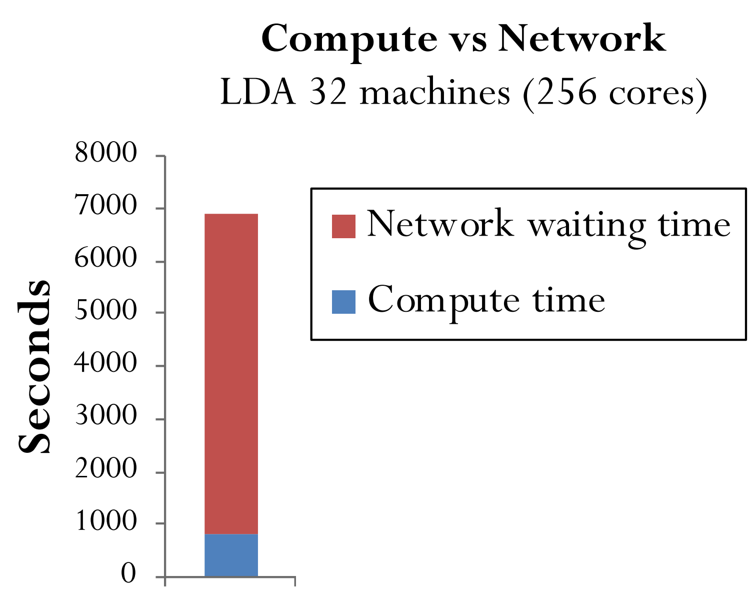
- Numerous implementations since 90s (list by **Bill McColl**):
  - Oxford BSP Toolset ('98), Paderborn University BSP Library ('01), Bulk Synchronous Parallel ML ('03), BSPonMPI ('06), ScientificPython ('07), Apache Hama ('08), Apache Pregel ('09), MulticoreBSP ('11), BSPedupack ('11), Apache Giraph ('11), GoldenOrb ('11), Stanford GPS Project ('11) …

But There Is No Ideal Distributed System!

- **Two distributed challenges:**
  - Networks are slow
  - “Identical” machines rarely perform equally

Result: BSP barriers can be slow





Hogwild! Algorithm

- Hogwild! algorithm: iterate in parallel for each core
  - Sample e uniformly at random from E
  - Read current parameter x_e; evaluate gradient of function f_e
  - Sample uniformly at random a coordinate v from subset e
  - Perform SGD on coordinate v with small constant step size

- Atomically update single coordinate, no mem-locking
- **Hogwild! takes advantage of sparsity in ML problems**
- Enables near-linear speedup on various ML problems
- **Excellent on single machine, less ideal for distributed**
  - Atomic update on multi-machine challenging to implement; inefficient and slow
  - Delay among machines requires explicit control… why? (see next slide)

The cost of uncontrolled delay – slower convergence [Dai et al. 2015]

- Theorem: Given lipschitz objective f_t and step size η_t,

$$P\left[\frac{R[X]}{T} - \frac{1}{\sqrt{T}}\left(\sigma L^2 + \frac{F^2}{\sigma} + 2\sigma L^2 \epsilon_m\right) \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{2\bar{\sigma}_T \boxed{\epsilon_v} + \frac{2}{3}\sigma L^2 (2s+1) P\tau}\right\}$$

  - where $R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*)$
  - Where L is a lipschitz constant, and ϵ_m and ϵ_v are the mean and variance of the delay

- Intuition: distance between current estimate and optimal value decreases exponentially with more iterations
  - But high variance in the delay ϵ_v incurs exponential penalty!
- Distributed systems exhibit much higher delay variance, compared to single machine

The cost of uncontrolled delay – unstable convergence [Dai et al. 2015]

- Theorem: the variance in the parameter estimate is

$$\mathrm{Var}_{t+1} = \mathrm{Var}_t - 2\eta_t cov(\boldsymbol{x}_t, \mathbb{E}^{\Delta_t}[\boldsymbol{g}_t]) + \mathcal{O}(\eta_t \xi_t) + \mathcal{O}(\eta_t^2 \rho_t^2) + \boxed{\mathcal{O}^*_{\epsilon_t}}$$

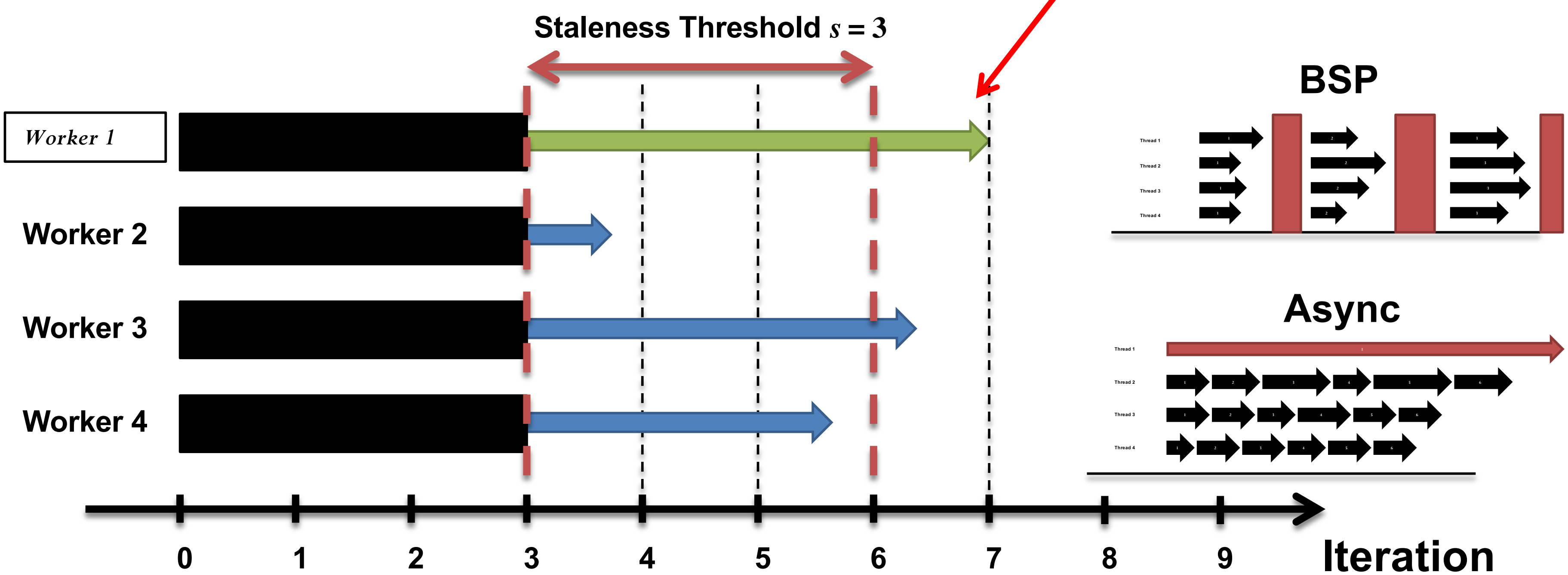
  - Where $cov(\boldsymbol{v}_1, \boldsymbol{v}_2) := \mathbb{E}[\boldsymbol{v}_1^T \boldsymbol{v}_2] - \mathbb{E}[\boldsymbol{v}_1^T]\mathbb{E}[\boldsymbol{v}_2]$
  - and $\mathcal{O}^*_{\epsilon_t}$ represents 5th order or higher terms, as a function of the delay ε_t

- Intuition: variance of the parameter estimate decreases near the optimum
  - But delay ε_t increases parameter variance => instability during convergence
- Distributed systems have much higher average delay, compared to single machine

Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

A Stale Synchronous Parallel Bridging Model [Ho et al., 2013]

Force stop worker 1 until worker 2 catches up



Stale Synchronous Parallel (SSP)

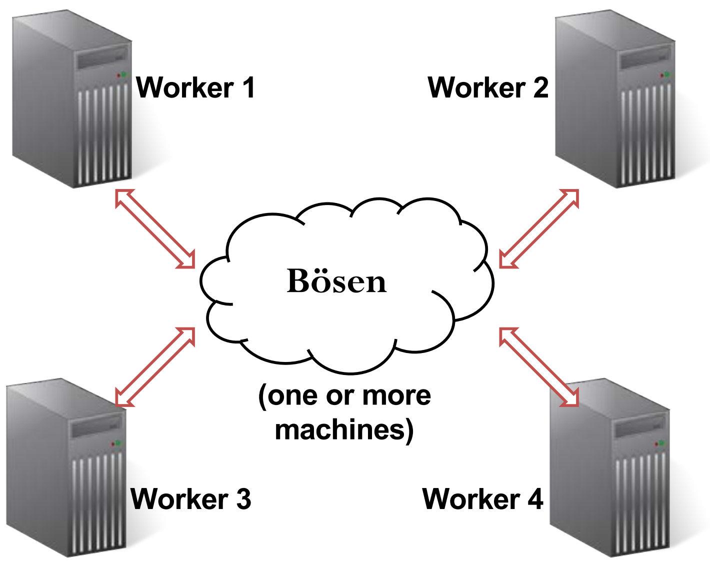
- Fastest/slowest workers not allowed to drift $>s$ iterations apart

Consequence

- Fast like async, yet correct like BSP
- Why? Workers' local view of model parameters "not too stale" ($\leq s$ iterations old)

Parameter Server Architecture

- Bösen: a bounded-asynchronous distributed key-value store
  - Data-parallel programming via distributed shared memory (DSM) abstraction
  - Managed communication for better parallel efficiency & guaranteed convergence

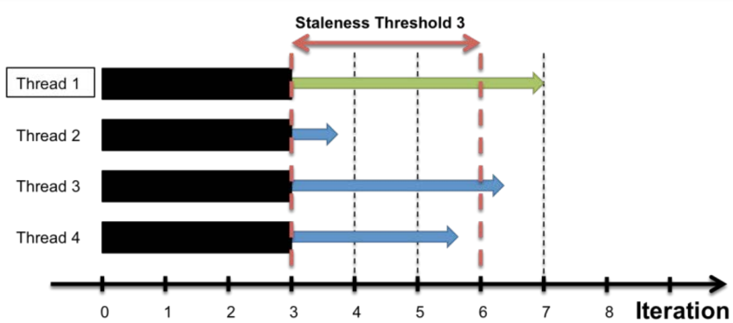


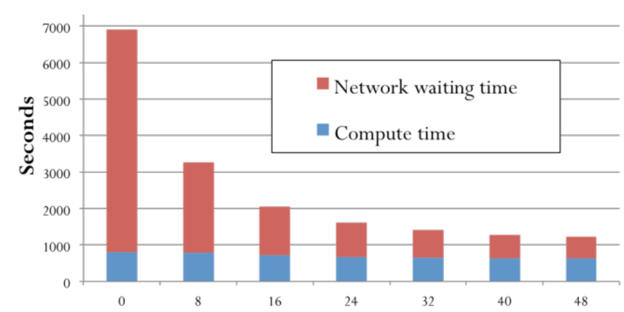
Single Machine Parallel

```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

Distributed with Bösen

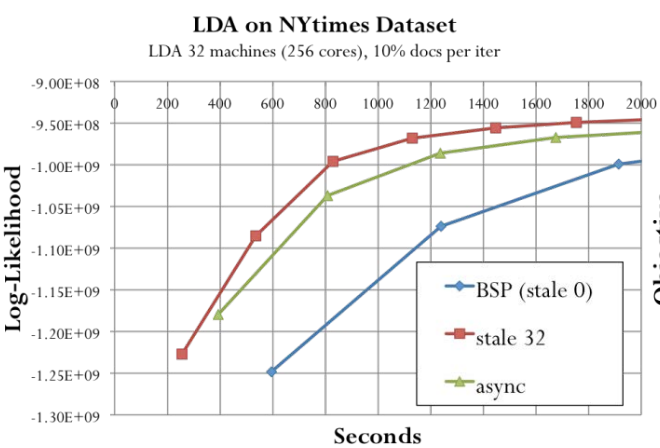
```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```



