Distributed Optimization

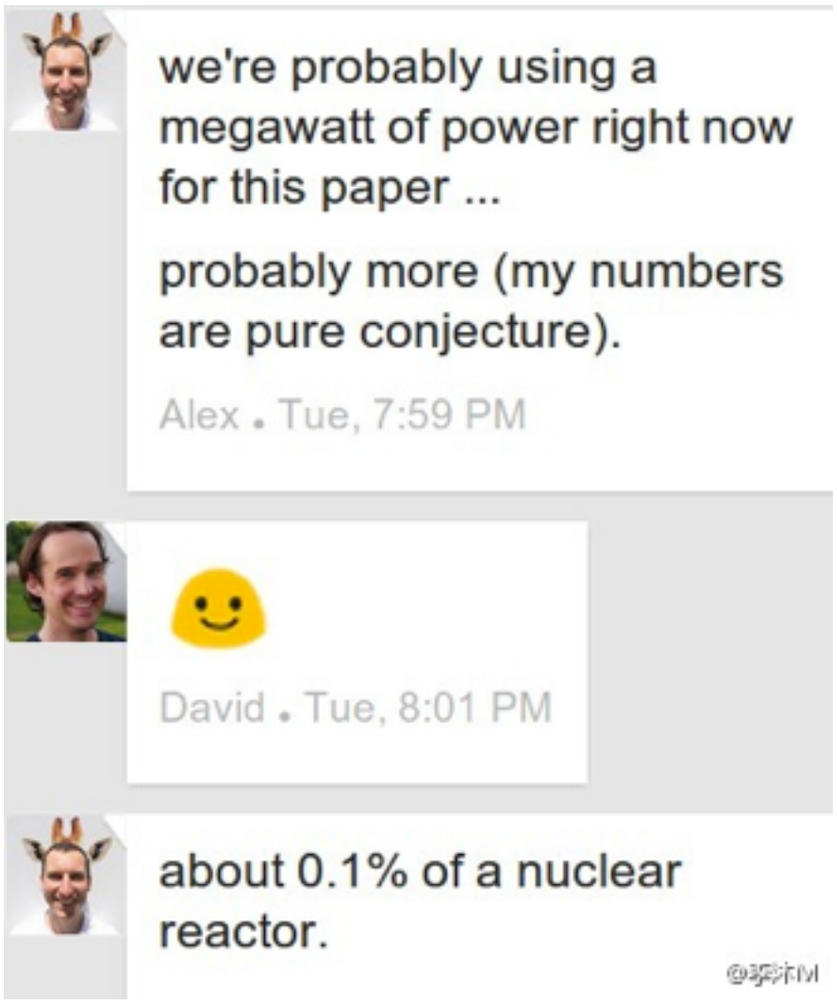
Tutorial of Parameter Server

Mu Li
IDL Baidu & CSD CMU

Why

Motivation

- For big data, machines are scarce resources
- We must find the solution as economic as possible
  - ★ saving several iterations may mean hundreds of machine hours (hundreds of $ on ec2)



Objective function

$$F(w) = f(w) + g(w) = \sum_{i=1}^{m} f(w, x_i, y_i) + g(w)$$

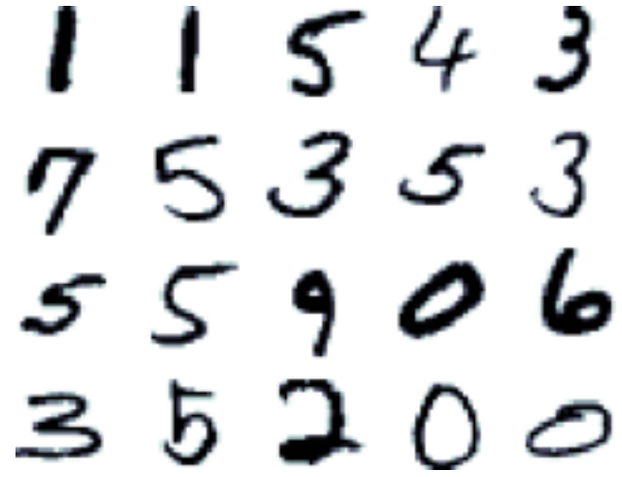
loss

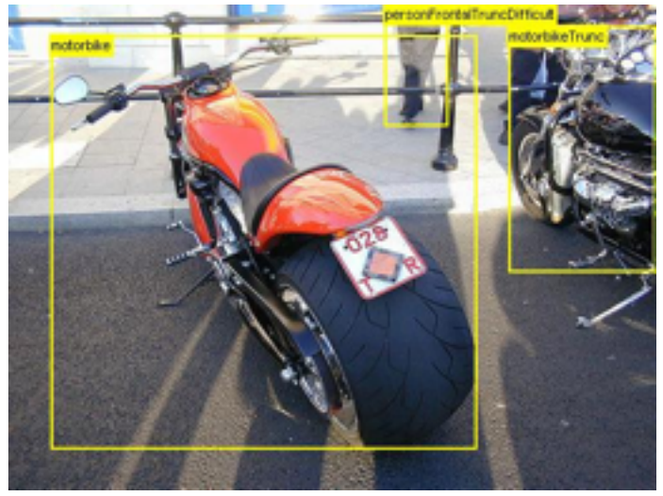
regularizer

- Many choices of losses, regularizers, …
- We have lots of data
  - Does not fit on single machine
  - Bandwidth constraints
  - May grow in real time

Batch and Online

- Batch
  - Very large dataset available
  - Require parameter only at the end
    - optical character recognition
    - speech recognition
    - image annotation / categorization
    - machine translation
- Online
  - Spam filtering
  - Computational advertising
  - Content recommendation / collaborative filtering

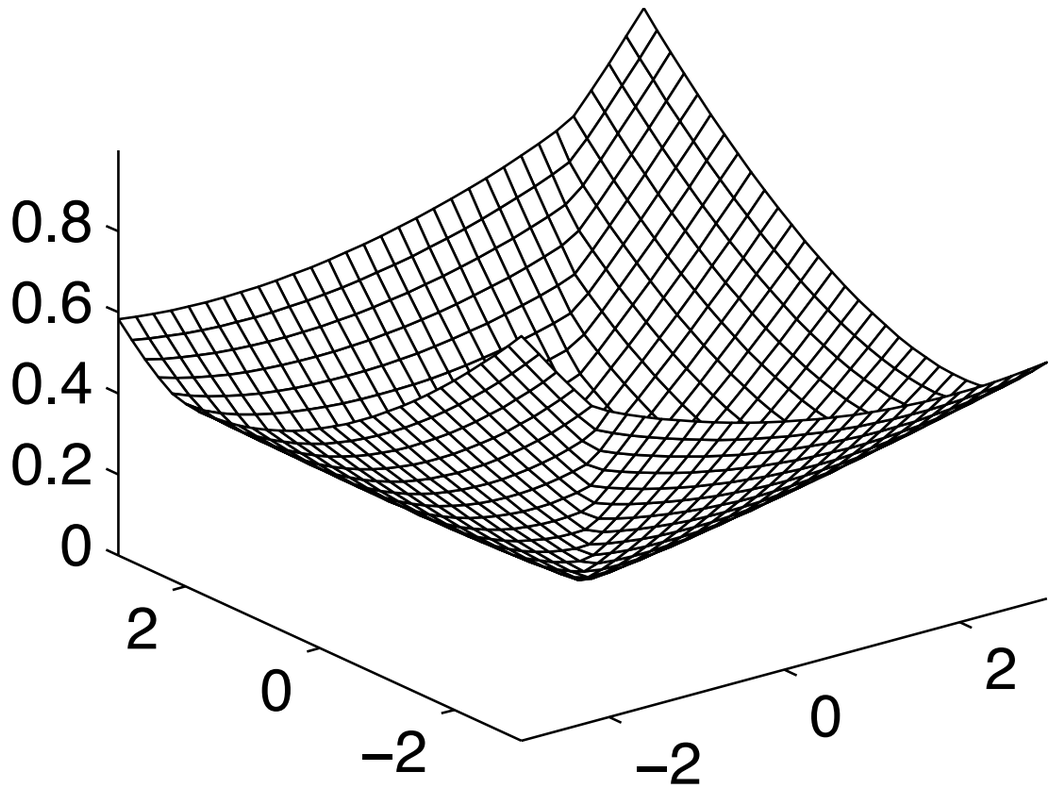


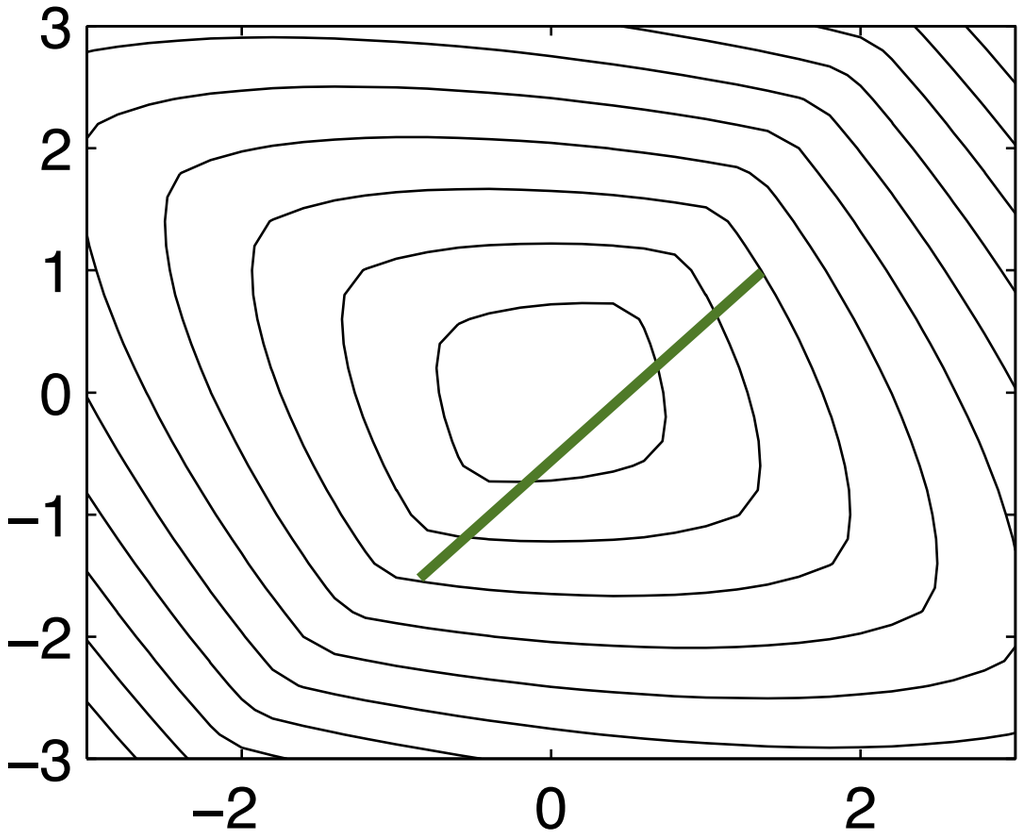



BLACK

Convexity 101

Convexity 101





- Convex set

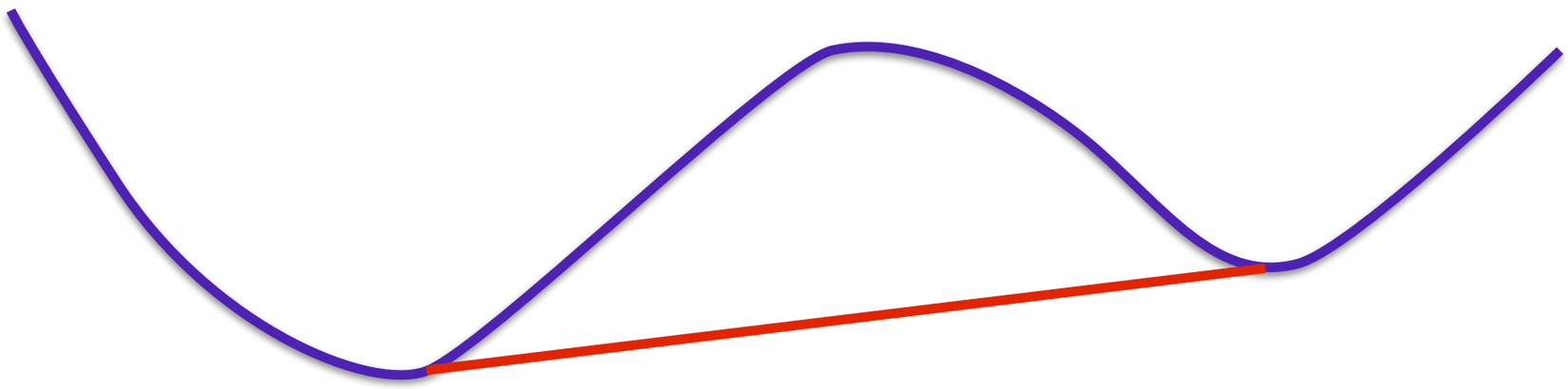
For $x, x' \in X$ it follows that $\lambda x + (1-\lambda)x' \in X$ for $\lambda \in [0, 1]$

- Convex function

$$\lambda\lambda f(x) + (1-\lambda) f(x') \geq f(\lambda x + (1-\lambda)x') \text{ for } \lambda \in [0, 1]$$

Convexity 101

- Any local minima is also global
  - Proof by contradiction: linear interpolation breaks local minimum condition

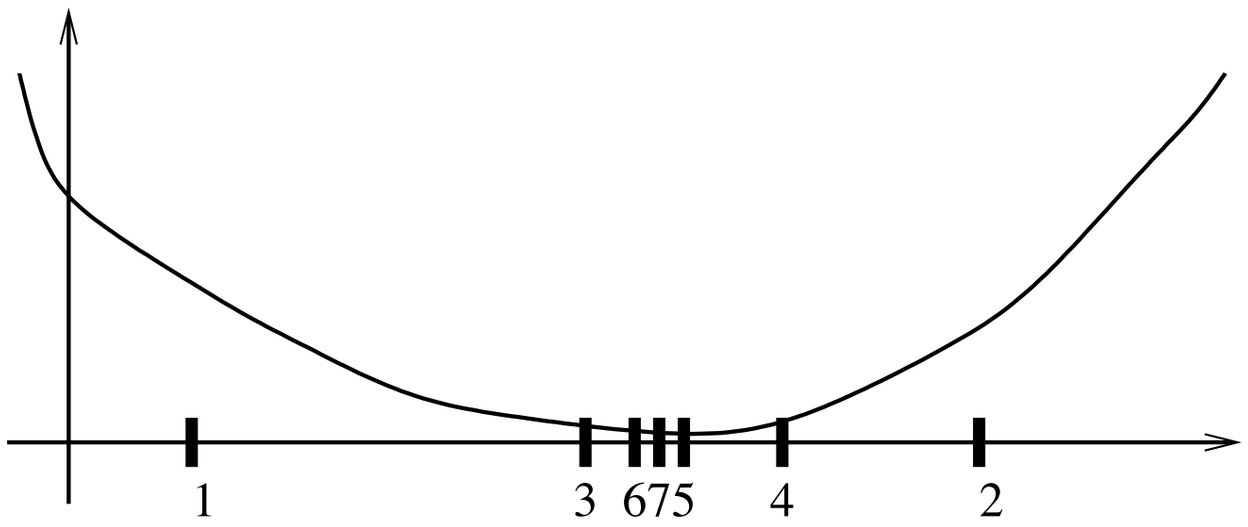


- Not true for non-convex functions
  - do not be afraid of it, a local minima is often good enough for practice (e.g. neural network)

Gradient descent

One dimensional problems

- search x for f’(x)=0
- direction: choose the interval by the sign of f’(x)
- step size: half of the interval