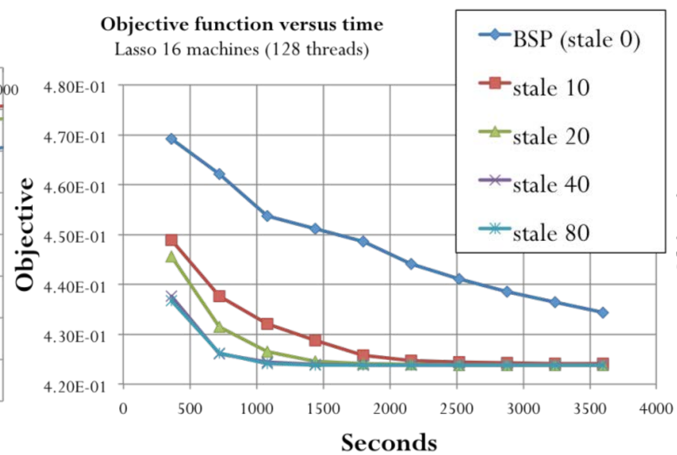


SSP Data-Parallel
Async Speed, BSP Guarantee

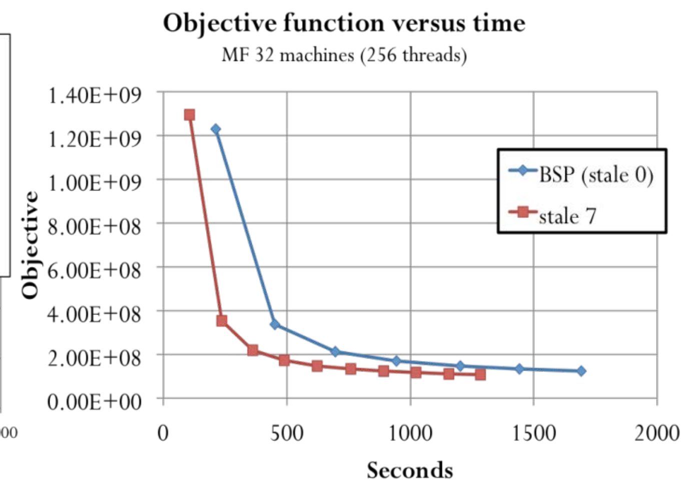
LDA



Lasso



Matrix Fact.



- Massive Data Parallelism
- Effective across different algorithms

SSP Data Parallel Convergence Theorem

[Ho et al., 2013, Dai et al., 2015]

Let observed staleness be γ_t

Let staleness mean, variance be $\mu_\gamma = \mathbb{E}[\gamma_t]$, $\sigma_\gamma = var(\gamma_t)$

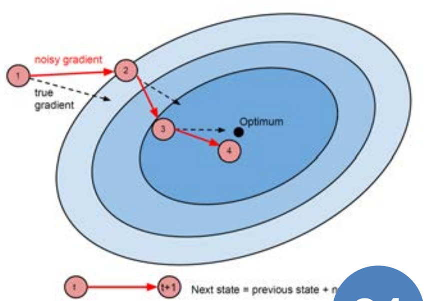
Theorem: Given L-Lipschitz objective f_t and step size h_t,

$$P\left[\frac{R[X]}{T} - \frac{\mathcal{O}(F^2 + \boxed{\mu_\gamma} L^2)}{\sqrt{T}} \geq \tau\right] \leq \exp\left\{\frac{-T\tau^2}{\mathcal{O}(\bar{\eta}_T \boxed{\sigma_\gamma} + L^2 s P \tau)}\right\}$$

where

$$R[X] := \sum_{t=1}^{T} f_t(\tilde{x}_t) - f(x^*) \qquad \bar{\eta}_T = \frac{\eta^2 L^4 (\ln T + 1)}{T} = o(T)$$

Explanation: the distance between true optima and current estimate decreases exponentially with more SSP iterations. *Lower staleness mean, variance μ_γ, σ_γ improve the convergence rate.*



Model-Parallel Proximal Gradient under SSP

- Model (e.g. SVM, Lasso …):

$$\min_{\mathbf{a}\in\mathbb{R}^d} \mathcal{L}(\boxed{\mathbf{a}}, \boxed{D}), \quad \text{where } \mathcal{L}(\mathbf{a}, D) = f(\mathbf{a}, D) + g(\mathbf{a})$$

data D, model a

- Model parallel
  - Model dimension d too large to fit in a single worker
  - Divide model among *P* workers $\mathbf{a} = (a_1, a_2, \ldots, a_P)$
- Algorithm: $\boxed{\forall p,}\ a_p(t+1) = a_p(t) + \boxed{\gamma_p(t)} \cdot F_p(\mathbf{a}^p(t))$

on worker p

workers can skip updates

$$= a_p(0) + \sum_{k=0}^{t} \gamma_p(k) \cdot F_p(\mathbf{a}^p(t))$$

staleness

$$\text{(local)} \quad \mathbf{a}^p(t) = \left(a_1(\tau_1^p(t)),\ \ldots\ ,\ a_P(\boxed{\tau_P^p(t)})\right)$$

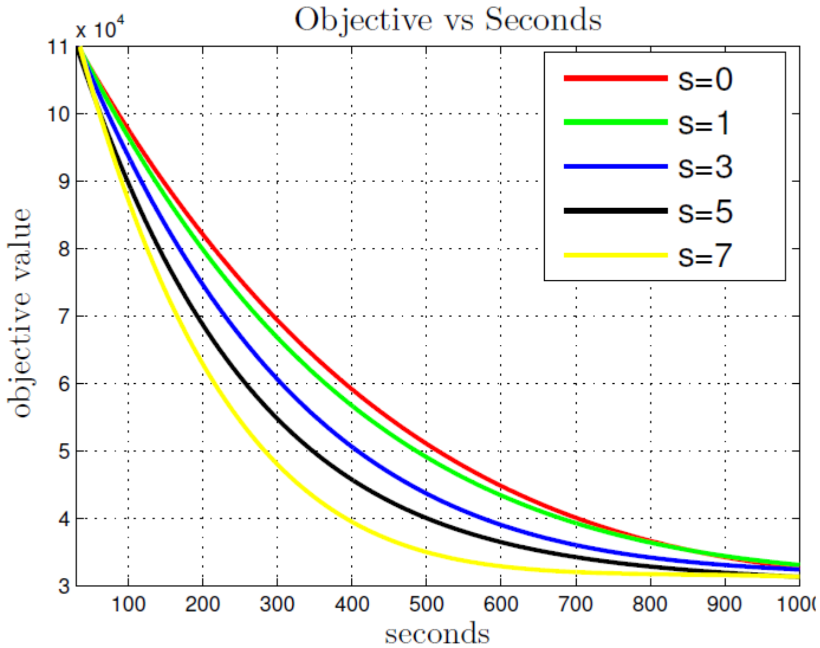
$$\text{(global)} \quad \mathbf{a}(t) = \left(a_1(t),\ \ldots\ ,\ a_P(t)\right).$$

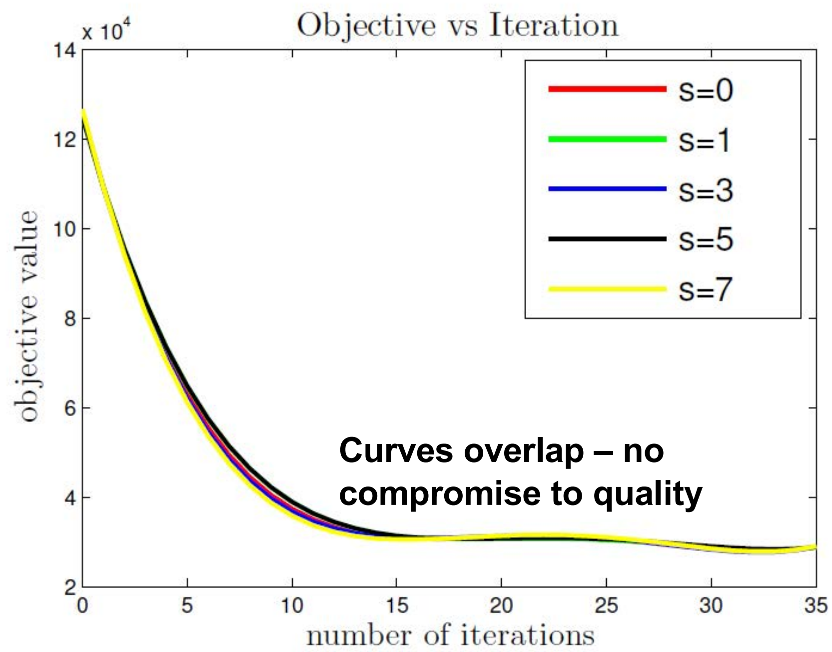
$$\mathbf{a}^p(t+1) := F_p(\mathbf{a}^p(t)) = \underbrace{\operatorname{prox}_{g_p}^{\eta}}_{\text{proximal step wrt } g}(a_p(t) - \overbrace{\eta\nabla_p f(\mathbf{a}^p(t))}^{\text{gradient step wrt } f}) - a_p(t)$$

  - worker p keeps local copy of the full model (can be avoided for linear models)

SSP Model-Parallel
Async Speed, BSP Guarantee

Lasso: 1M samples, 100M features, 100 machines





- Massive Model Parallelism
- Effective across different algorithms

SSP Model Parallel Convergence Theorem

[Zhou et al., 2016]

Theorem: Given that the SSP delay is bounded, with appropriate step size and under mild technical conditions, then

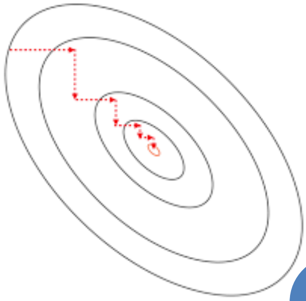
Finite length

$$\sum_{t=0}^{\infty} \|\mathbf{a}(t+1) - \mathbf{a}(t)\| < \infty \qquad \sum_{t=0}^{\infty} \|\mathbf{a}^p(t+1) - \mathbf{a}^p(t)\| < \infty$$

In particular, the global and local sequences converge to the same critical point, with rate $O(t^{-1})$:

$$\mathcal{L}\left(\tfrac{1}{t}\sum_{k=1}^{t} \mathbf{a}(k)\right) - \inf \mathcal{L} \leq O\left(t^{-1}\right)$$

Explanation: Finite length guarantees that the algorithm stops (the updates must eventually go to zero). Furthermore, the algorithm converges at rate $O(t^{-1})$ to the optimal value; same as BSP model parallel.



System/Algorithm Co-design

3. What to Communicate:

Trading-off computing and communication

Matrix-Parameterized Models (MPMs)

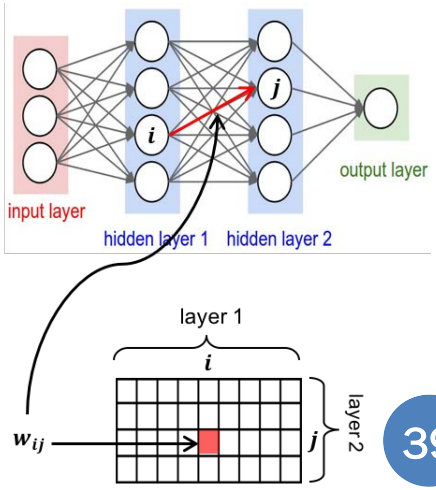
Matrix parameter W

$$\min_{W} \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i; b_i) + h(W)$$

Loss function

Regularizer

Distance Metric Learning, Topic Models, Sparse Coding, Group Lasso, Neural Network, etc.



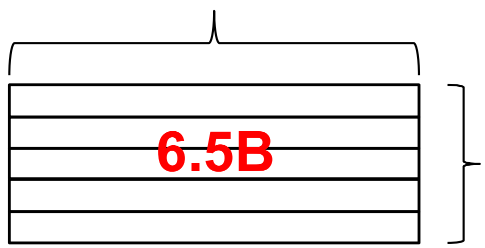
Big MPMs

Billions of params = 10-100 GBs, costly network synchronization

What do we actually need to communicate?

Multiclass Logistic Regression on Wikipedia

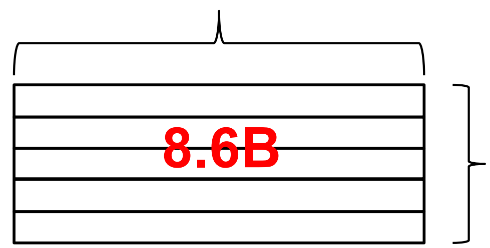
Feature dim. = 20K



#classes=325K

Distance Metric Learning on ImageNet

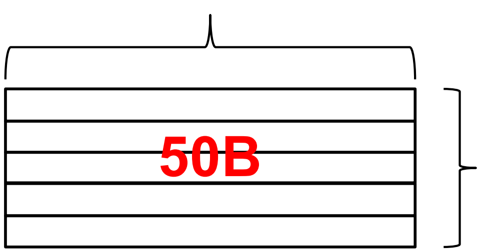
Feature dim. = 172K



Latent dim. = 50K

Topic Model on WWW

Feature dim. = 1M



Dic. Size= 1M

Neural Network of Google Brain

#neurons in layer 0 = 40K

#neurons in layer 1 = 33K

Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Full Updates

- Let matrix parameters be W. **Need to send parallel worker updates** ΔW to other machines…
  - Primal stochastic gradient descent (SGD)

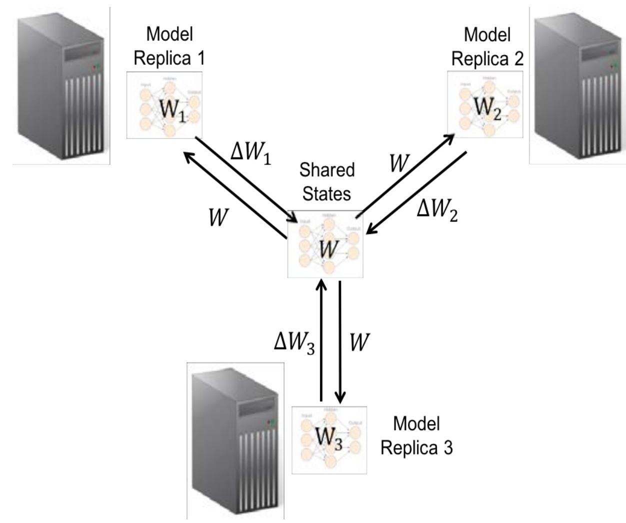
$$\min_{W} \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i;b_i)+h(W)$$

$$\boxed{\Delta W = \frac{\partial f(Wa_i,b_i)}{\partial W}}$$

  - Stochastic dual coordinate ascent (SDCA)

$$\min_{Z} \frac{1}{N}\sum_{i=1}^{N} f_i^*(-z_i)+h^*(\frac{1}{N}ZA^{\mathrm{T}})$$

$$\boxed{\Delta W = (\Delta z_i)a_i}$$



Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Pre-update: the Sufficient Vectors [Xie et al., UAI 2015]

- **Full parameter matrix update** ΔW can be computed as **outer product of two vectors** uv^T -- the sufficient vectors (SV)
  - Primal stochastic gradient descent (SGD)

$$\min_W \frac{1}{N}\sum_{i=1}^{N} f_i(Wa_i;b_i) + h(W)$$

$$\boxed{\Delta W = uv^{\mathrm{T}} \quad u = \frac{\partial f(Wa_i,b_i)}{\partial(Wa_i)} \quad v = a_i}$$

  - Stochastic dual coordinate ascent (SDCA)

$$\min_Z \frac{1}{N}\sum_{i=1}^{N} f_i^*(-z_i) + h^*(\frac{1}{N}ZA^{\mathrm{T}})$$

$$\boxed{\Delta W = uv^{\mathrm{T}} \quad u = \Delta z_i \quad v = a_i}$$

More on Sufficient Vectors

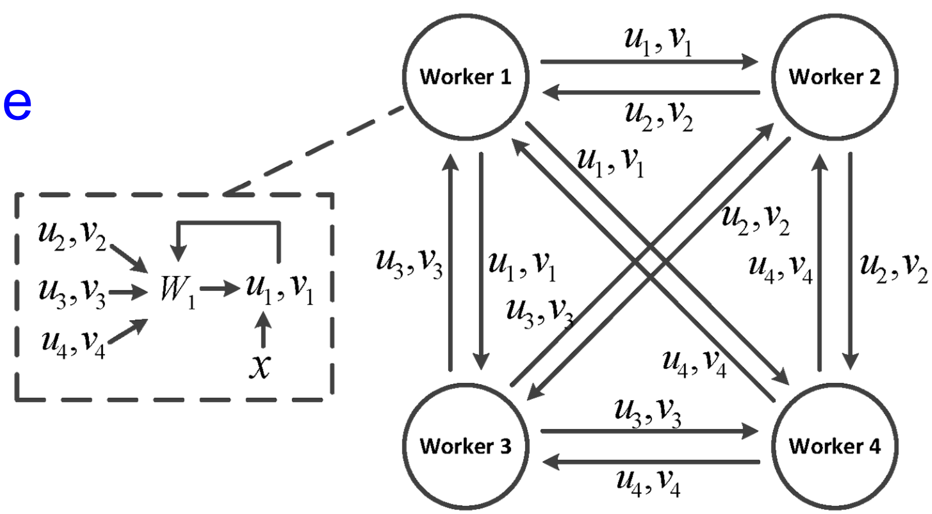
- Other Cases
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm

$$\Delta W = u(u-v)^{\mathrm{T}} - vu^{\mathrm{T}}$$

  - Contrastive divergence algorithm in Restricted Boltzmann Machine

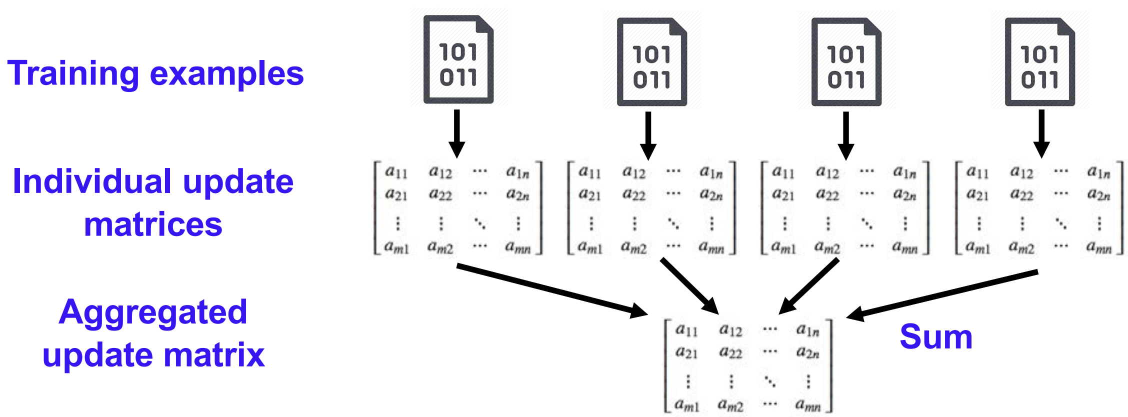
$$\Delta W = u_1 v_1^{\mathrm{T}} - u_2 v_2^{\mathrm{T}}$$

- What about communicating the lightweight SV updates (u,v), instead of the expensive full-matrix ΔW updates?

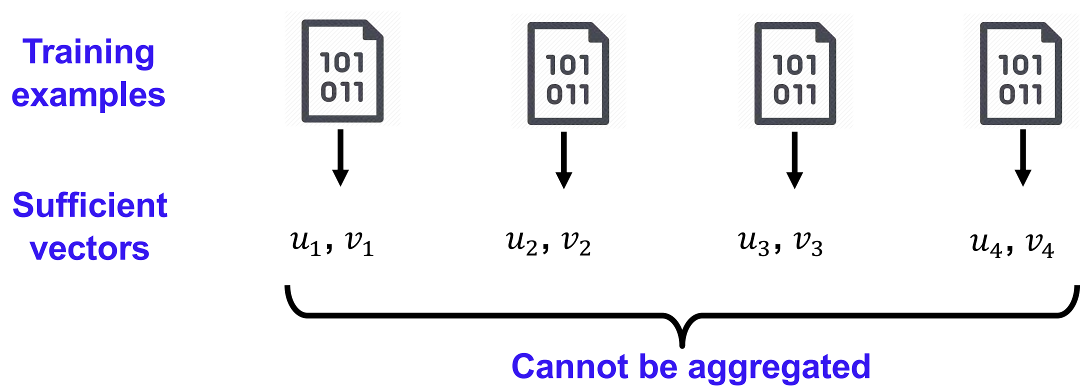


A computing & communication tradeoff

- Full update:



- Pre-update



- Stochastic algorithms
  - Mini-batch: C samples

Matrix Representation	$O(JK)$
SV Representation	$O((J+K)C)$

SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Storage advantage

- Store SFs in memory to represent parameters
- Space complexity

MR	$O(JK)$
SVR	$O(t(J+K))$

Advantageous when t is small

- Memory Management

Properties of SVs

- Read only →
- Dynamically growing →

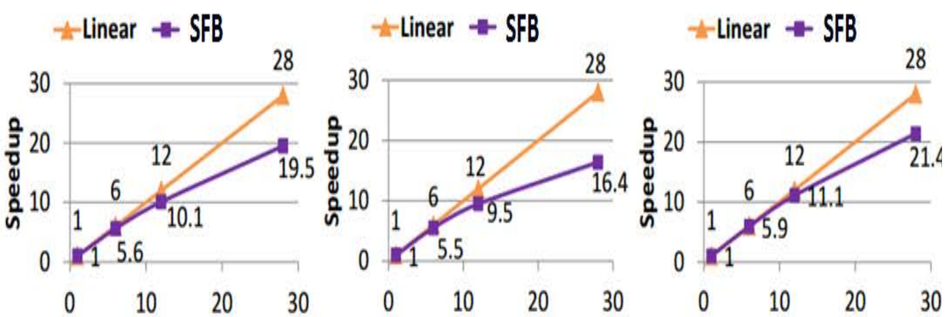
Memory Management

- GPU texture memory
  - Provide high performance read only cache
- Dynamic allocation of memory blocks

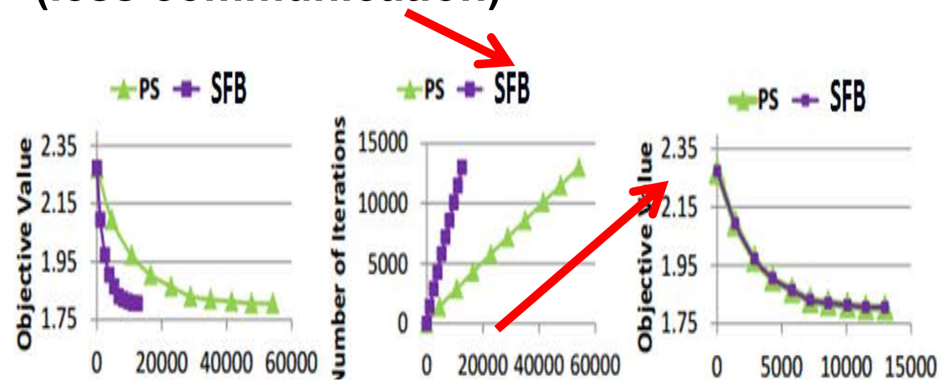
Why is SFB faster?