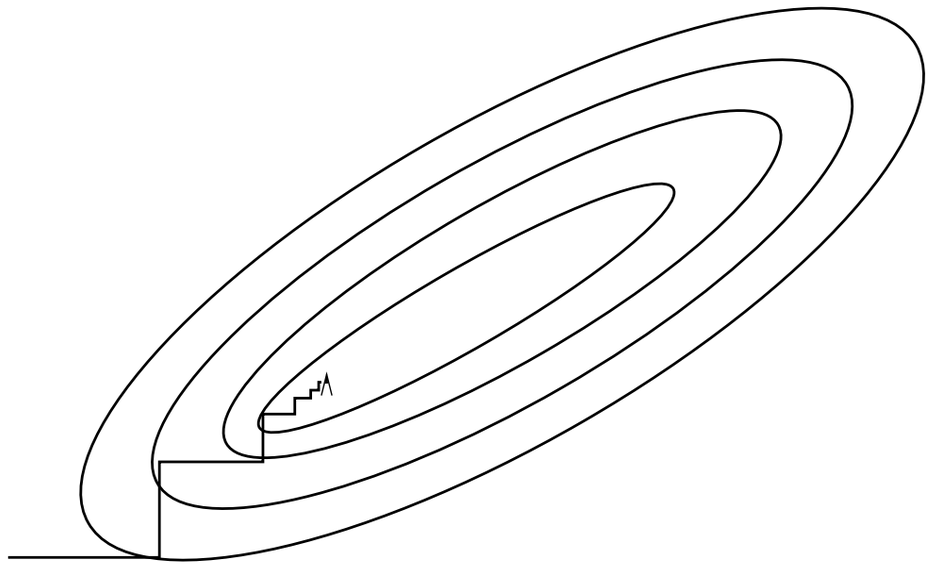


Require: a, b, Precision ϵ
  Set $A = a, B = b$
  repeat
    if $f'\left(\frac{A+B}{2}\right) > 0$ **then**
      $B = \frac{A+B}{2}$
    else
      $A = \frac{A+B}{2}$
    end if
  until $(B - A)\min(|f'(A)|, |f'(B)|) \leq \epsilon$
Output: $x = \frac{A+B}{2}$

Gradient descent

$$w_{t+1} = w_t - \eta_t \nabla f(w_t) \text{ for } t = 0, 1, \ldots$$

- Find ∇f(w) = 0
- Direction: the gradient
- Step size: line search



When f is Lipschitz

- L-Lipschitz: how smooth a function is

$$|f(x) - f(y)| \leq L\|x - y\|_2$$

  - choose a decay learning rate

$$\eta_t = 1/\sqrt{t}$$

  - convergence to ϵ-good solution

$$\|w_t - w^*\| \leq \epsilon$$

  - needs $O(1/\epsilon^2)$ iterations

∇f is Lipschitz

- L- gradient Lipschitz: how smoothly the gradient changes

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$$

- ★ constant learning rate $\eta_t \leq 1/L$
- ★ convergence to ϵ-good solution in $O(1/\epsilon)$ iterations

Backtracking

- L is often unknown
  - For each iteration t, fix α, β, start with η=1

$$f(w_t - \eta \nabla f(w_t)) \leq f(w_t) - \alpha\eta \|\nabla f(w_t)\|_2^2$$

  - if not true, set η=βη
- The minimal η obtained is min{1, β/L}

Strongly convex f

- M-strongly convex: how fast a function grows

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{M}{2}\|y - x\|_2^2$$

  - choose constant learning rate $\eta_t \leq 2/(M + L)$
  - convergence to ϵ-good solution in iterations $O(\ln(1/\epsilon))$
- If not strongly convex?
  - try

$$f(w) + \frac{M}{2}\|w\|_2^2$$

Accelerated gradient descent

- Rather than w_t, use a compensate point

$$u = w_t + \frac{t-1}{t+2}(w_t - w_{t-1})$$

$$w_{t+1} = u - \eta_t \nabla f(u)$$

- If f L-Lipshitz, fix $\eta_t \leq 1/L$, then convergence to ϵ-good solution in $O(1/\sqrt{\epsilon})$ iterations
- Very active research topic in the past few years
- May sensitive to the learning rate

Issues for sparse data

- Some features are dense, while some sparse
- Fixed learning rate for all features may have problem
  - a large value: do not convergence for dense features
  - a small value: convergence slowly for sparse features

Adaptive Gradients

- Normalize the gradients via history

$$w^{t+1} = w^t - \eta \left[\sum_{k \le t} \mathrm{diag}(\nabla f(w^k))^2 \right]^{-\frac{1}{2}} \nabla f(w^t)$$

- More solutions:
  - Newton / Quasi-newton method
  - Coordinate descent

Multi-thread

Logistic Regression

- The most expensive part: compute gradients
- For logistic loss:

$$f(w) = \mathrm{sum}(\log(1 + \exp(y. * (Xw))$$

$$\nabla f(w) = X^T(y./(1 + \exp(y. * (Xw))))$$

- Two key steps

$$x = A^T y \qquad y = Ax$$

- Multi-threaded by previous technologies

Parameter Server demo

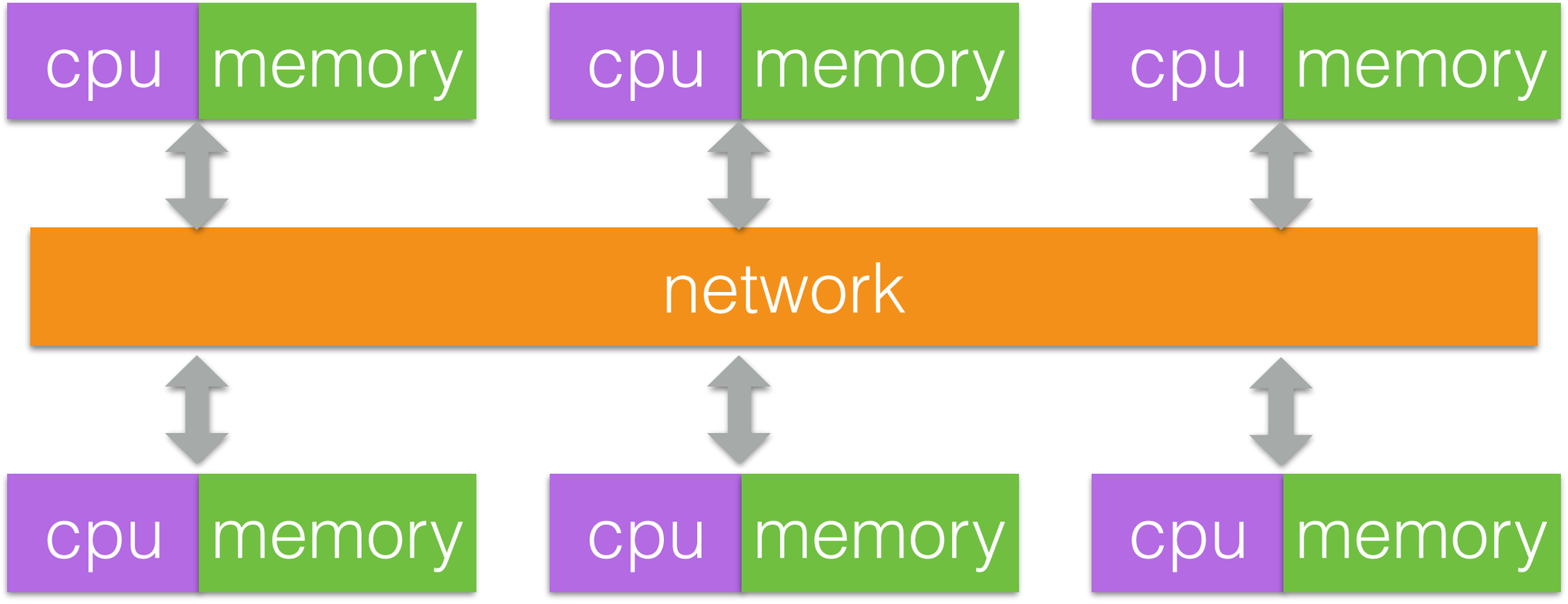
- require gcc >= 4.7 (linux) or llvm>=3.4 (mac)
- goto https://github.com/mli/parameter_server or google “parameter server github”
- goto bottom “wrap up” section:

```
mkdir your_working_dir
cd your_working_dir
git clone git@github.com:mli/parameter_server.git .
git clone git@github.com:mli/parameter_server_third_party.git third_party
git clone git@github.com:mli/parameter_server_data.git data
third_party/install.sh
cd src && make -j8
mpirun -np 4 ./ps_mpi -num_servers 1 -num_workers 2 -app ../config/rcv1_l1lr.config
```

Distributed Implementation

Distributed architecture

- Data are exchange via network



Data center

3-level fat-tree
432 servers (gray)
180 switches (red)