- Faster than PS and Spark

- Near-linear scalability

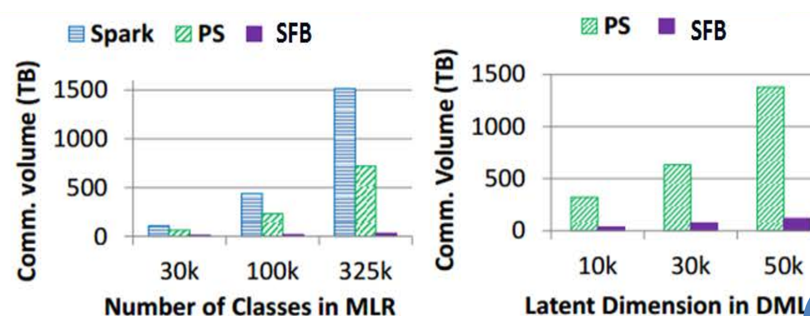


Because SFB has faster iterations (less communication)



… while keeping the same iteration quality as PS

SFB communication up to 100x smaller than PS and Spark



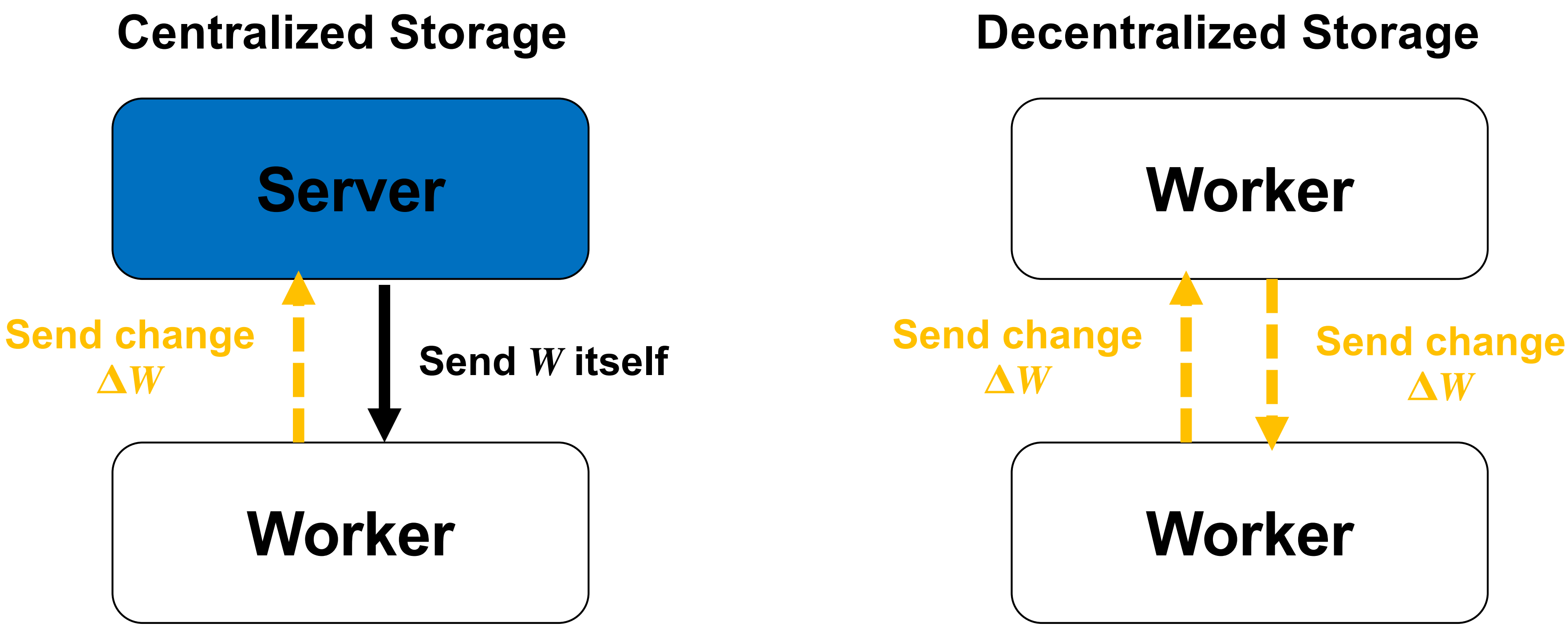
Theoretical guarantees?

System/Algorithm Co-design

4. How to Communicate:

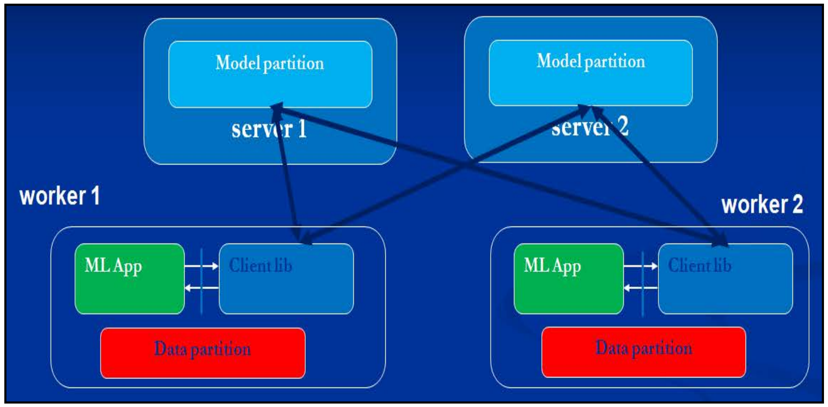
What Topologies to use?

Parameter Storage and Communication Paradigms

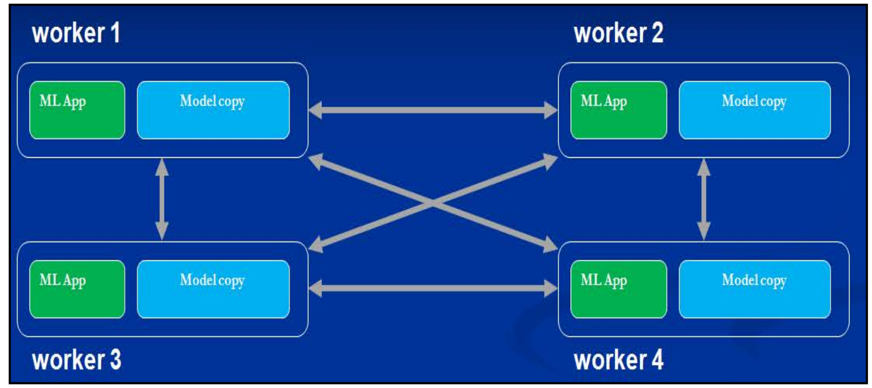


- **Centralized:** send parameter W itself from server to worker
  - Advantage: allows compact comms topology, e.g. bipartite
- **Decentralized:** always send changes ΔW between workers
  - Advantage: more robust, homogeneous code, low communication (?)

Topologies: Master-Slave versus P2P?



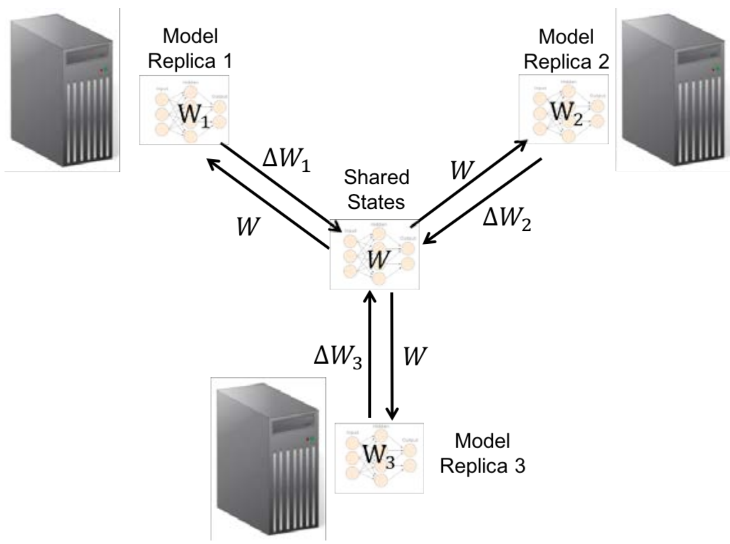
- Master-slave
  - Used with **centralized storage** paradigm
  - **Disadvantage:** need to code/manage clients and servers separately
  - **Advantage:** bipartite topology is comms-efficient
  - Popular for Parameter Servers: Yahoo LDA, Google DistBelief, Petuum PS, Project Adam, Li&Smola PS, …



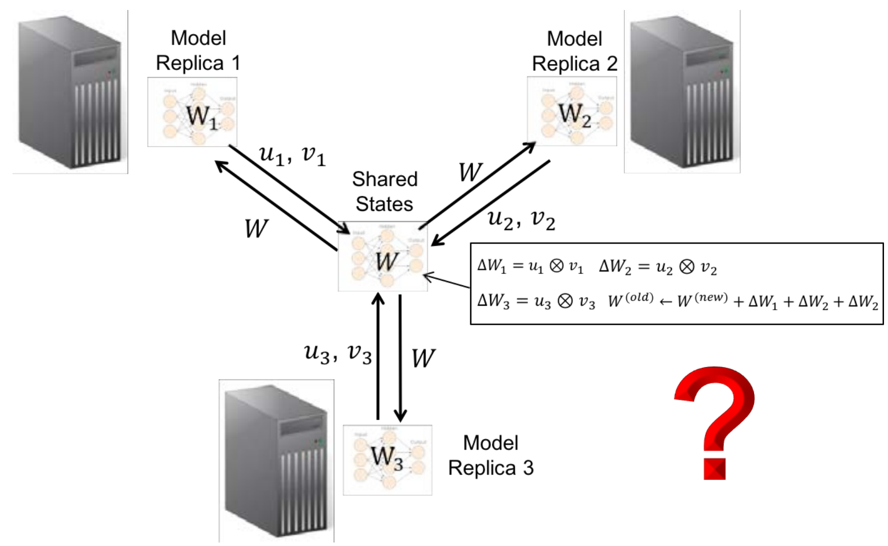
- P2P
  - Used with **decentralized storage**
  - **Disadvantage (?):** high comms volume for large # of workers
  - **Advantage:** same code for all workers; no single point of failure, high elasticity to resource adjustment
  - Less well-explored due to perception of high communication overhead?

Synchronization of Parameter Replicas

parameter server



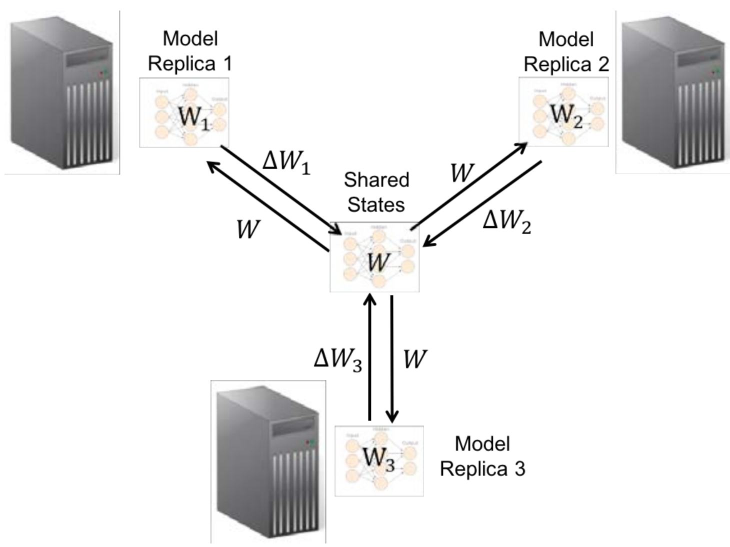
Transfer SVs instead of ΔW



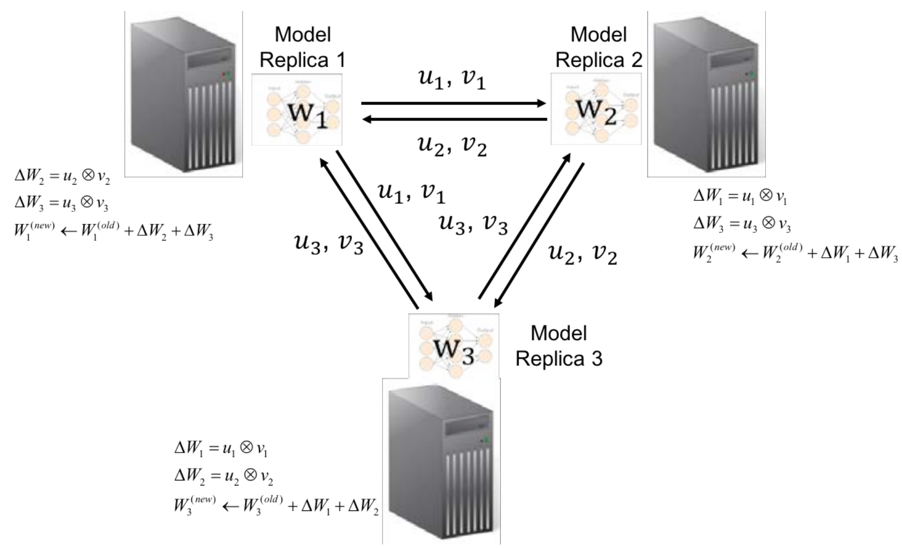
- **Sync directly on W:**
  - High communication cost
- **Sync via SVs:**
  - Reduce network traffic in the worker-to-server direction
  - Server-to-worker traffic remains high since W cannot be represented as SVs

Synchronization of Parameter Replicas

parameter server



Transfer SVs instead of ΔW

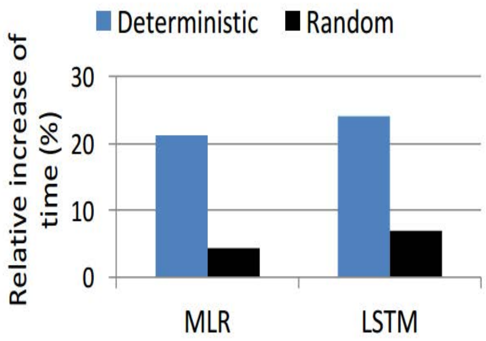


- **A Cost Comparison**

	Size of one message	Number of messages	Network Traffic
P2P SV-Transfer	$O(J+K)$	$O(P^2)$	$O((J+K)P^2)$
Parameter Server	$O(JK)$	$O(P)$	$O(JKP)$

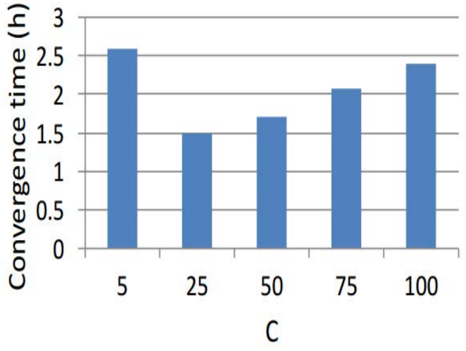
How to reduce traffic in P2P?

- Random Partial Broadcasting
  - Each machine randomly selects Q<<P machines to send messages (instead of full broadcast)
  - Message cost reduced: from $O(P^2)$ to $O(PQ)$, scales linearly with machine count P!

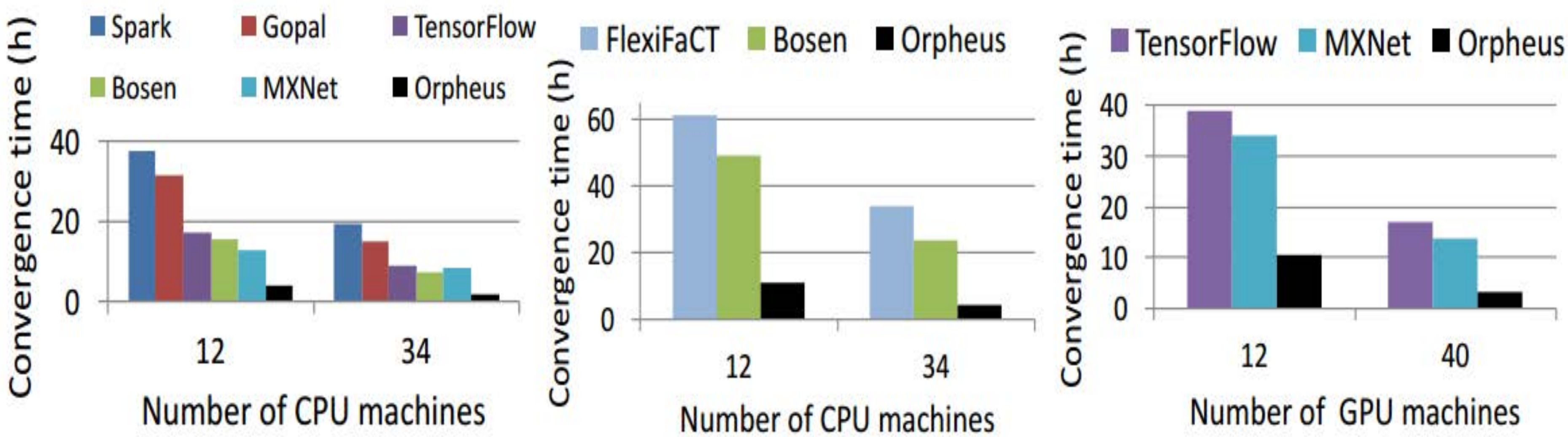


- SV Selection
  - Select a subset of "representative" SVs to communicate

$$\sum_{k=1}^{K} \left\| V^{(k)} - V_\tau^{(k)} \left(V_\tau^{(k)} \right)^\dagger V^{(k)} \right\|_2$$

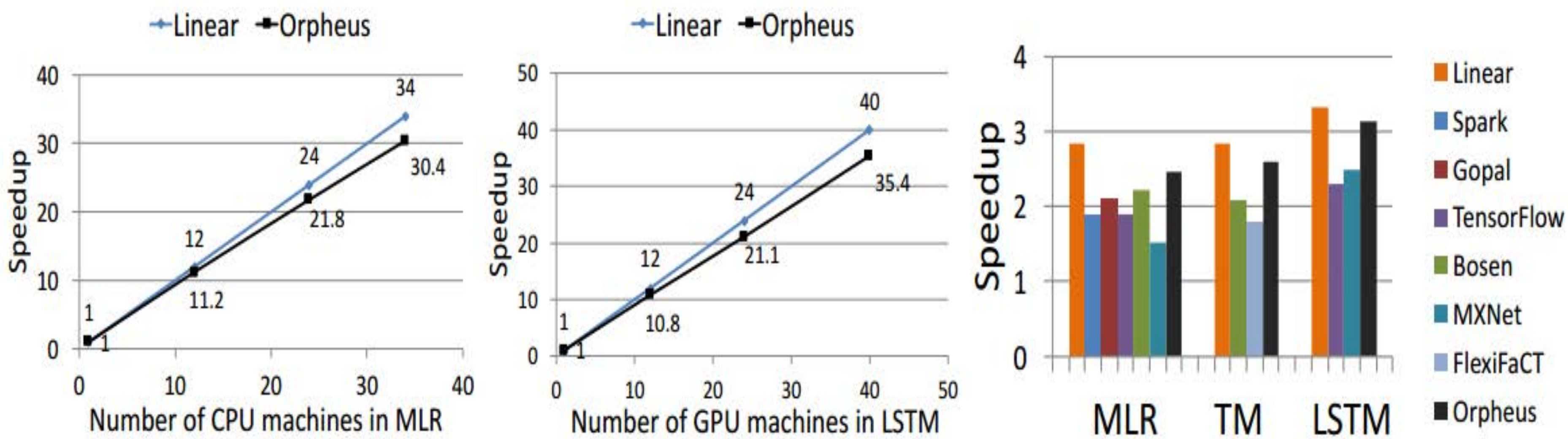


Convergence Speed



Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Scalability



Convergence Guarantee

- Assumptions
  - Bridging model
    - Staleness Synchronous Parallel (SSP) with staleness parameter s
    - Bulk Synchronous Parallel is a special case of SSP when $s = 0$
  - Communication methods
    - Partial broadcast (PB): sending messages to a subset of Q $(Q < P - 1)$ machines
    - Full broadcast is a special case of PB when $Q = P - 1$
  - Additional assumptions

Assumption 1. *(1) For all j, f_j is continuously differentiable and F is bounded from below; (2) $\nabla F, \nabla F_p$ are Lipschitz continuous with constants L_F and L_p, respectively, and let $L = \sum_{p=1}^{P} L_p$; (3) There exists G, σ^2 such that for all p and c, we have (almost surely) $\|U_p(\mathbf{W}_p^c, I_p^c)\| \leq G\eta$ and $\mathbb{E}\| \, |S_p| \sum_{j \in I_p} \nabla f_j(\mathbf{W}) - \nabla F_p(\mathbf{W}) \, \|_2^2 \leq \sigma^2$.*

Convergence Guarantee

- Results

Theorem 1. *Let Assumption 1 hold, and let* $\{\mathbf{W}_p^c\}$, $p = 1, \ldots, P$, $\{\mathbf{W}^c\}$ *be the local sequences and the auxiliary sequence, respectively.*

Under full broadcasting (i.e., $Q = P - 1$*) and set the learning rate* $\eta := \eta_c = O(\sqrt{\frac{1}{L\sigma^2 Psc}})$*, we have*

- $\liminf_{c\to\infty} \mathbb{E}\|\nabla F(\mathbf{W}^c)\| = 0$*, hence there exists a subsequence of* $\nabla F(\mathbf{W}^c)$ *that almost surely vanishes;*
- $\lim_{c\to\infty} \max_p \|\mathbf{W}^c - \mathbf{W}_p^c\| = 0$*, i.e., the maximal disagreement between all local sequences and the auxiliary sequence converges to 0 (almost surely);*
- *There exists a common subsequence of* $\{\mathbf{W}_p^c\}$ *and* $\{\mathbf{W}^c\}$ *that converges almost surely to a stationary point of* F*, with the rate* $\min_{c\le C} \mathbb{E}\|\sum_{p=1}^P \nabla F_p(\mathbf{W}_p^c)\|_2^2 \le O\left(\sqrt{\frac{L\sigma^2 Ps}{C}}\right)$

Under partial broadcasting (i.e., $Q < P - 1$*) and set a constant learning rate* $\eta = \frac{1}{CLG(P-Q)}$*, where* C *is the total number of iterations. Then we have*

$$\min_{c\le C} \mathbb{E}\left[\|\sum_{p=1}^P \nabla F_p(\mathbf{W}_p^c)\|_2^2\right] \le O\left(LG(P-Q) + \frac{P(sG+\sigma^2)}{CG(P-Q)}\right).$$

Hence, the algorithm converges to a $O(LG(P-Q))$ *neighbourhood if* $C \to \infty$.