by Brighten Godfrey

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

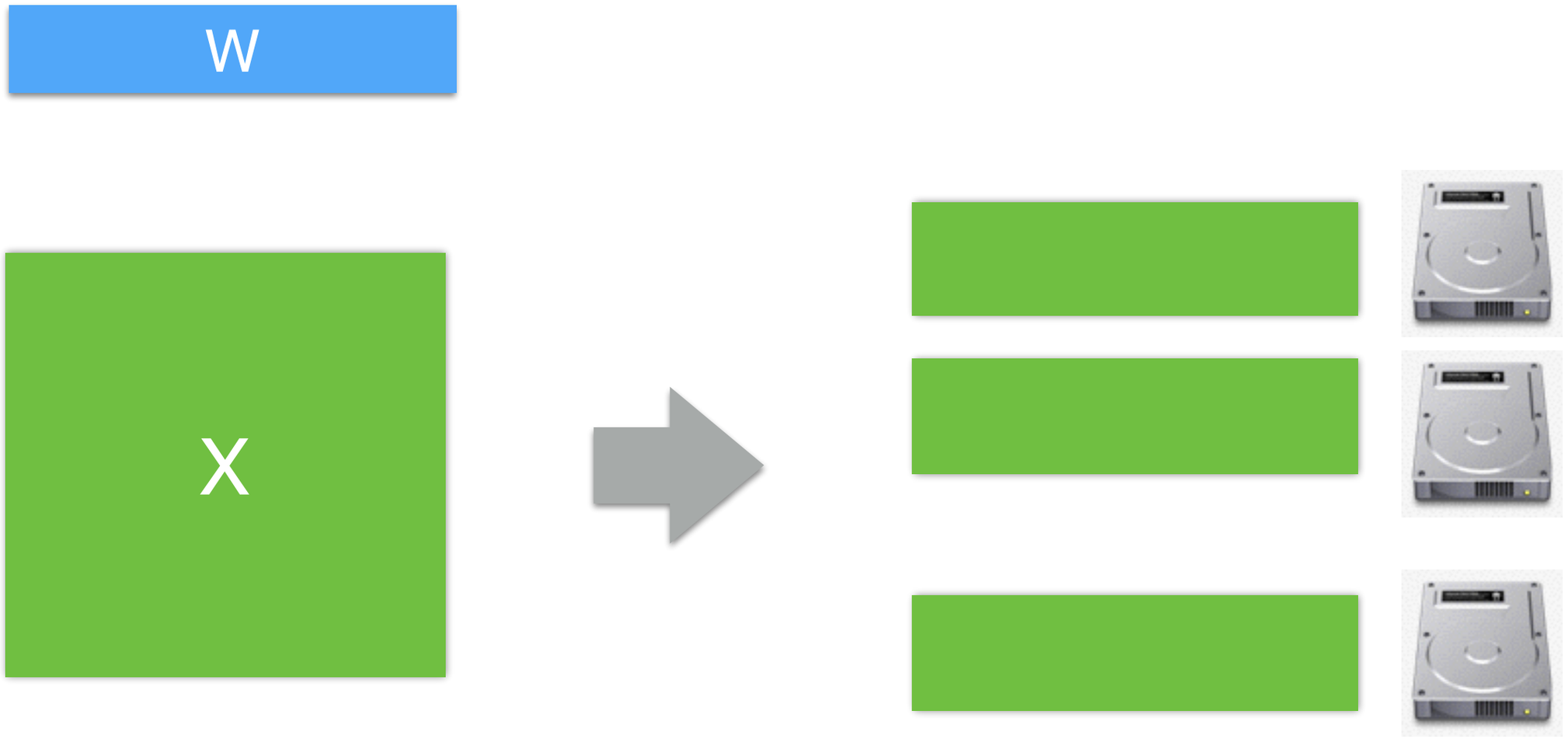
500 times

slides by Jeff Dean

Google

Data partition

- Assume w is not so large



Distributed Implementation

Partition data into machines

Get the working set of *w*

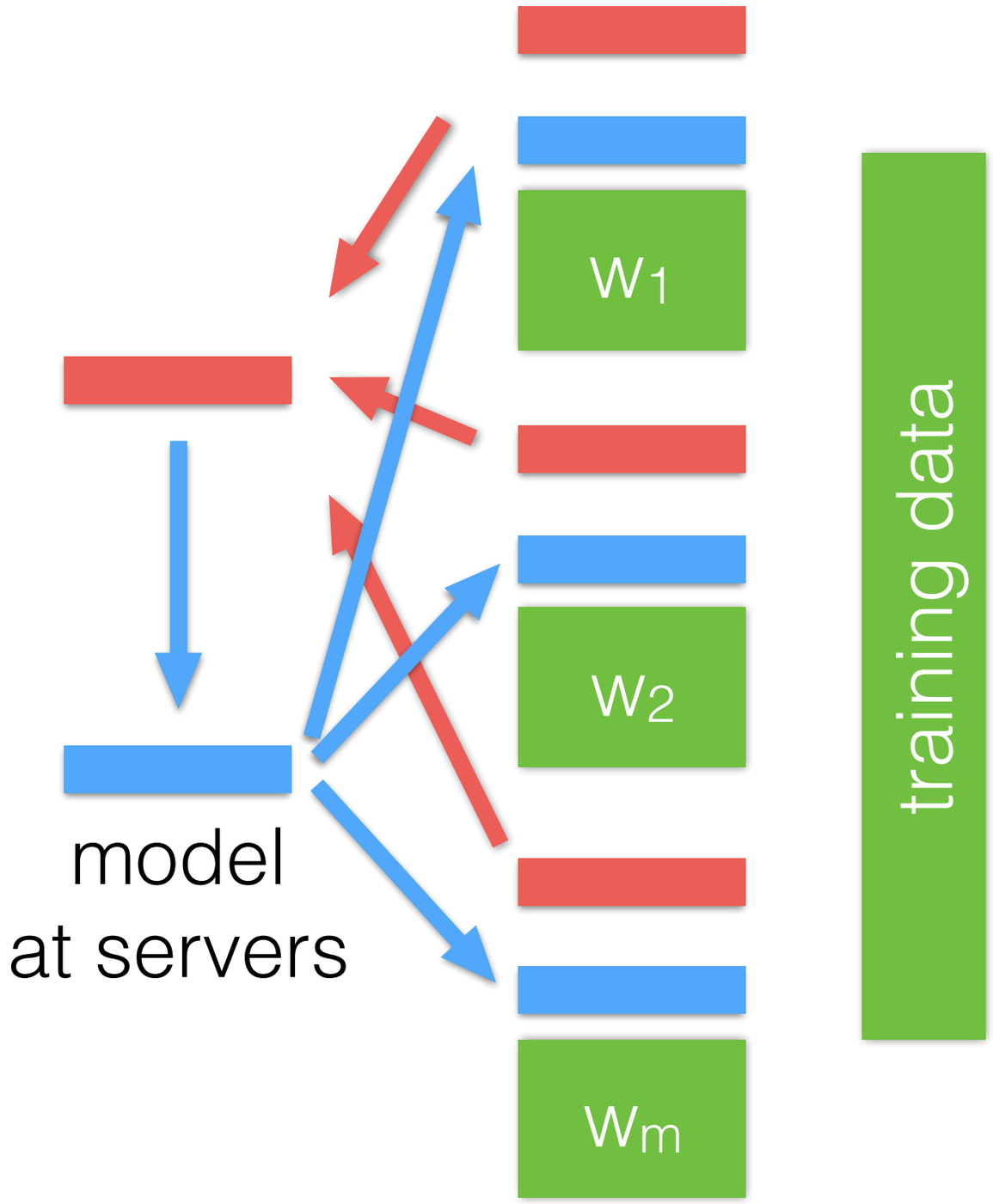
Iterate *t = 1, 2, …*

compute gradients

aggregate gradients

update *w*

get updated *w*



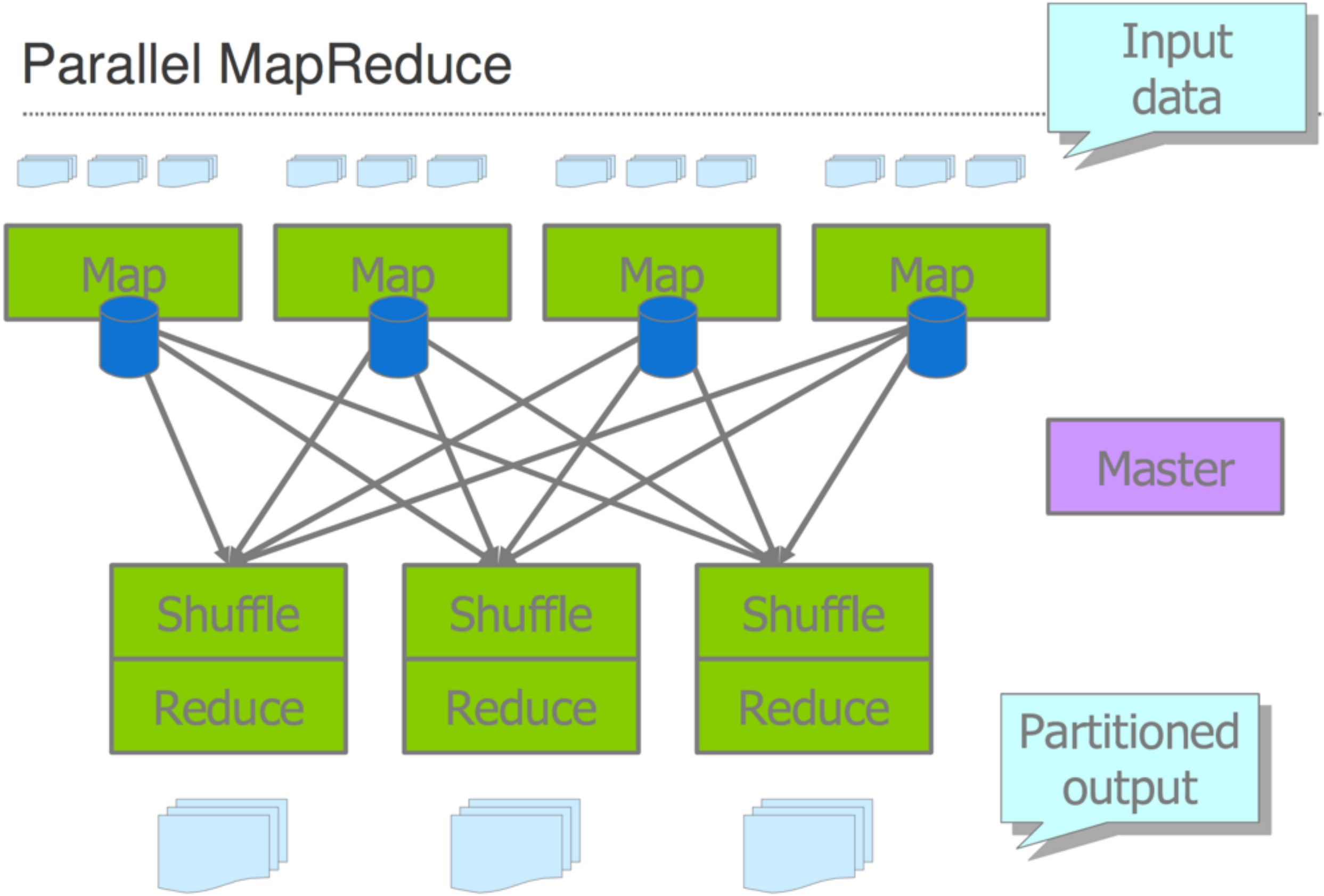
MapReduce

- Proposed by Google, open source Hadoop, Spark, …
- Need to specify two primary methods
  - map (k, v) —> [k’, v’]*
  - reduce (k, [v]*) —> [k,v’]*

Implement GD

- Each iteration is a map/reduce
  - map: each worker computes gradients, emits (feature_id, gradient_value)
  - reduce: sum gradient with the same featured id

Parallel MapReduce



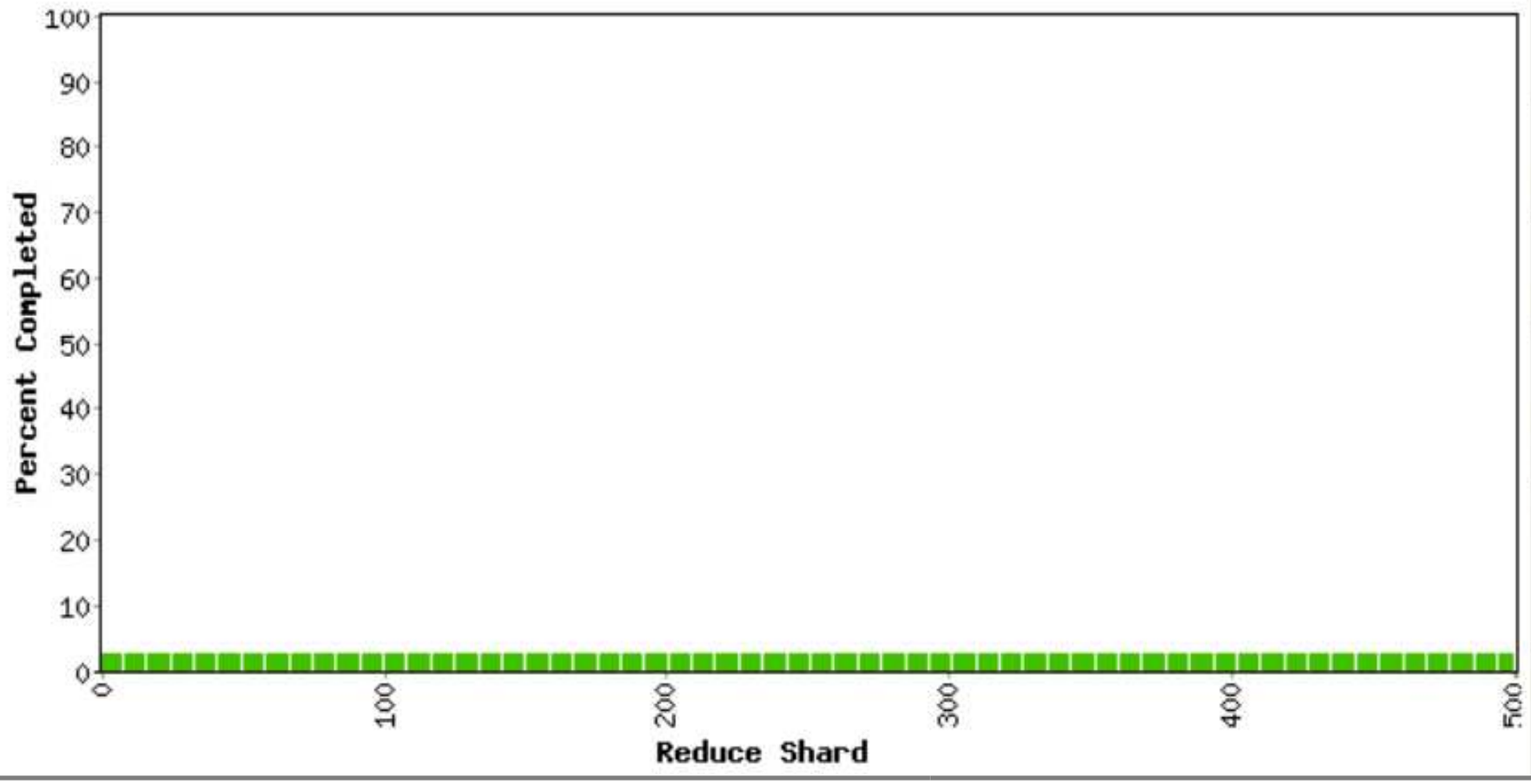
by Jeff Dean

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

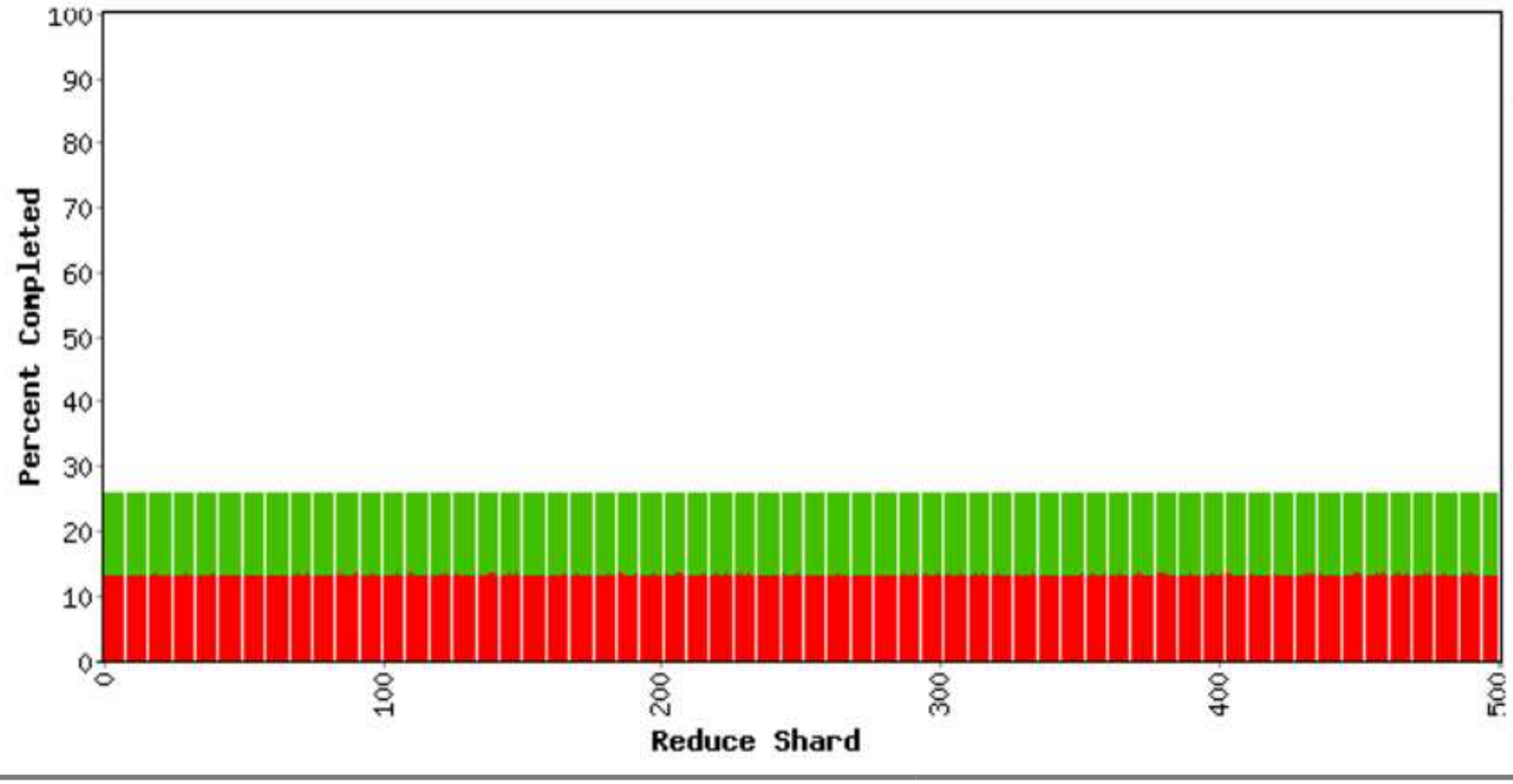
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

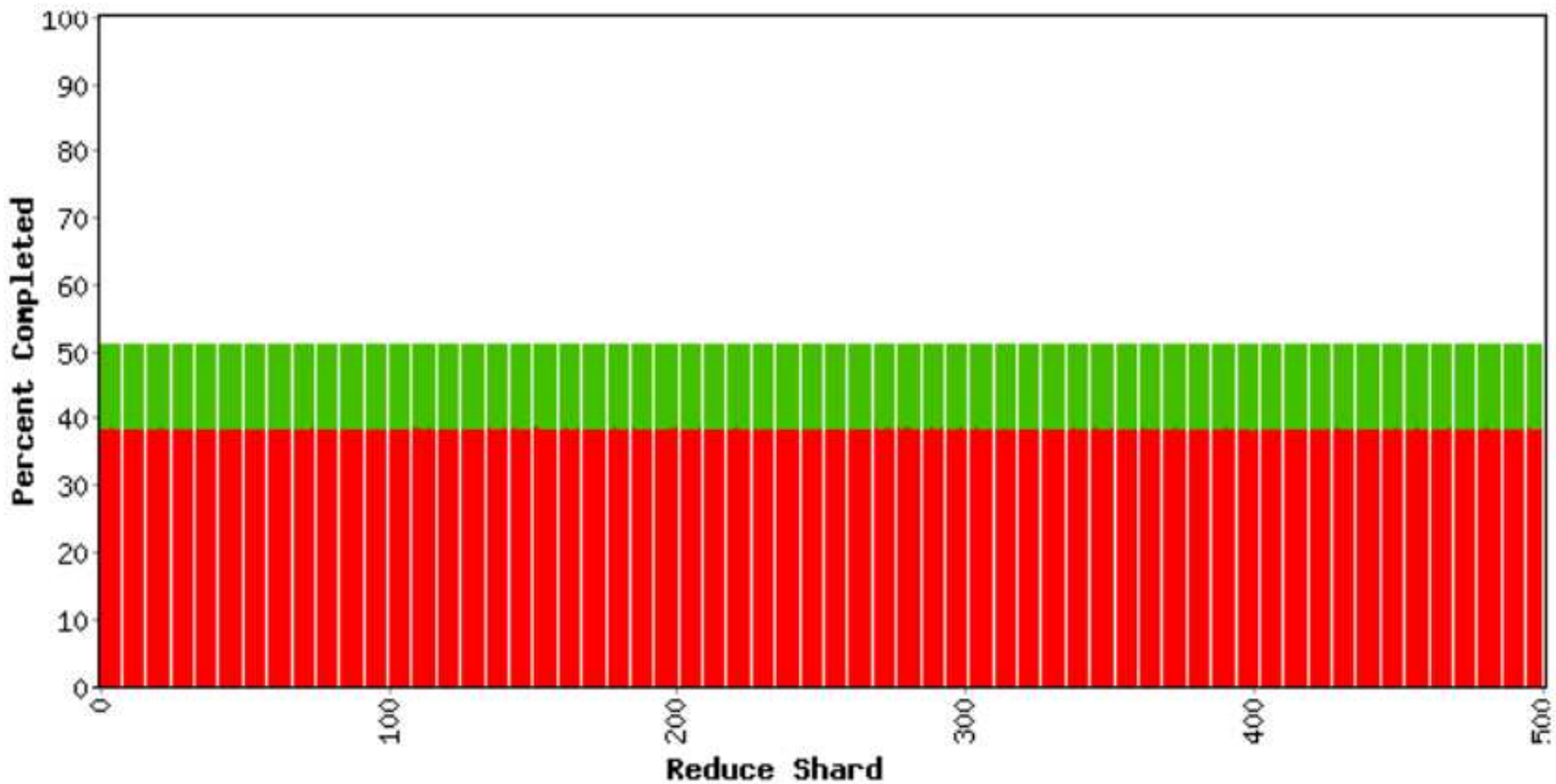
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