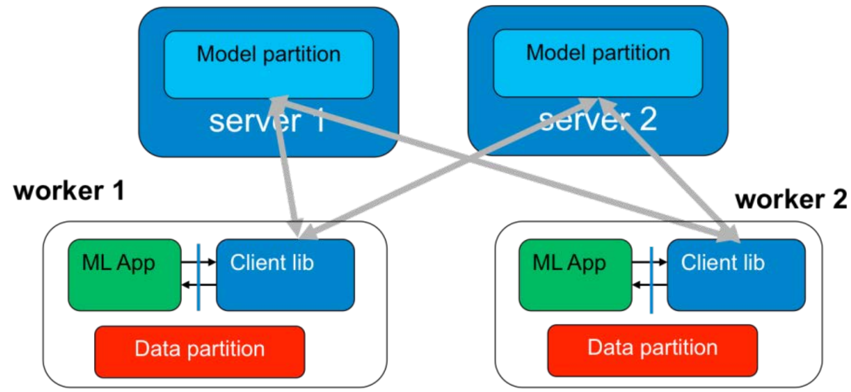
Convergence Guarantee

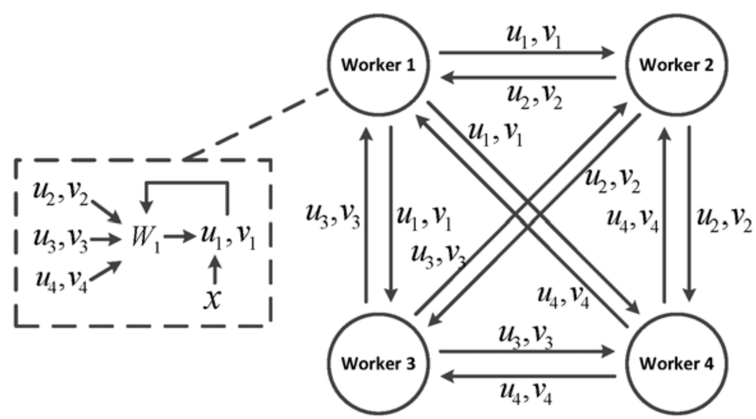
- Take-home message:
  - Under full broadcasting, given a properly-chosen learning rate, all local worker parameters W_p^c eventually converge to stationary points (i.e. local minima) of the objective function, despite the fact that SV transmission can be delayed by up to s iterations.
  - Under partial broadcasting, the algorithm converges to a $O(LG(P-Q))$ neighbourhood if $C \longrightarrow \infty$.

Hybrid Updates: PS + SFB

- Hybrid communications: Parameter Server + Sufficient Factor Broadcasting
  - Parameter Server: Master-Slave topology
  - Sufficient factor broadcasting: P2P topology

- For problems with a mix of large and small matrices,
  - Send small matrices via PS
  - Send large matrices via SFB

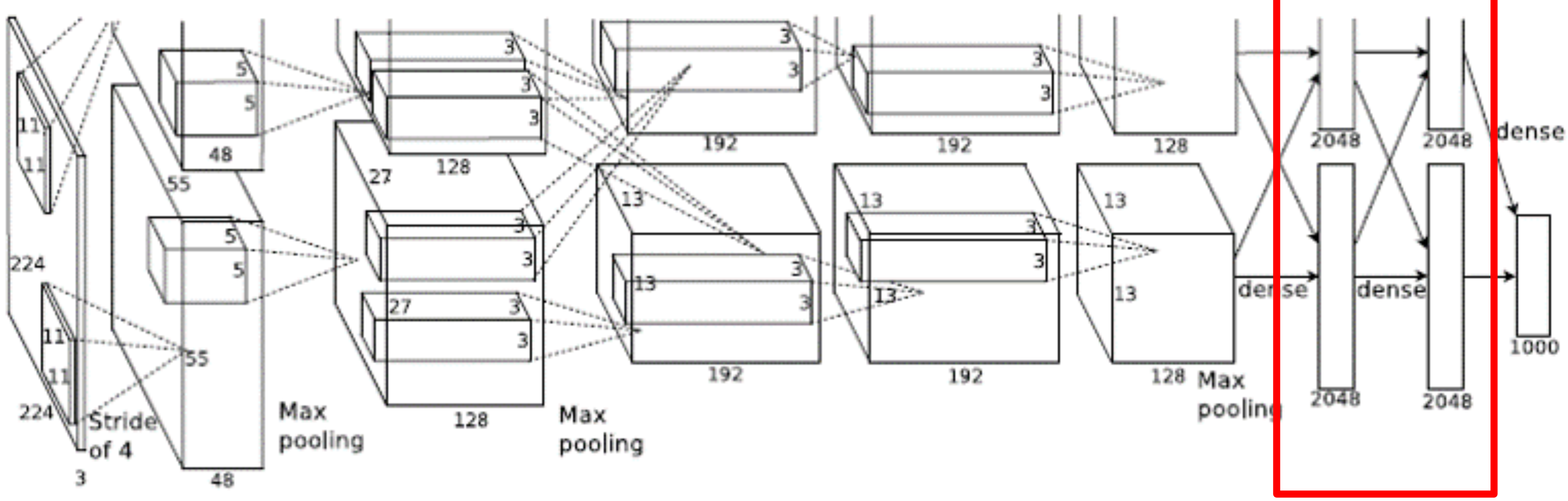




Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

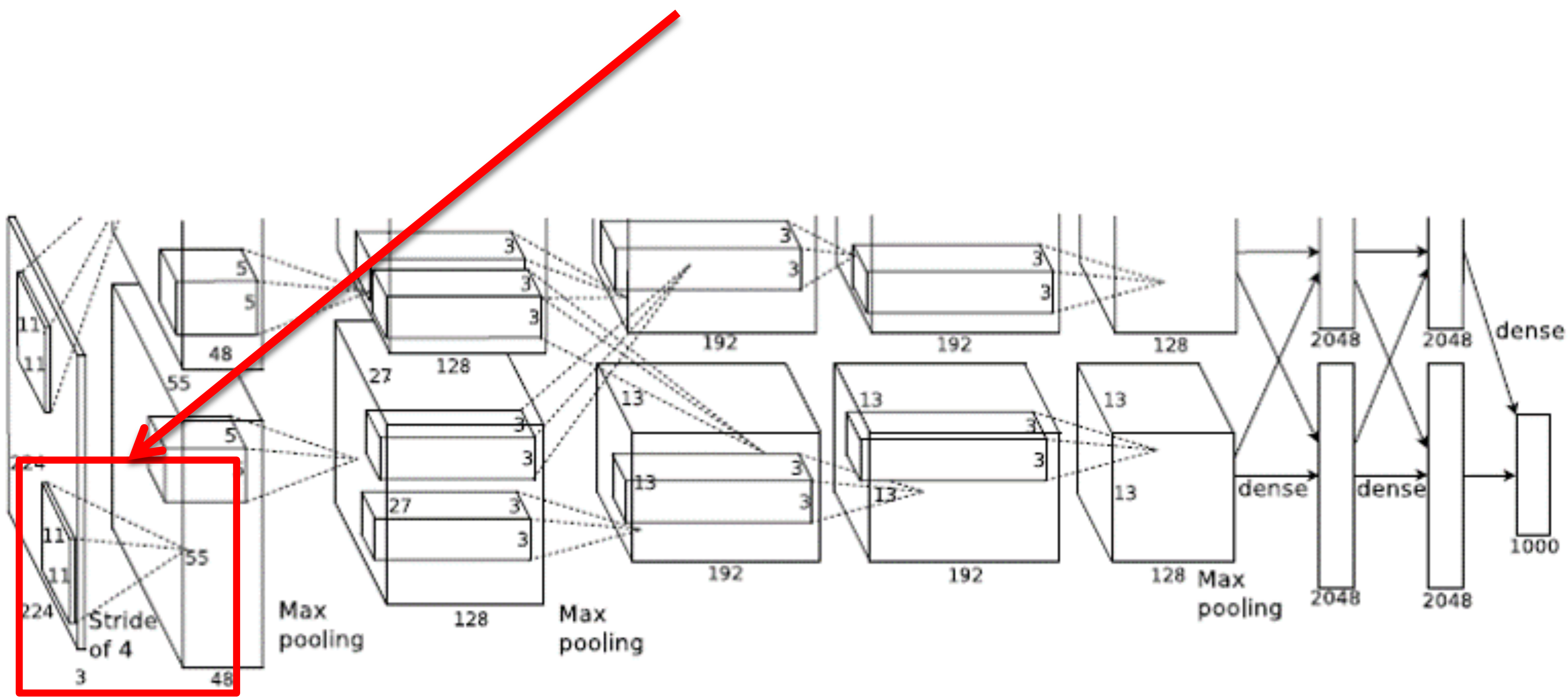
Hybrid example: CNN [Zhang et al., 2015]

- Example: AlexNet CNN model
  - Final layers = 4096 * 4096 matrix (17M parameters)
  - Use SFB to communicate
    - 1. Decouple into two 4096 vectors: u, v
    - 2. Transmit two vectors
    - 3. Reconstruct the gradient matrix



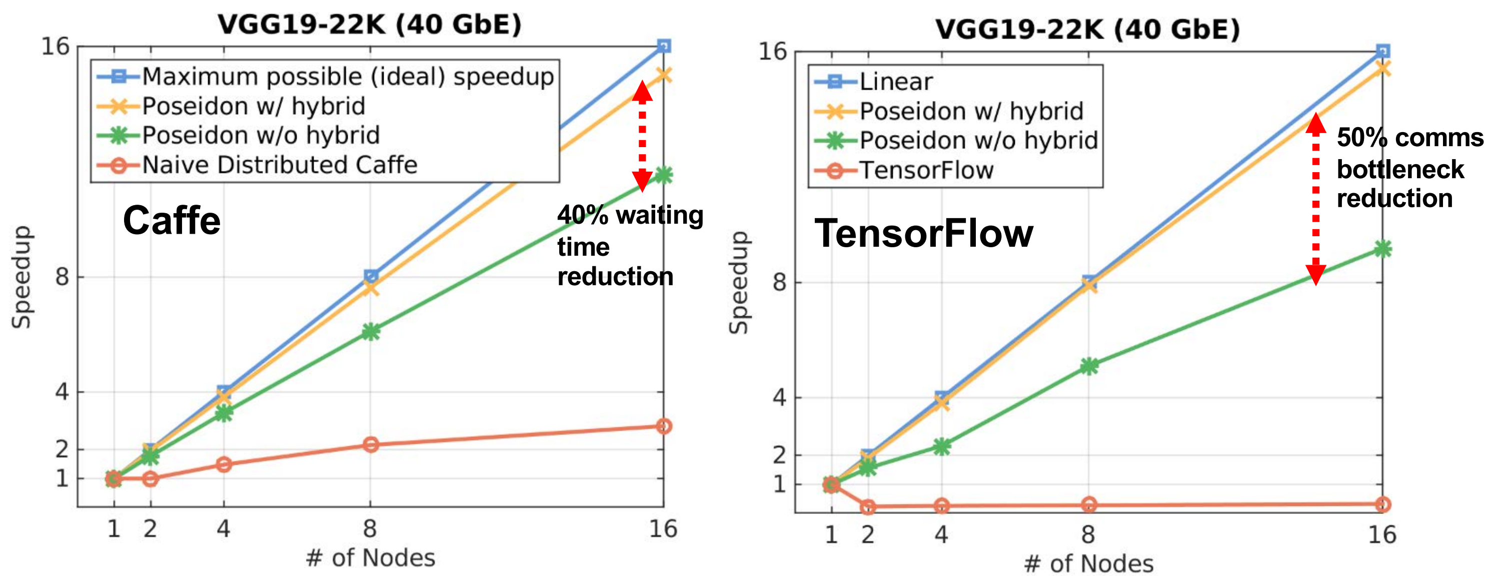
Hybrid example: CNN [Zhang et al., 2015]

- Example: AlexNet CNN model
  - Convolutional layers = e.g. 11 * 11 matrix (121 parameters)
  - Use Full-matrix updates to communicate
    - 1. Send/receive using Master-Slave PS topology



Hybrid CNN and Managed Communications [Zhang et al., 2015]

- Hybrid comms eliminate up to 50% of comms bottlenecks in CNNs
  - Use managed comms [Wei et al., 2015] for further 33% comms bottleneck reduction



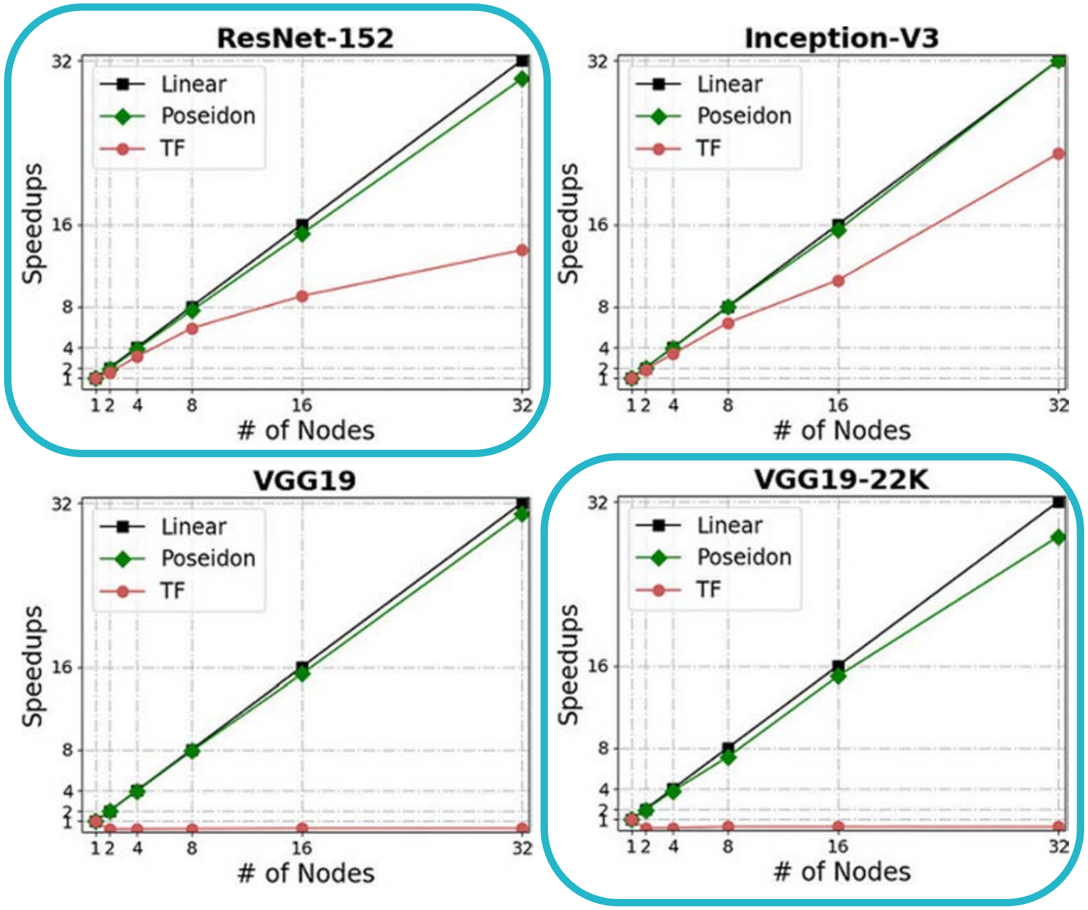
  - Good Science: Count machines, not GPUs; Measure performance, not throughput
    - Greatest comms bottleneck is between machines, not GPUs (one machine can have 8 GPUs)
    - e.g. Tensorflow blog reports perfectly-linear scaling up to 8 GPUs, but not how many machines were used (other important but missing info: top-1 or top-5 accuracy? Accuracy measured on train or test data?)

SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Poseidon Scalability

ILSVRC2015 winner
params: 60.2M

Just demoed



ILSVRC2013 winner
params: 60.2M

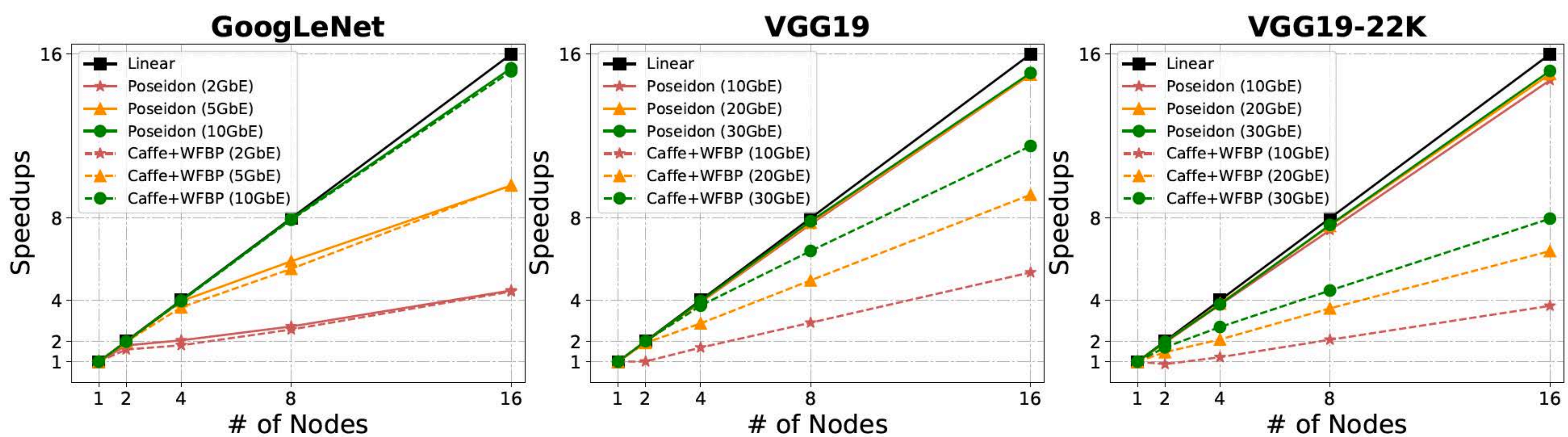
ILSVRC2013 winner
params: 143M
Most-adopted feature
Extraction network

ILSVRC2013 winner
params: 229M
Extended to 22K categories

Extremely Large DL problem, TensorFlow cannot scale at all

Poseidon scalability (Limited Bandwidth)

- Scenario:
  - Training Large Models
  - Limited network bandwidth



 # parameters 5M 143M 229M

Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

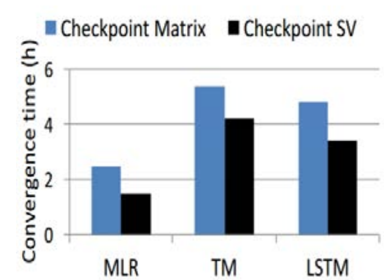
Summary

1. **How to Distribute?**
   - Structure-Aware Parallelization
   - Work Prioritization
2. **How to Bridge Computation and Communication?**
   - BSP Bridging Model
   - SSP Bridging Model for Data and Model Parallel
3. **What to Communicate?**
   - Full Matrix updates
   - Sufficient Factor updates
   - Hybrid FM+SF updates (as in a DL model)
4. **How to Communicate?**
   - Managed comms – interleave comms/compute, prioritized comms
   - Parameter Storage: Centralized vs Decentralized
   - Communication Topologies: Master-Slave, P2P, Partial broadcast

SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

Other system issues:

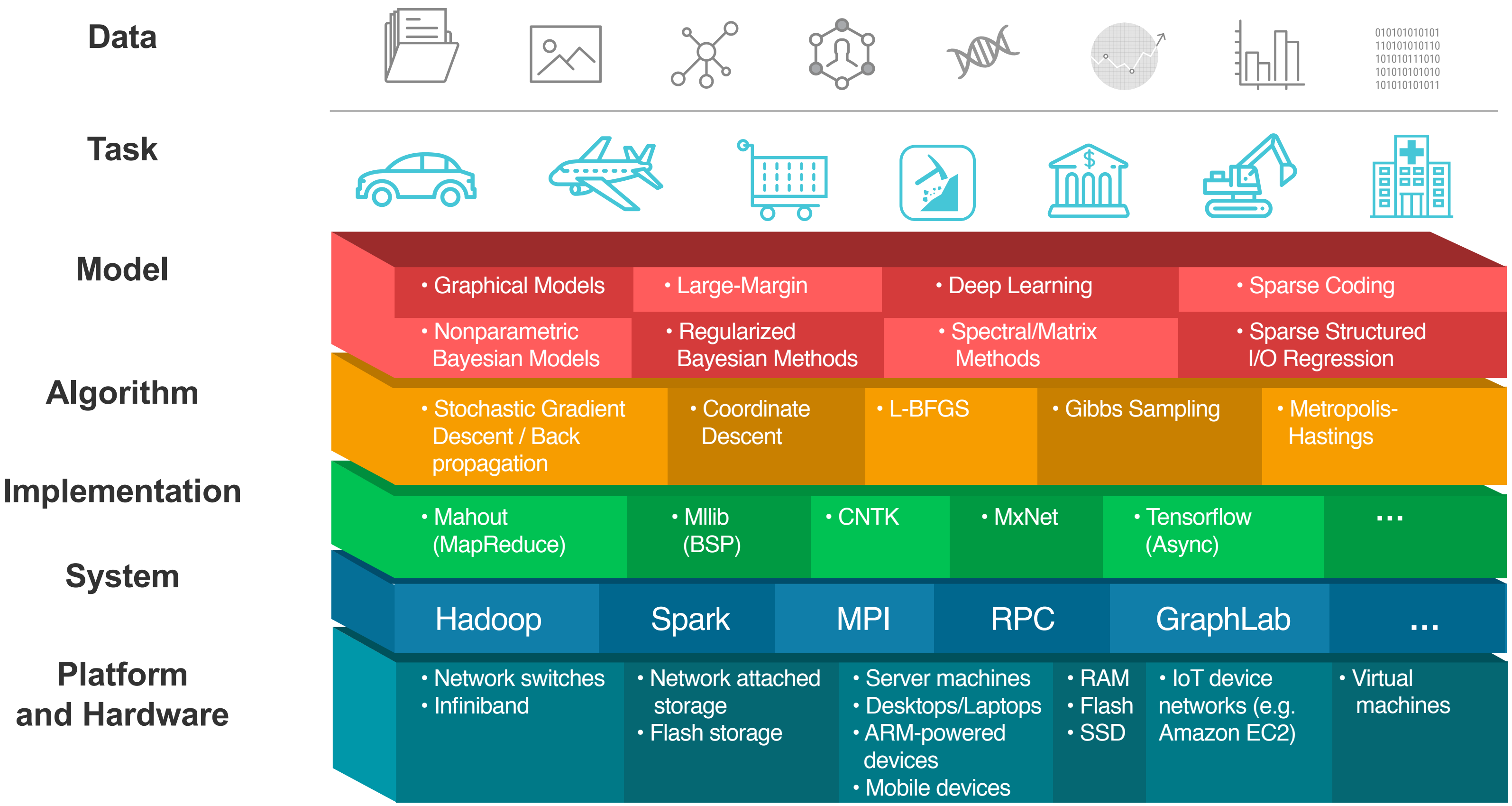
- Broadcast schemes
  - Tailored to system configurations
    - Hardware-level
      - CPU-to-CPU, GPU-to-GPU
      - InfiniBand, Ethernet
    - Software-level
      - BSP, SSP
      - Full broadcast, partial broadcast
- Fault Tolerance
  - SV-based checkpoint: save SVs generated in each clock onto disk
    - Light-weight in disk IO
    - No waste of compute cycles
    - Fine-grained (any clock) rollback