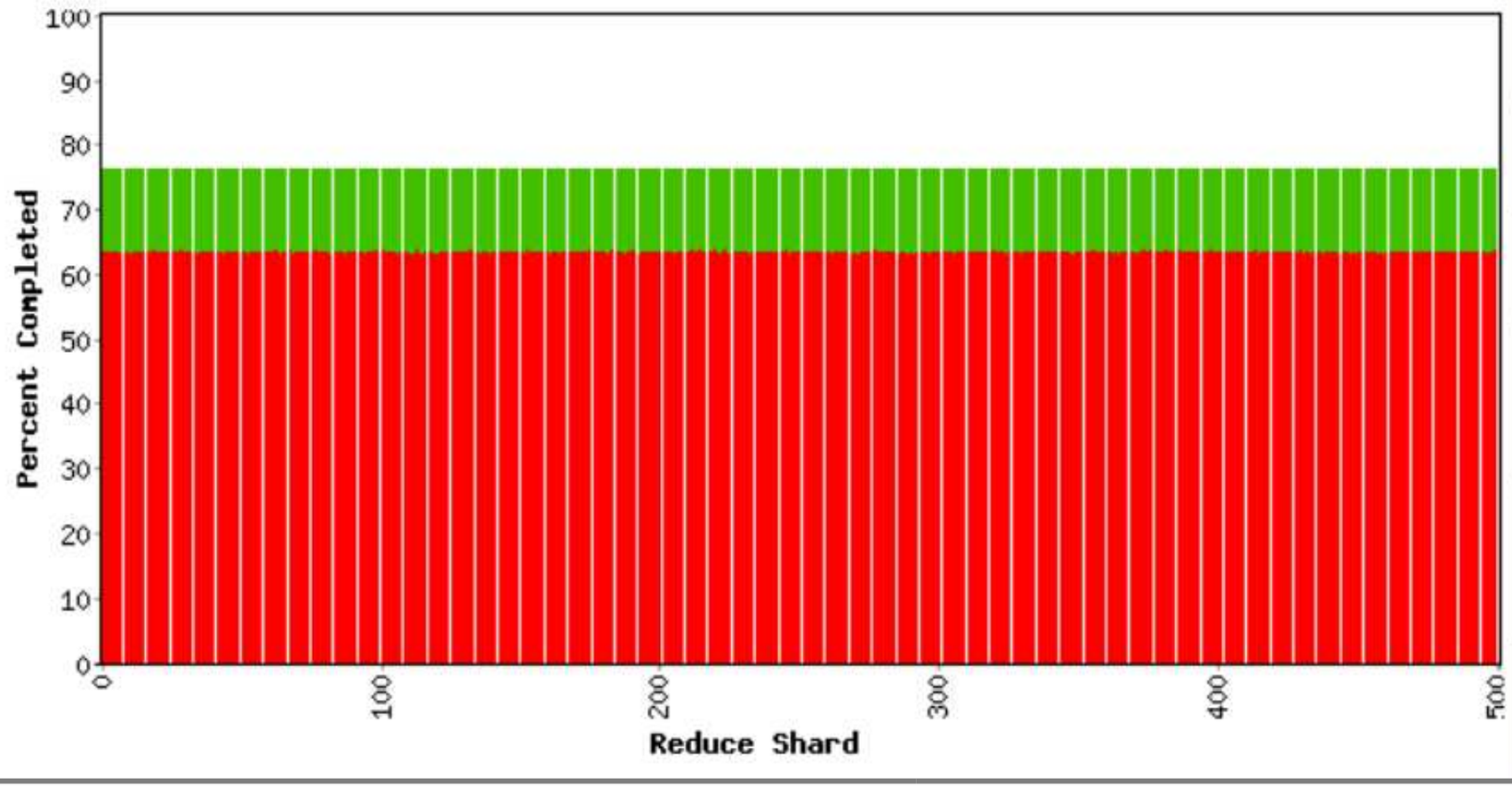
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

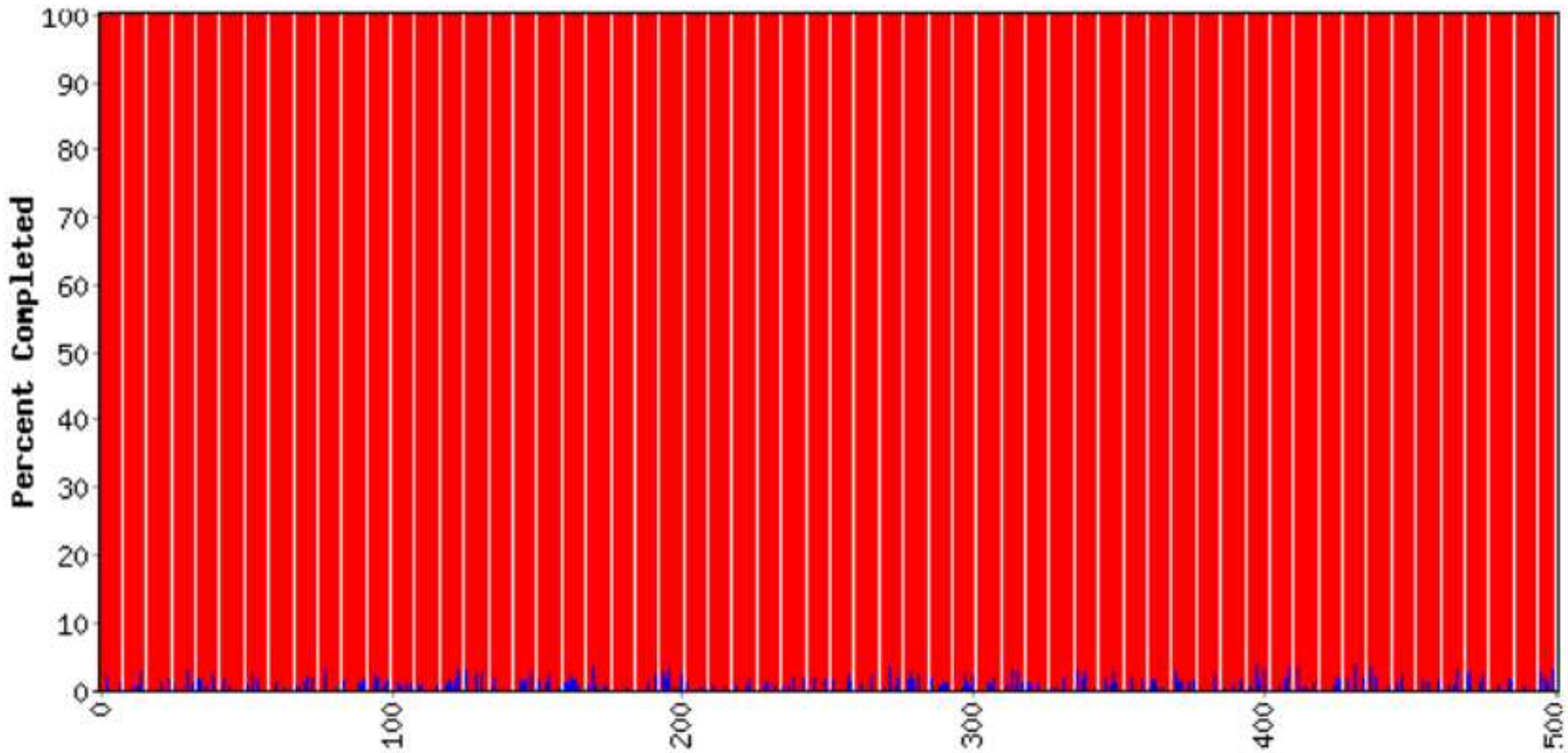
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

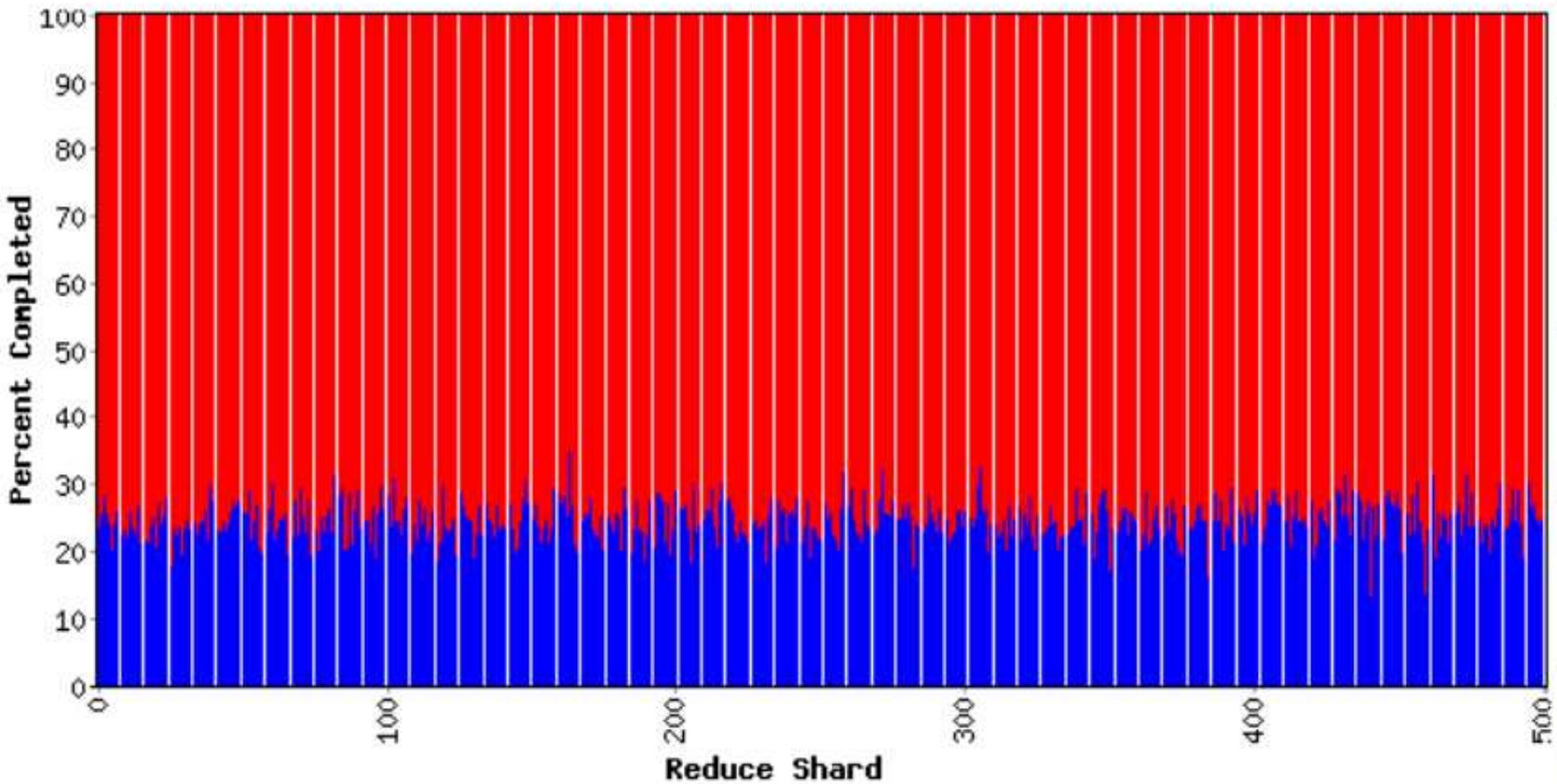
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

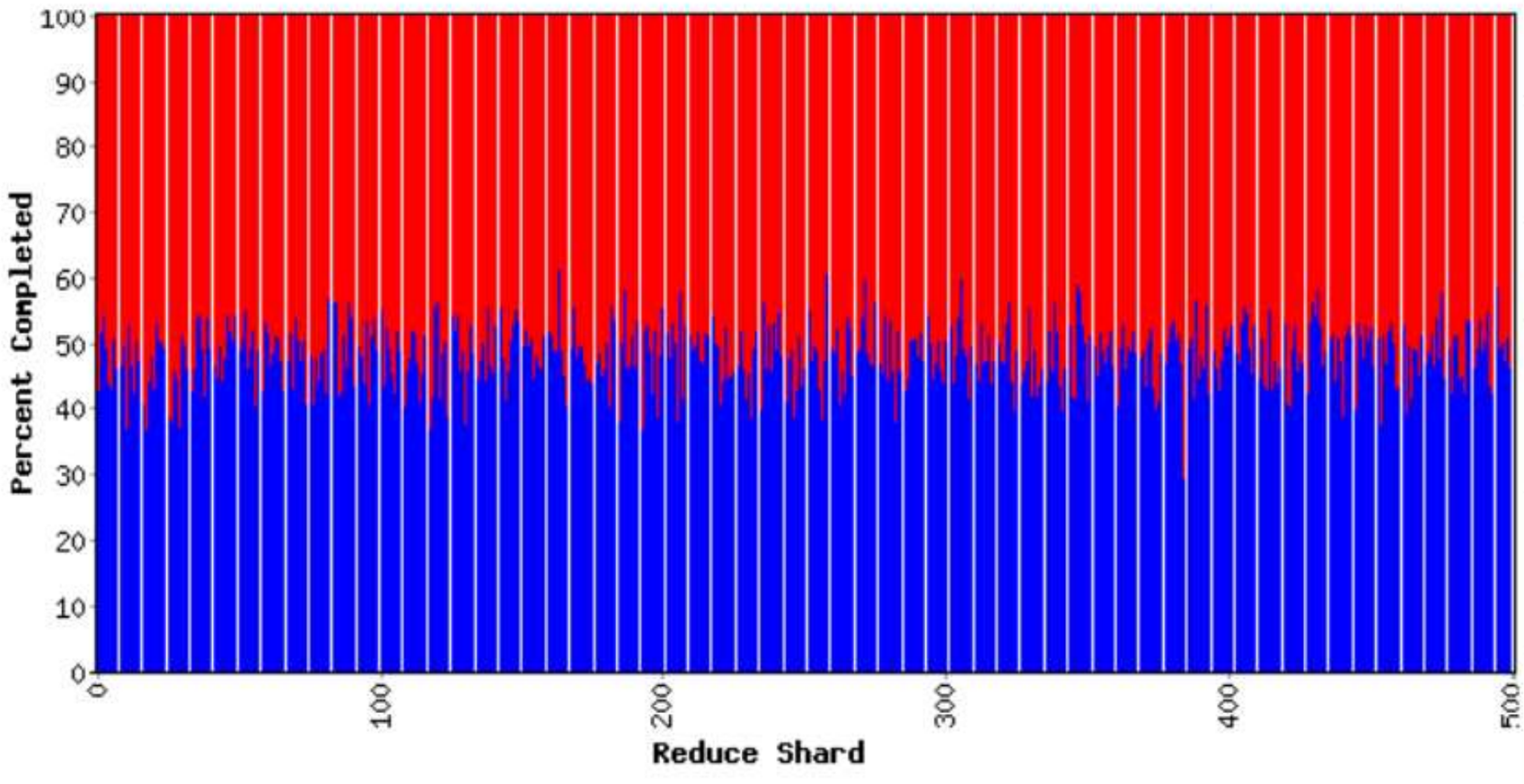
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1225.1
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

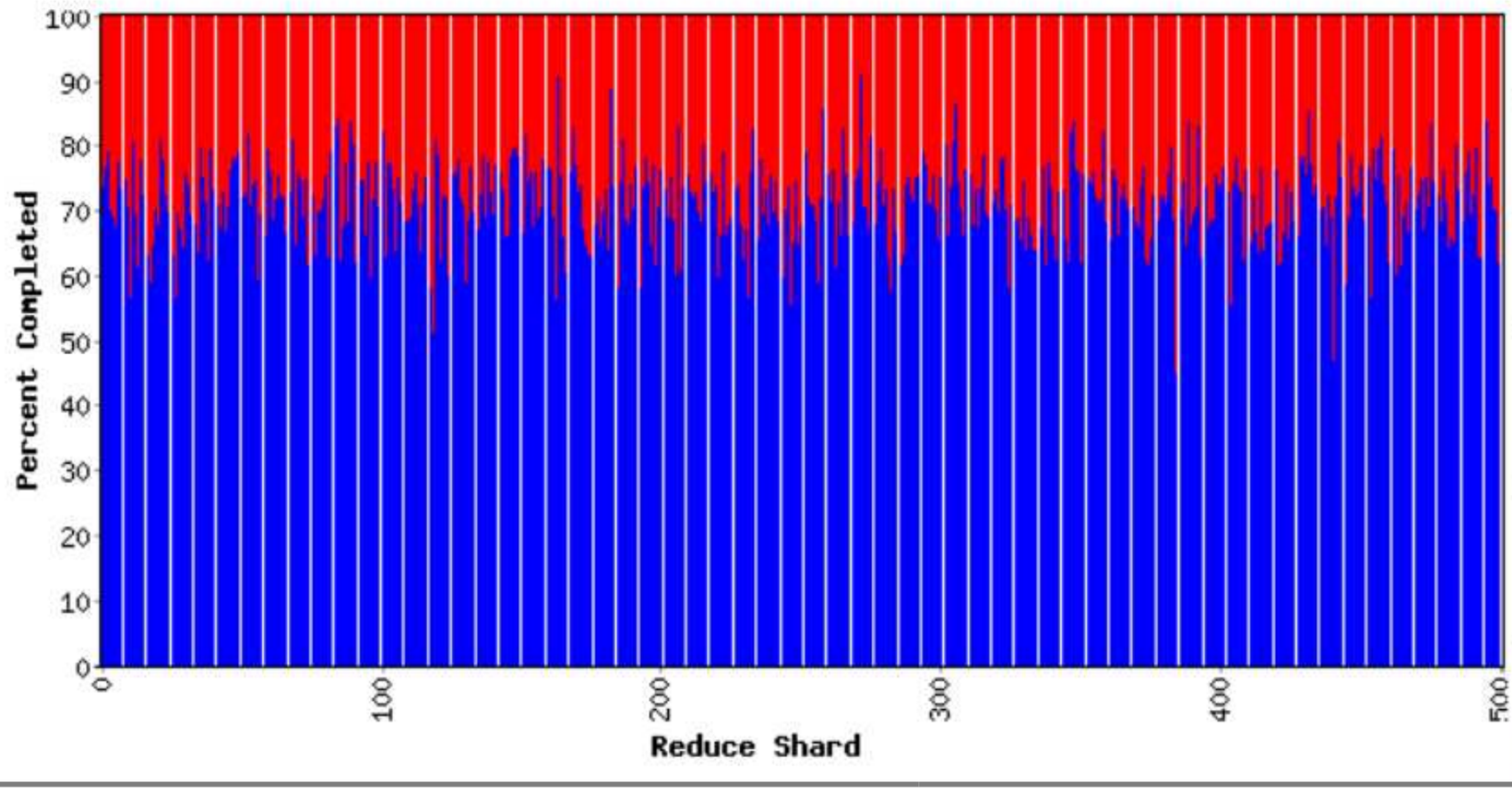
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1222.0	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51640600	
mr-merge-outputs	51640600	

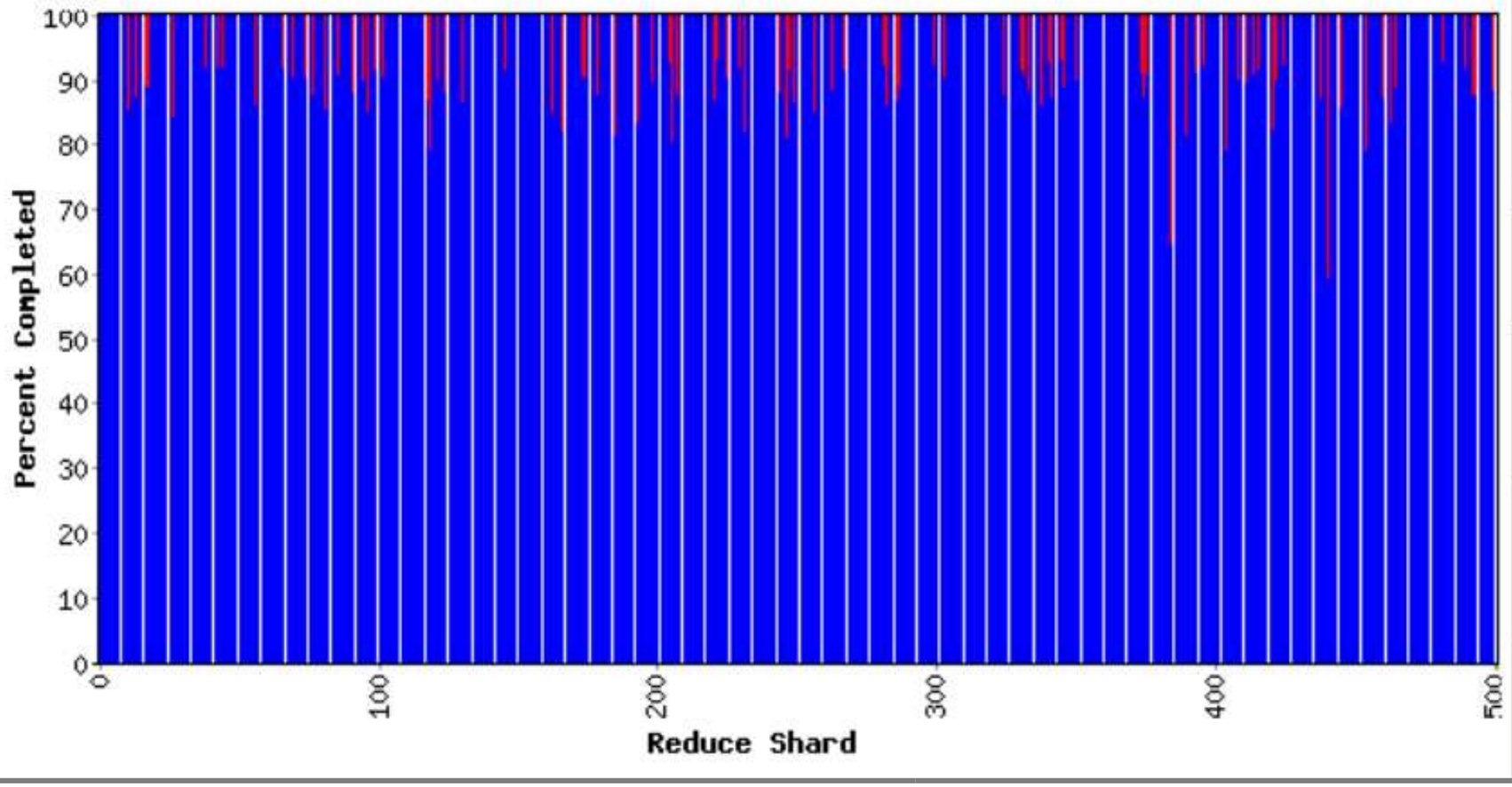
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

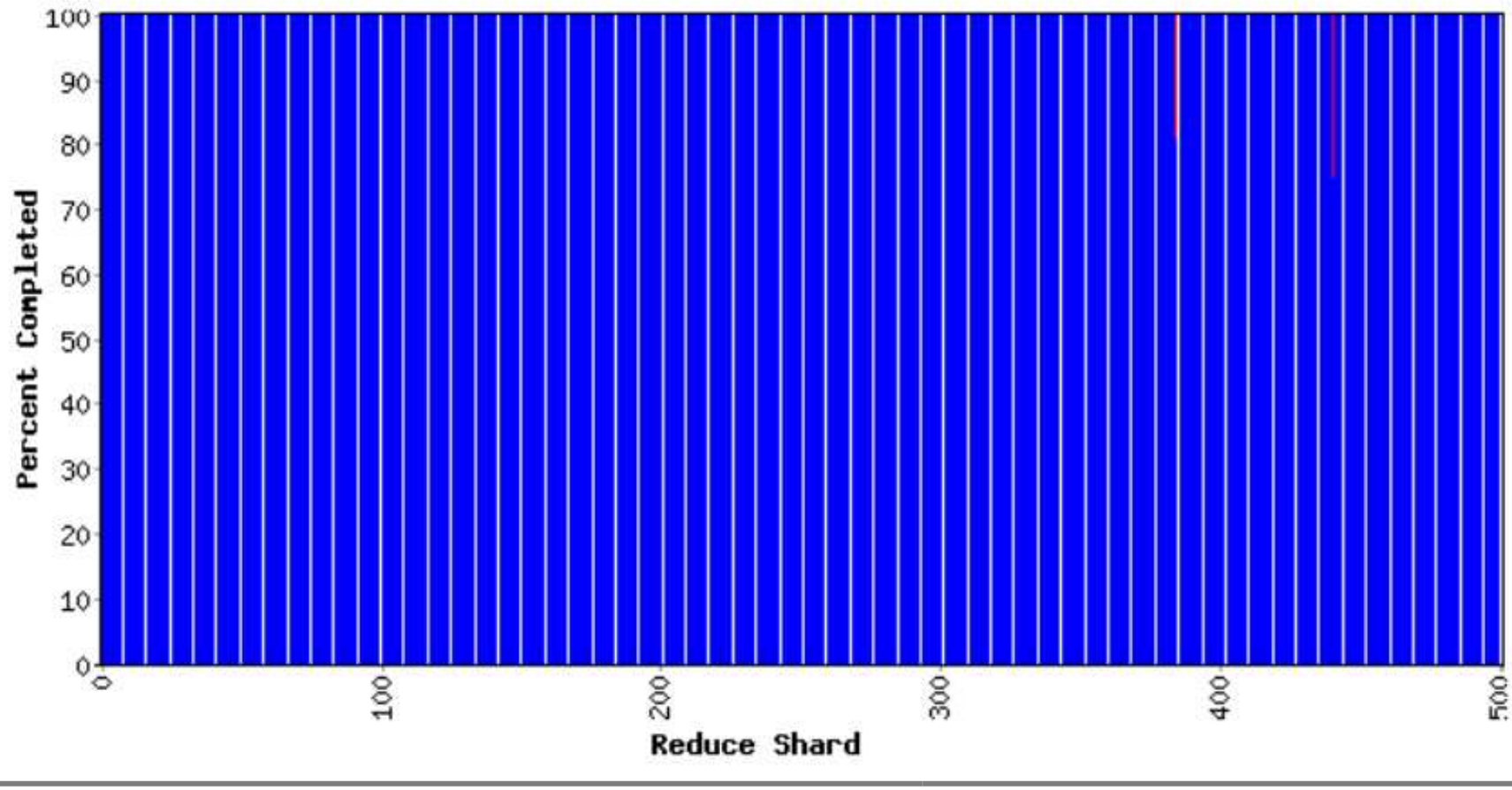
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	394792	
mr-merge-outputs	394792	

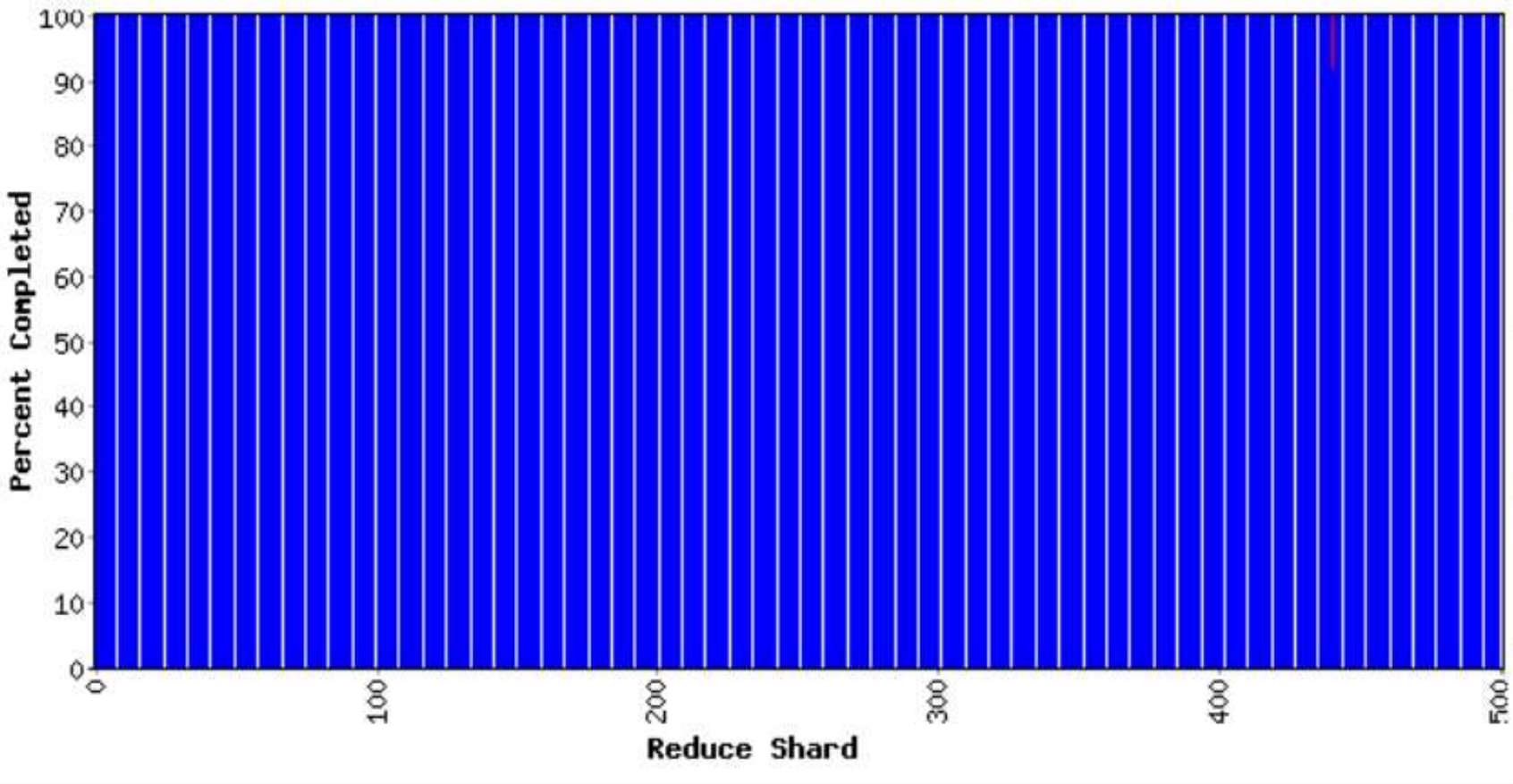
Google

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Google

Practice Guide

- Use #map >> #reduce > #machines
  - fine granularity tasks
  - better load balance
- Do not use it if you have a lot of iterations, 10 is fine, 100 is terrible
  - good for feature extraction
  - often not good for optimization

MPI

- A message passing interface
- Proposed and implemented by HPC community
- OpenMPI: http://www.open-mpi.org/
- mpich: http://www.mpich.org/

Basic model

- Group
  - ordered set of processes
  - rank from 0 to N-1 (for N processes)
- Communicator
  - group of processes that can communicate with each other
  - default communicator: MPI_COMM_WORLD