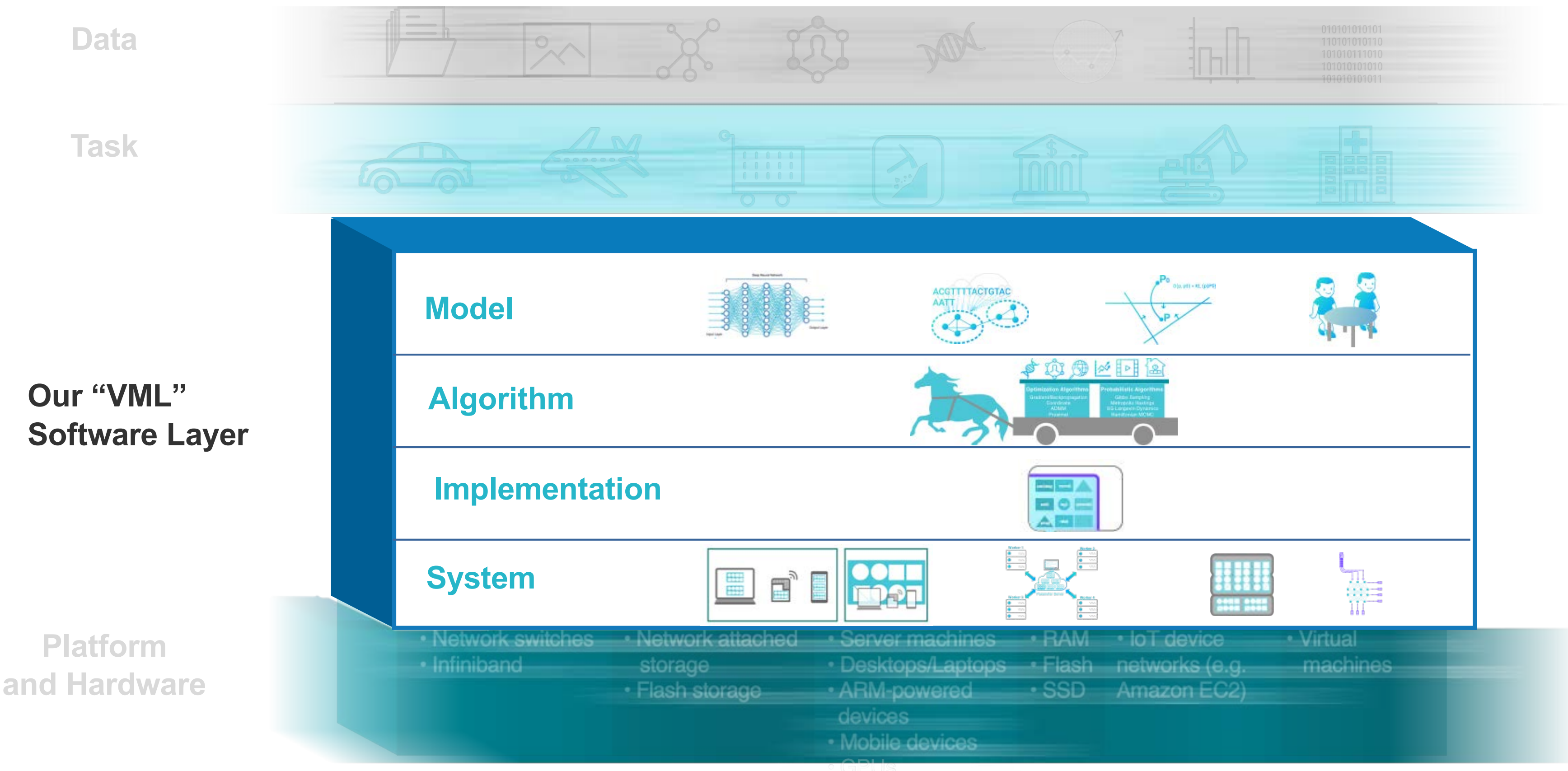
- Omni-Hardware
  - Each operator has a CPU and GPU implementation
  - Kernel fusion
- Elasticity
  - Adding/removing machines do not interrupt current execution

In Closing: Toward New System for ML/AI

Elements of Modern AI

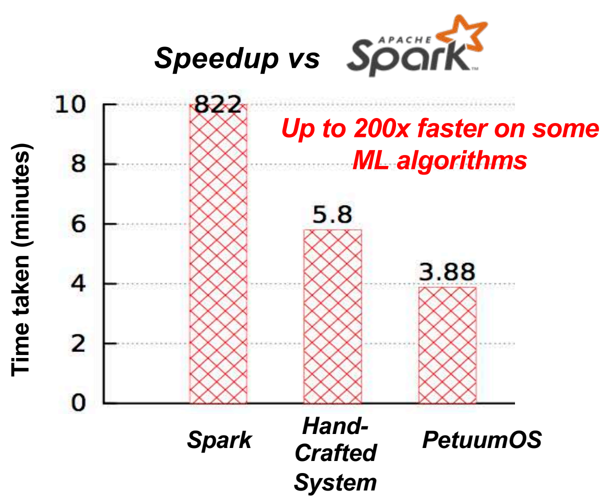


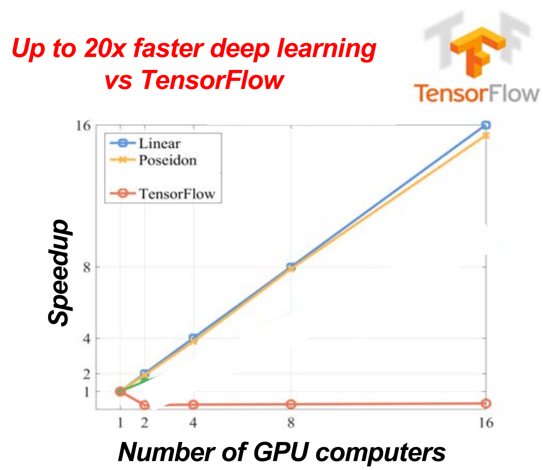
Sys-Alg Co-design Inside!

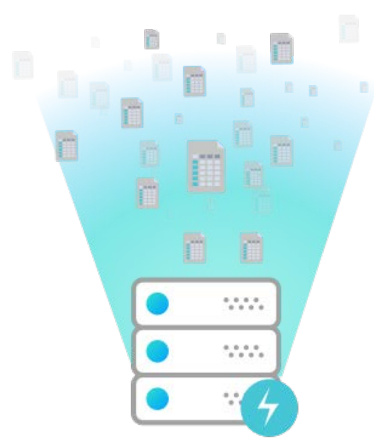


Better Performance

- Fast and Real-Time
  - Orders of magnitude faster than Spark and TensorFlow
  - As fast as hand-crafted systems
- Any Scale
  - Perfect straight-line speedup with more computing devices
  - Spark, TensorFlow can slow down with more devices
- Low Resource
  - Turning a regular cluster into a super computer:
    - Achieve AI results with much more data, but using fewer computing devices
    - Google brain uses ~1000 machines whereas Petuum uses ~10 for the same job

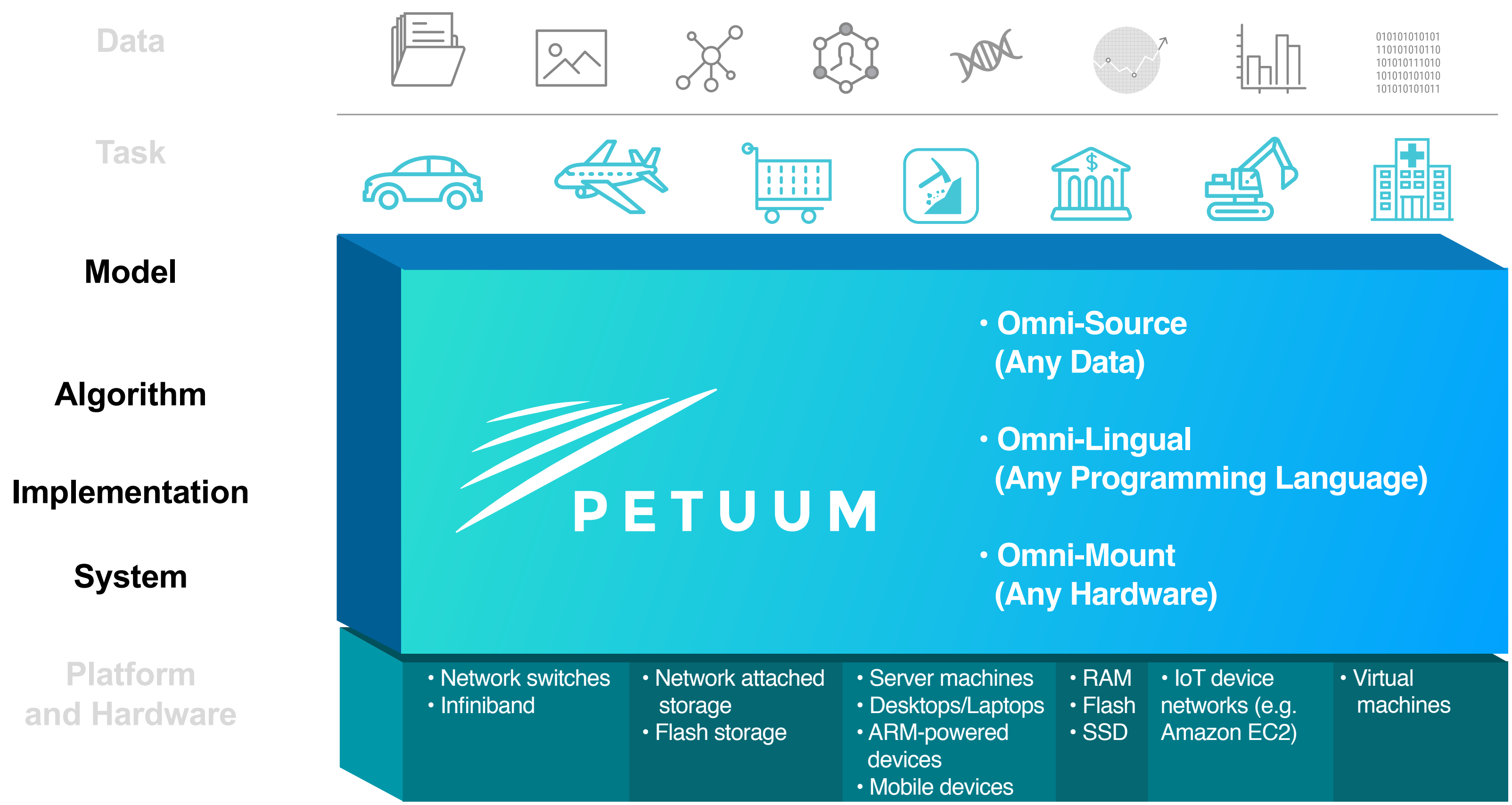






Laboratory for Statistical Artificial InteLligence & INtegrative Genomics

A Petuum Vision



Acknowledgements

SAILING LAB
Laboratory for Statistical Artificial InteLligence & INtegrative Genomics


Jin Kyu Kim


Seunghak Lee


Jinliang Wei

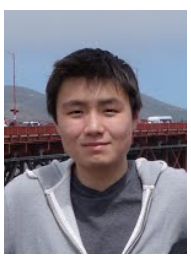

Wei Dai


Pengtao Xie


Xun Zheng

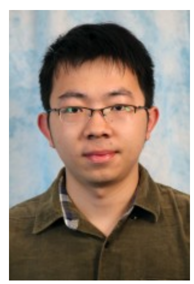

Abhimanu Kumar

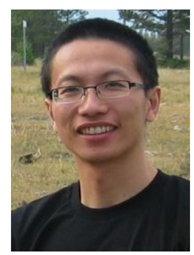

Qirong Ho


Aurick Qiao

www.sailing.cs.cmu.edu

$$$:


Hao Zhang


Yaoliang Yu


Garth Gibson

Greg Ganger


Phillip Gibbons


James Cipar