A MPI program structure

- #include “mpi.h”
- initialize MPI environment: MPI_Init
- MPI_xxx()
- MPI_Finalize()

```
int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  int my_rank, rank_size;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &rank_size);
  printf("# of processses %d, my rank %d\n", rank_size, my_rank);
  MPI_Finalize();
}
```

Collective Routines

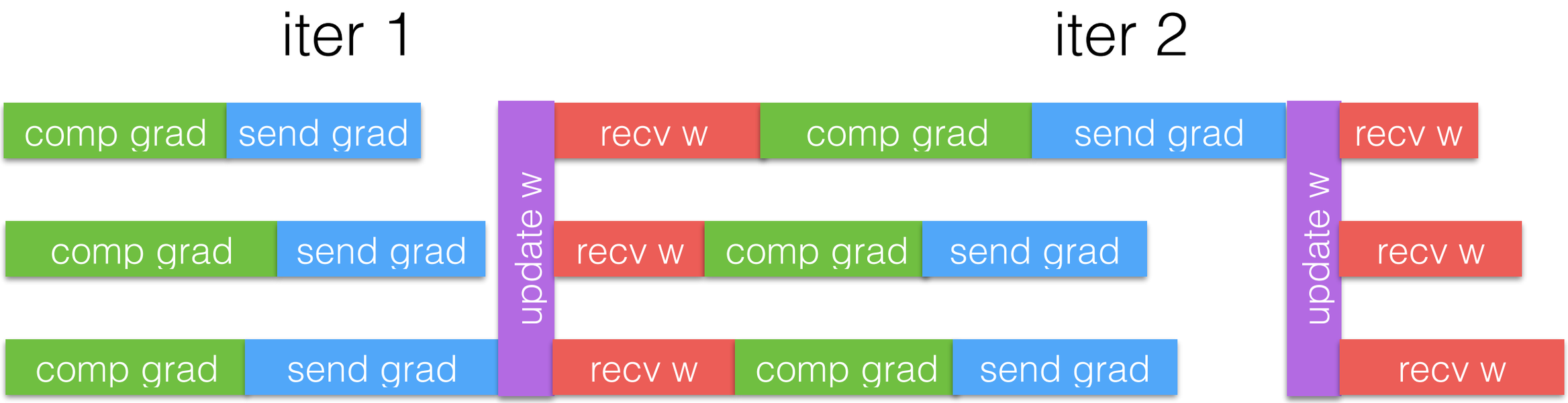
- reduce the gradients

```
MPI_Reduce(local_grad.data(),
           global_grad->data(),
           feature_size,
           MPI_DOUBLE,
           MPI_SUM,
           0,
           MPI_COMM_WORLD);
```

- broadcast the weight

```
MPI_Bcast(weight.data(),
          feature_size,
          MPI_DOUBLE,
          0,
          MPI_COMM_WORLD);
```

Performance Analysis



- CPU is idle when
  - send gradient and receive weight
  - waiting all machines are done
- We should parallel CPU and I/O

Practice Guide

- MPI is efficient
  - the model size is not too large
  - dense communication
  - assume reliable machines
  - only you are using these machines
- Quite difficult to program if
  - a huge model required model partition
  - sparse communication

More choices

- Graphlab: http://graphlab.com/
  - present the sparse data as a graph
- Spark: http://spark.apache.org/
  - in-memory hadoop
- REEF
  - will be presented on next week
- …

Parameter Server demo

- require gcc >= 4.7 (linux) or llvm>=3.4 (mac)
- goto https://github.com/mli/parameter_server or google “parameter server github”
- goto bottom “wrap up” section:

```
mkdir your_working_dir
cd your_working_dir
git clone git@github.com:mli/parameter_server.git .
git clone git@github.com:mli/parameter_server_third_party.git third_party
git clone git@github.com:mli/parameter_server_data.git data
third_party/install.sh
cd src && make -j8
mpirun -np 4 ./ps_mpi -num_servers 1 -num_workers 2 -app ../config/rcv1_l1lr.config
```

Newton's Method

Newton Method

- Convex objective function f
- Nonnegative second derivative

$$\partial_x^2 f(x) \succeq 0$$

- Taylor expansion

$$f(x+\delta) = f(x) + \langle \delta, \partial_x f(x) \rangle + \frac{1}{2}\delta^\top \partial_x^2 f(x)\delta + O(\delta^3)$$

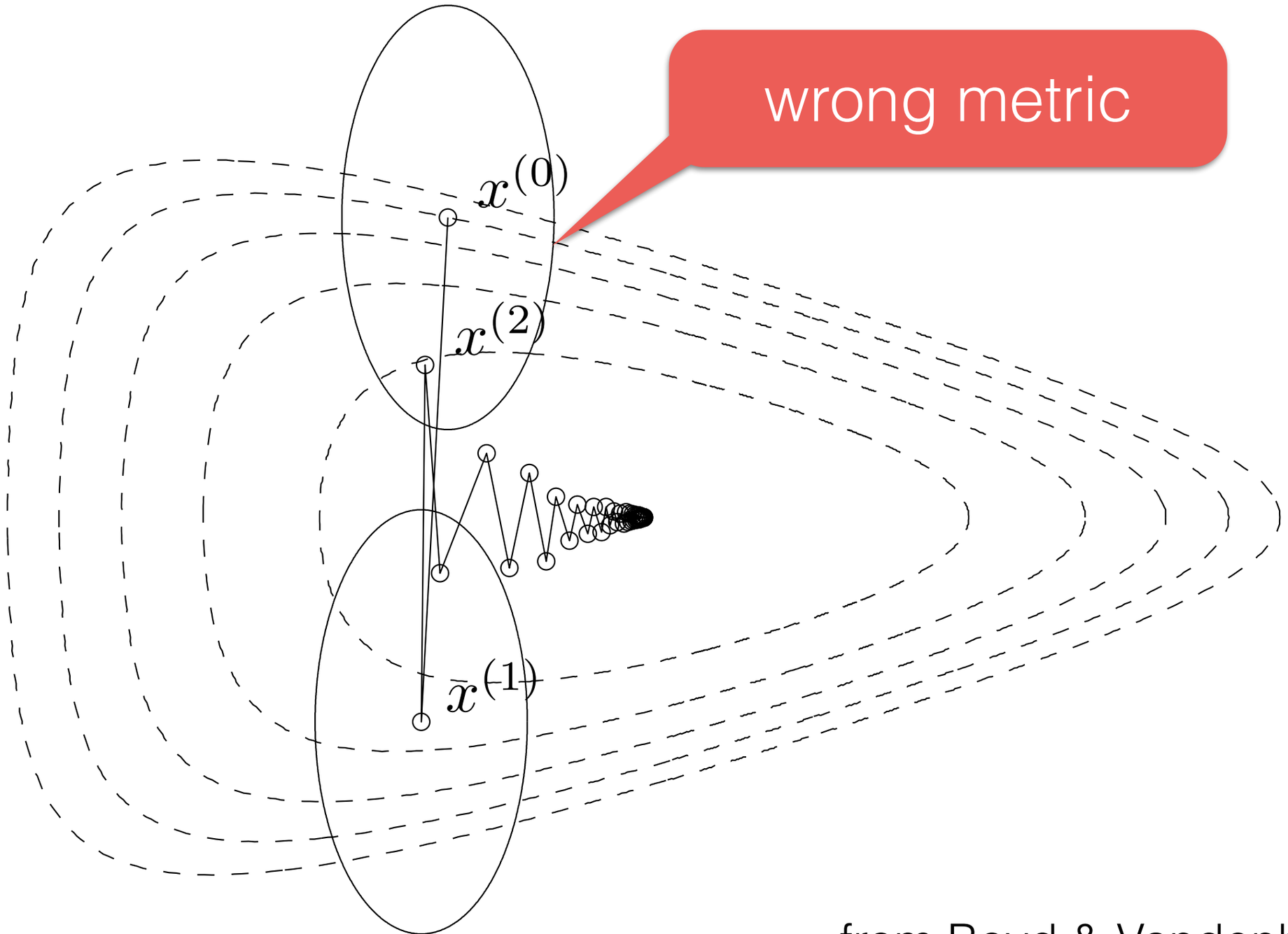
gradient

Hessian

- iterate until converged

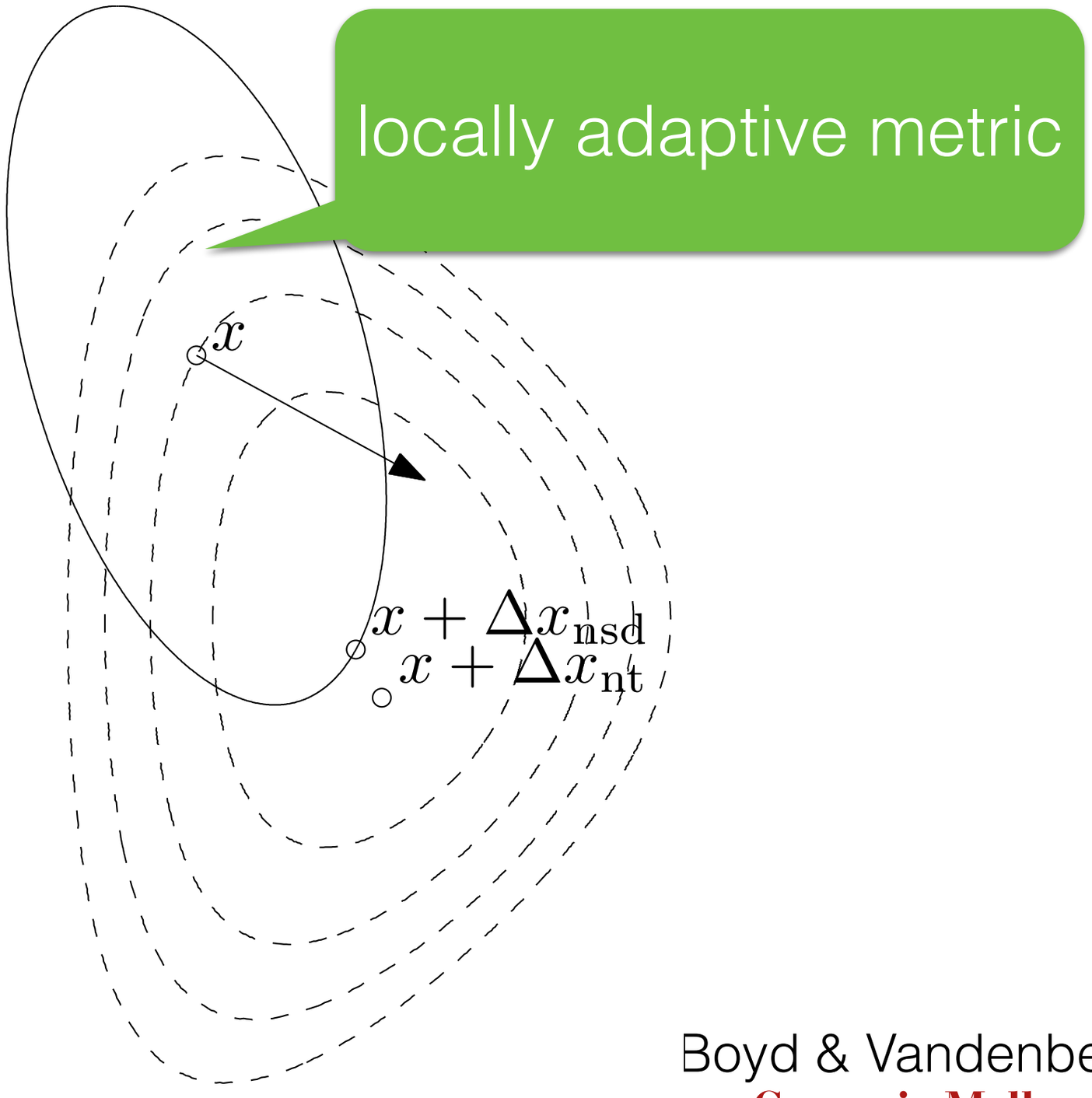
$$x \leftarrow x - \left[\partial_x^2 f(x)\right]^{-1} \partial_x f(x)$$

Newton method rescales space



from Boyd & Vandenberghe

Newton method rescales space



Boyd & Vandenberghe

Carnegie Mellon University

Parallel Newton Method

- Good rate of convergence
- Few passes through data needed
- Parallel aggregation of gradient and Hessian
- Gradient requires O(d) data
- Hessian requires O(d2) data
- Update step is O(d3) & nontrivial to parallelize
- Use it only for low dimensional problems

BFGS algorithm

Basic Idea

- Newton-like method to compute descent direction

$$\delta_i = B_i^{-1} \partial_x f(x_{i-1})$$

- Line search on f in direction

$$x_{i+1} = x_i - \alpha_i \delta_i$$

- Update B with rank 2 matrix

$$B_{i+1} = B_i + u_i u_i^\top + v_i v_i^\top$$

- Require that Quasi-Newton condition holds

$$B_{i+1}(x_{i+1} - x_i) = \partial_x f(x_{i+1}) - \partial_x f(x_i)$$

$$B_{i+1} = B_i + \frac{g_i g_i^\top}{\alpha_i \delta_i^\top g_i} - \frac{B_i \delta_i \delta_i^\top B_i}{\delta_i^\top B_i \delta_i}$$

Properties

- Simple rank 2 update for B
- Use matrix inversion lemma to update inverse
- Memory-limited versions L-BFGS
- Use toolbox if possible (TAO, MATLAB) (typically slower if you implement it yourself)
- Works well for nonlinear nonconvex objectives (often even for nonsmooth objectives)

Coordinate Descent

Basic Idea

- Update one feature each time
- Can solve this small problem by expensive method, say Newton method

for t = 1, 2, …
  pick up feature i (in random)
  solve $\min_{w_i} f(w)$ by fixing $w_1, \ldots, w_{i-1}, w_{i+1}, \ldots$

- Works very well for sparse data (liblinear)
- A sequential method

Shotgun

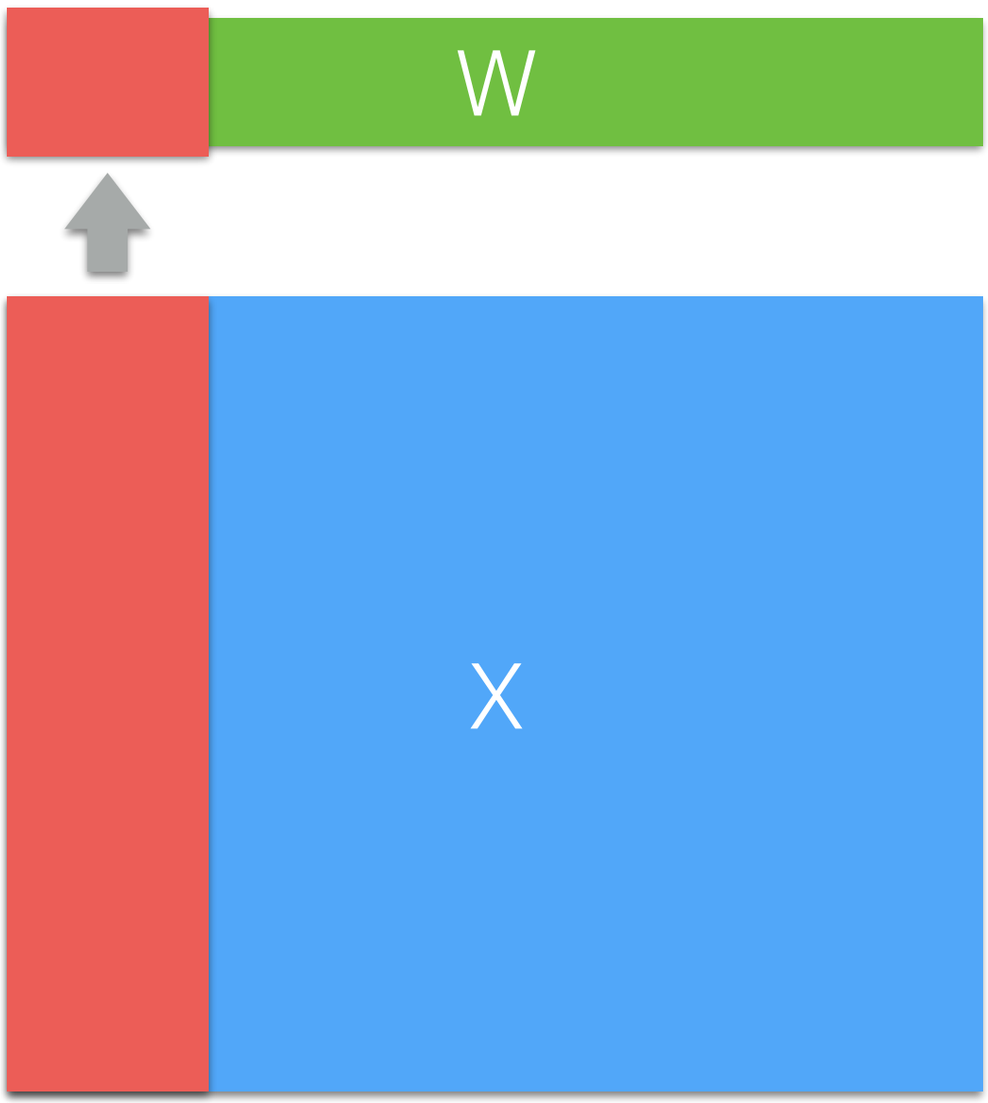
- Assume n threads, each thread updates one feature in parallel
- Works well if features are not so correlated:

$$\#\text{threads} = O\left(\frac{\#\text{features}}{\rho(X^T X)}\right)$$

- May suffer from large thread synchronization cost

Block coordinate descent

- Update a block of features simultaneously



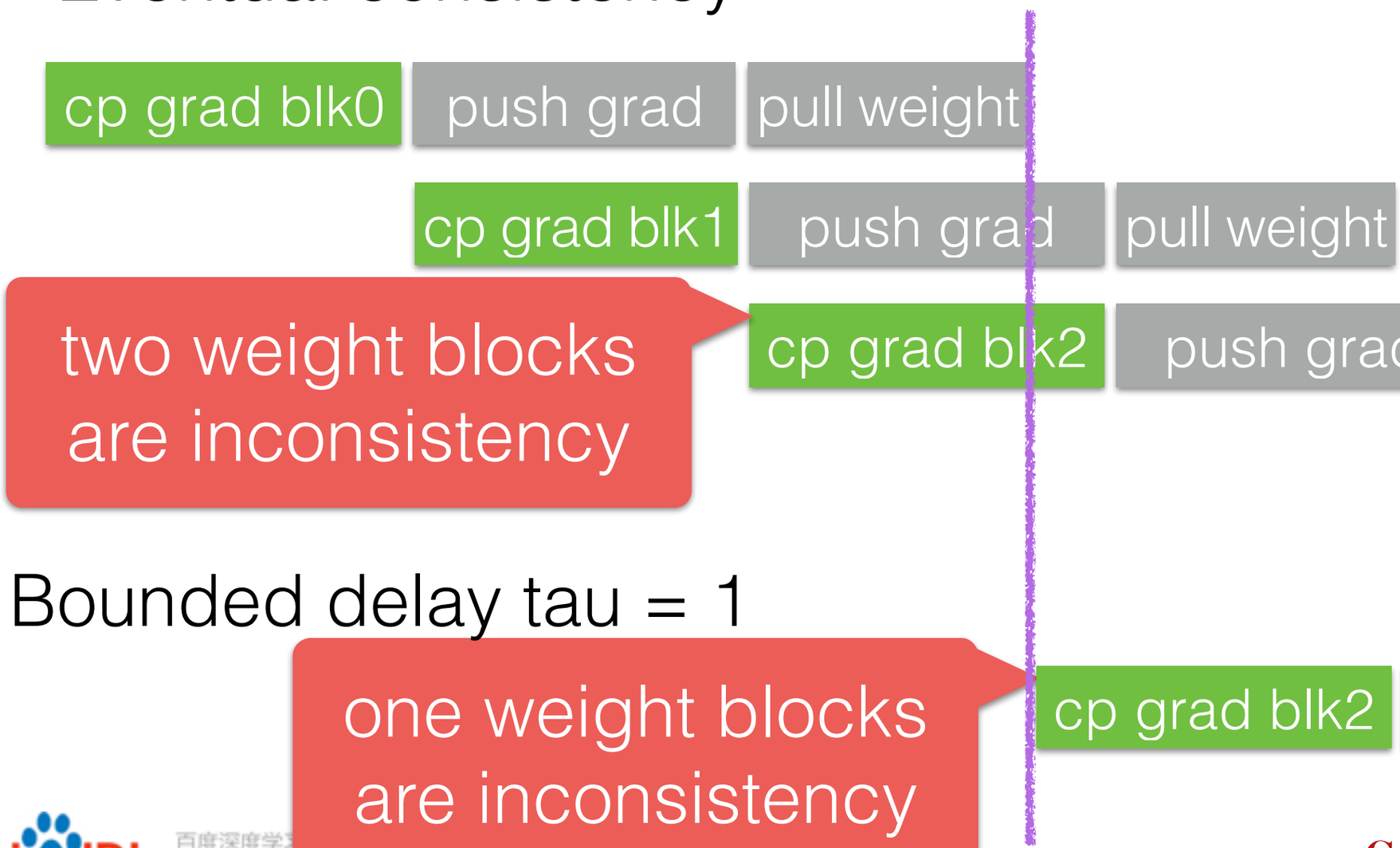
Implement Block CD

- Multi-threaded within in a block
- Easy to extend to a distributed implementation
- Works well if features in a block are less correlated
- Should use small learning rate to guarantee convergence

Asynchronous Updating

- Parallel CPU and I/O, hide synchronization cost
- A worker:

Eventual consistency



Bounded delay tau = 1

Analysis

- L_{var}: correlation of features in a block
- L_{cov}: correlation of features between too neighbor blocks
- τ-bounded delay
- fixed learning rate guarantee convergence (even for non-convex problems)

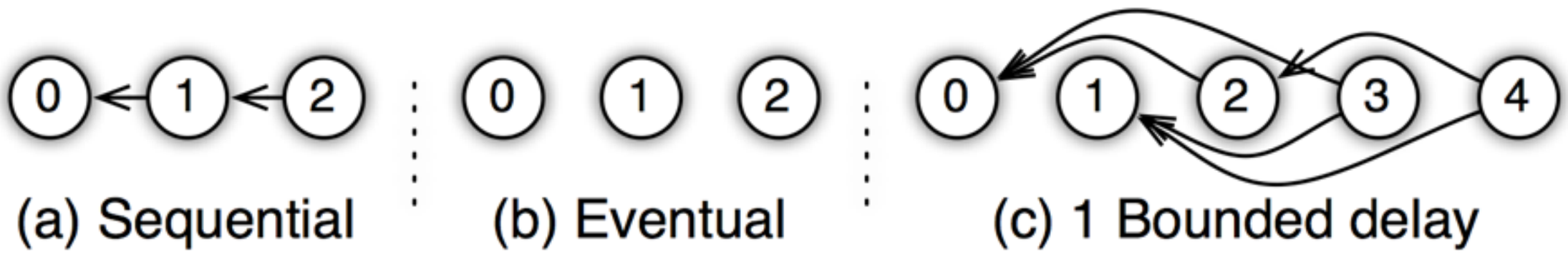
$$\eta_t = O\left(\frac{1}{L_{var} + \tau L_{cov}}\right)$$

make features in a block less correlated

make blocks less correlated

Implement via ParaServer

- Communication API:
  - range push and pull
- Task: a push, pull, or any user-defined function
  - one iteration (can contain push and pull)
- Task dependency graph:



Implement bounded-delay

- LinearBlockIterator::run()
  - executed at the scheduler

```
for (int iter = 0; iter < cf.max_pass_of_data(); ++iter) {
  std::random_shuffle(block_order.begin(), block_order.end());
  for (int b : block_order)  {
    Task update;
    update.set_wait_time(time - tau);
    auto cmd = RiskMinimization::setCall(&update);
    cmd->set_cmd(RiskMinCall::UPDATE_MODEL);
    // set the feature key range will be updated in this block
    blocks[b].second.to(cmd->mutable_key());
    time = pool->submit(update);
  }
```

Experiments

Sparse Logistic Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle x_i, w \rangle)) + \lambda \|w\|_1$$

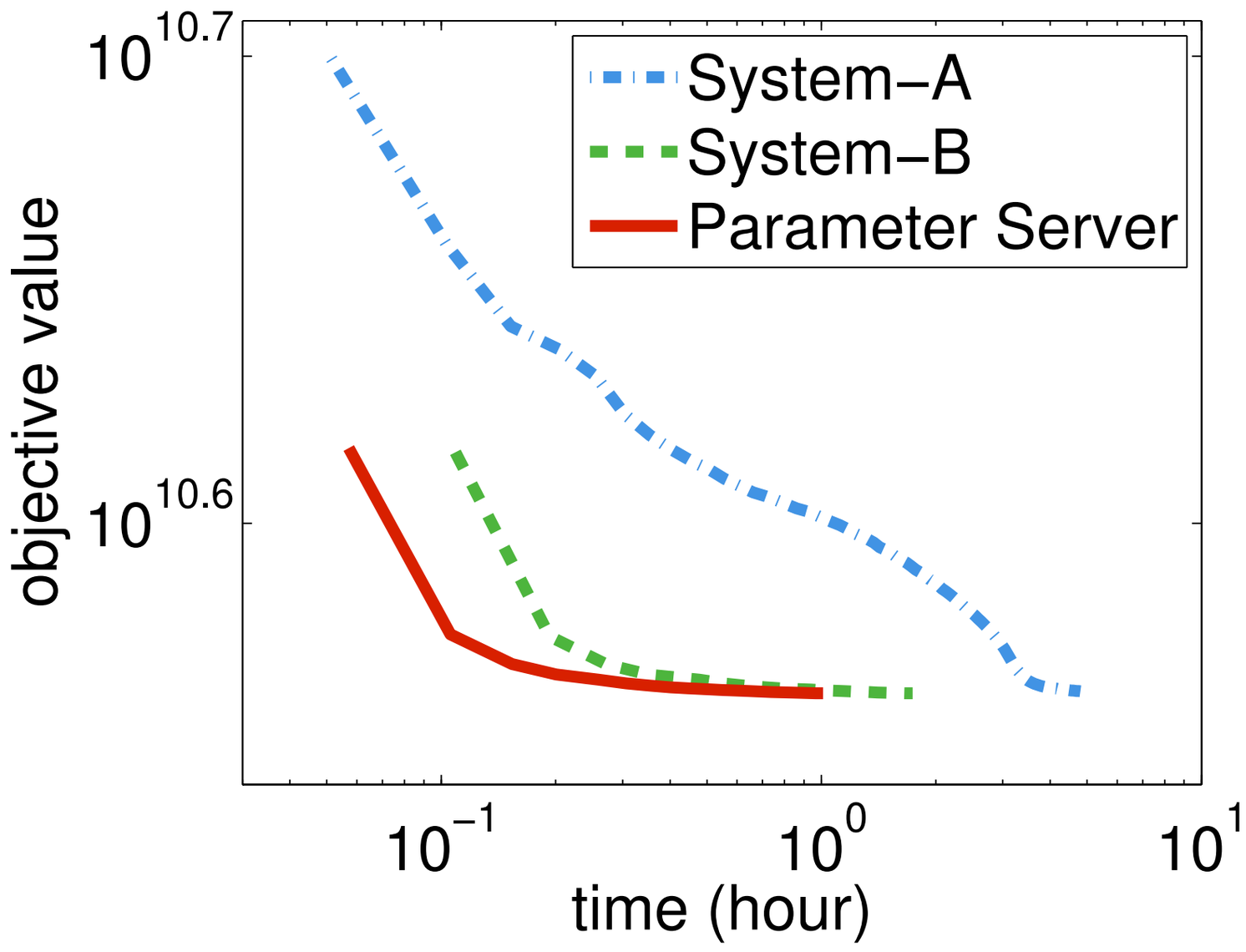
Training

examples	170 B
features	65 B
raw text data	636 T
machines	1,000
cores	16,000

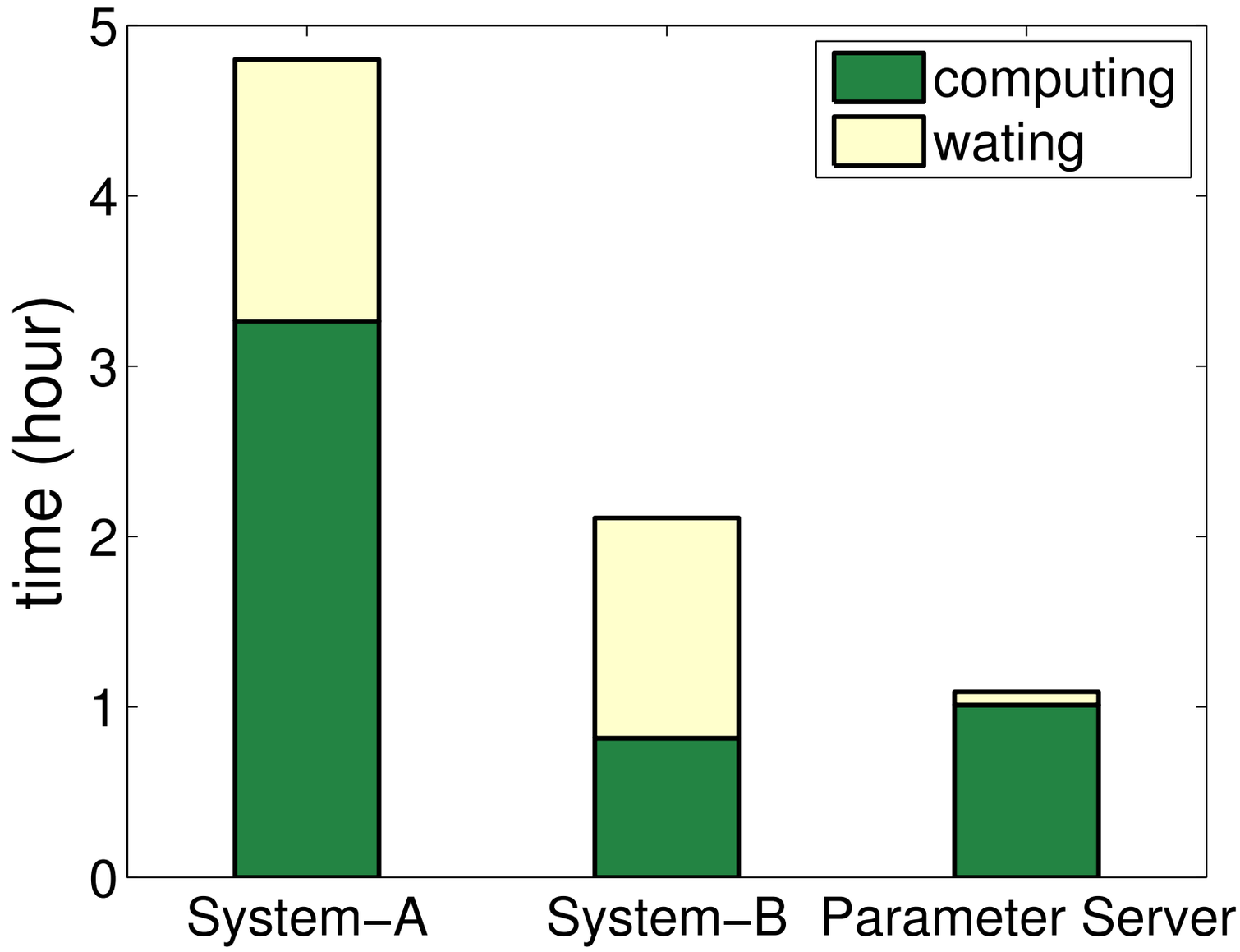
Comparison

Algorithm	Model		LOC
System-A	L-BFGS	Sequential	10K
System-B	Block CD	Sequential	30K
Parameter Server	Block CD	Bounded Delay + KKT	300

Objective vs. Time



Time Decomposition



Demo

Stochastic Gradient Descent

Idea

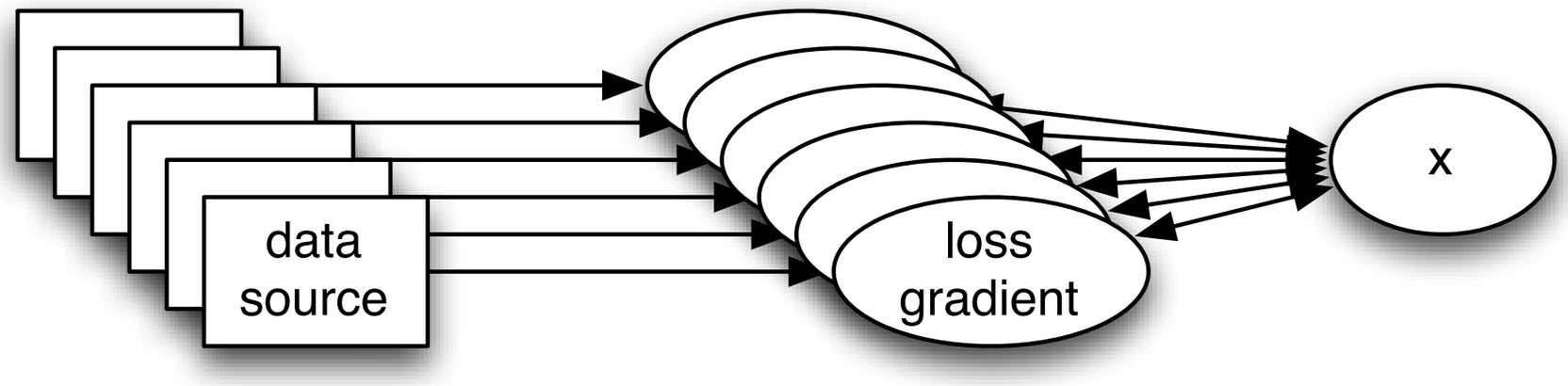
- Process one example at a time

for t = 1, 2, …
    pick up example i
    update $w^t = w^{t-1} - \eta_t f(w^{t-1}, x_i, y_i)$

- The cost of an iteration is small
- Often converges faster than batch algorithm such as gradient descent
- A sequential algorithm

Parallel gradients

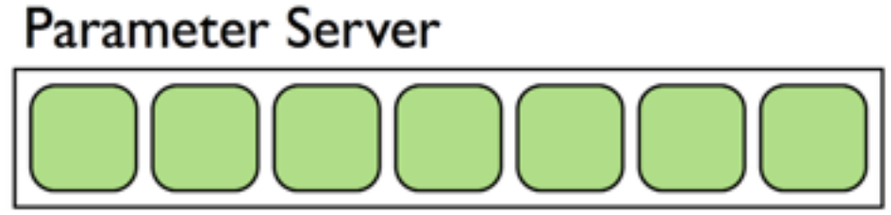
- Machines process examples in parallel



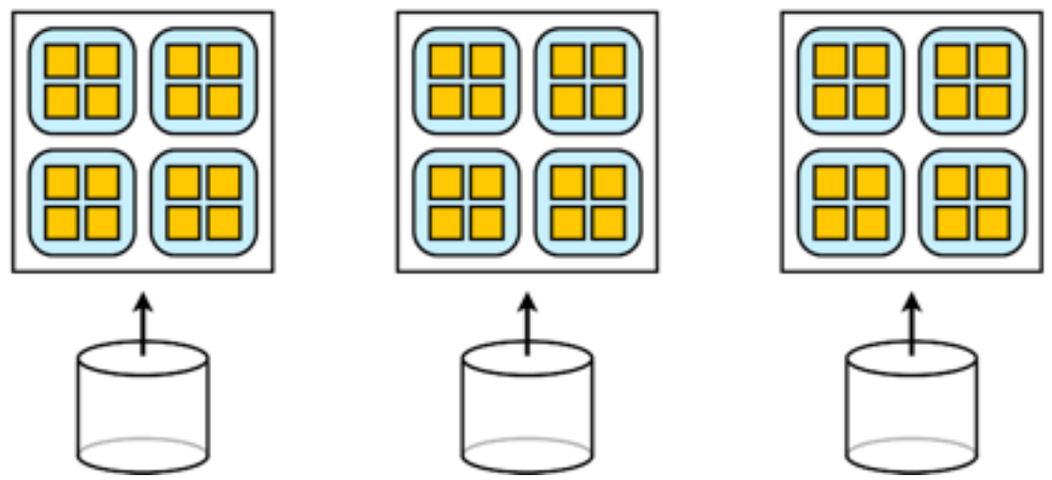
- n machines, then the delay is n-1
- the assumption of examples are not correlated often does not hold

Parallel models

- Google brain
- Several worker groups update the parameter at the same time



- Good system performance
- May affect the convergence rate
  - 10x machine may only have 2x speedup



Minibatch SGD

- Process several examples (rather than 1) in an iteration
- Convergence rate $O(1/\sqrt{bT} + 1/T)$
- Multi-thread / distributed implementation within a mini-batch
- Increasing the batch size may harms the convergence

Increase effective workload

- The overhead of a mini-batch is large:
  - read from disk, communicate over network
- Why not take advantages of this batch more before throw it away?

$$w_t = \operatorname*{argmin}_{w \in \Omega} \left[\phi_{I_t}(w) + \frac{\gamma_t}{2} \|w - w_{t-1}\|_2^2 \right]$$

- Convergence rate $\mathcal{O}(1/\sqrt{bT})$ and for strongly convexity $\mathcal{O}(\log T/(\lambda bT) + \lambda/(\sqrt{b}T))$

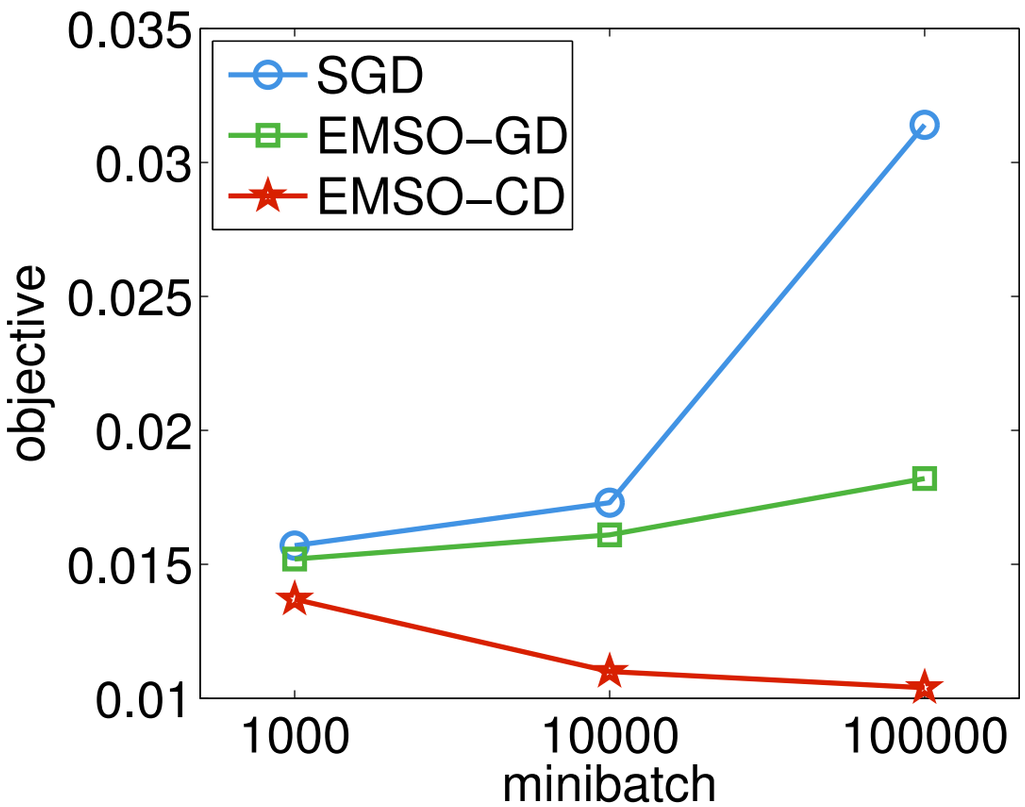
Practical implementation

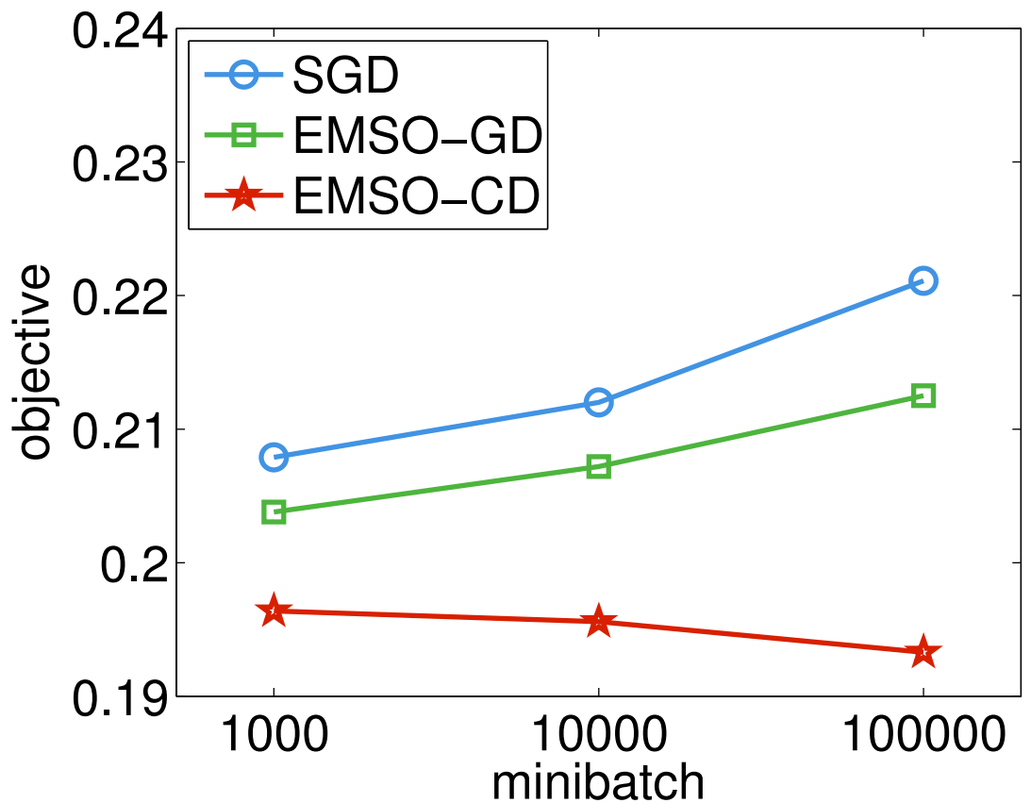
$$w_t = \underset{w \in \Omega}{\operatorname{argmin}} \left[\phi_{I_t}(w) + \frac{\gamma_t}{2} \|w - w_{t-1}\|_2^2 \right]$$

- Solve the subprolem approximately by early stop
  - ⋆ by gradient descent
  - ⋆ by coordinate descent

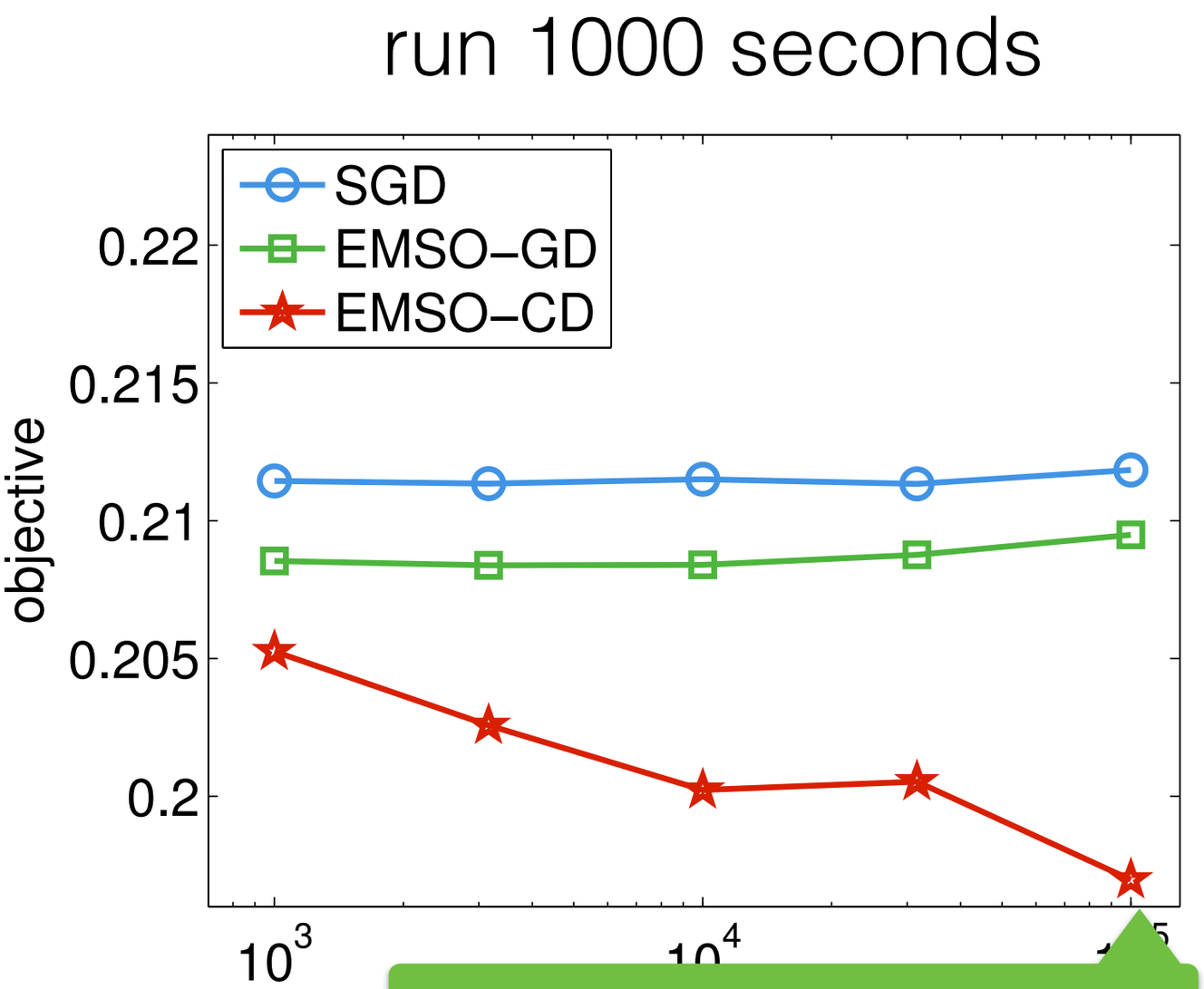
Vary minibatch size

- Logistic Regression





12 machines



percent of effective workload increased

Conclusion

- Stochastic gradient descent is used more widely than coordinate descent
- Distributing SGD is more difficulty than CD
  - samples are often more correlated comparing to features