Distributed Matrix Completion

Christina Teflioudi, Faraz Makari,
Rainer Gemulla

max planck institut
informatik

Matrix Completion

	Avatar	The Matrix	Up
Alice		4	2
Bob	3	2	
Charlie	5		3

Matrix Completion

	Avatar	The Matrix	Up
Alice	?	4	2
Bob	3	2	?
Charlie	5	?	3

Matrix Completion

- Discover (rank=1)
  - User factors
  - Movie factors

H

	W	Avatar	The Matrix	Up
		2.24	1.92	1.18
Alice	1.98	?	4	2
Bob	1.21	3	2	?
Charlie	2.30	5	?	3

V

Matrix Completion

- Discover (rank=1)
  - User factors
  - Movie factors

H

		Avatar	The Matrix	Up
	H	2.24	1.92	1.18
Alice	1.98	?	4 3.8	2 2.3
Bob	1.21	3 2.7	2 2.3	?
Charlie	2.30	5 5.2	?	3 2.7

W V

- Minimize loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

Matrix Completion

- Discover (rank=1)
  - User factors
  - Movie factors
- Minimize loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

H

	Avatar	The Matrix	Up
	2.24	1.92	1.18

W				
Alice	1.98	? 4.4	4 3.8	2 2.3
Bob	1.21	3 2.7	2 2.3	? 1.4
Charlie	2.30	5 5.2	? 4.4	3 2.7

V

Matrix Completion

- Discover (rank=1)
  - User factors
  - Movie factors

H

		Avatar	The Matrix	Up
		2.24	1.92	1.18
W	Alice 1.98	? 4.4	4 3.8	2 2.3
	Bob 1.21	3 2.7	2 2.3	? 1.4
	Charlie 2.30	5 5.2	? 4.4	3 2.7

V

- Minimize loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} \underbrace{(\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2}_{\text{Local loss}}$$

+ Bias
+ Regularization
+ ...

Distributed Matrix Completion

- Real applications can be large
  - Millions of users, Millions of items, Billions of rating
    e.g., Netflix: >20M users, >20k movies, ≅4B ratings (projected)

Distributed Matrix Completion

- Real applications can be large
  - Millions of users, Millions of items, Billions of rating

    e.g., Netflix: >20M users, >20k movies, ≅4B ratings (projected)

Scalable algorithms are necessary.

Distributed Matrix Completion

- Real applications can be large
  - Millions of users, Millions of items, Billions of rating

    e.g., Netflix: >20M users, >20k movies, ≅4B ratings (projected)

Scalable algorithms are necessary.

- Existing MapReduce algorithms
  e.g., DALS, DSGD-MR
- Strength
  - Faster than sequential algorithms
  - Can handle large datasets
- Drawbacks
  - Slow
  - Synchronous
  - No use of shared memory

Distributed Matrix Completion

- Real applications can be large
  - Millions of users, Millions of items, Billions of rating

    e.g., Netflix: >20M users, >20k movies, ≅4B ratings (projected)

Scalable algorithms are necessary.

- Existing MapReduce algorithms
  e.g., DALS, DSGD-MR
- Strength
  - Faster than sequential algorithms
  - Can handle large datasets
- Drawbacks
  - Slow
  - Synchronous
  - No use of shared memory

- New algorithms
  ASGD, DSGD++
- Strength
  - In-memory processing
  - Exploit multi-core
  - Asynchronous

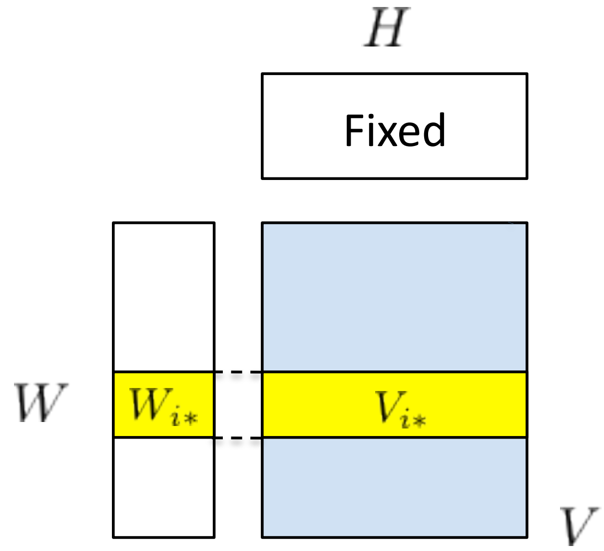
Outline

- Motivation
- Algorithms
  - **Distributed Alternating Least Squares**
  - Distributed SGD-based algorithms
    - Asynchronous SGD
    - DSGD-MR
    - DSGD++
- Experimental Results
- Summary

Alternating Least Squares (ALS)

Alternate

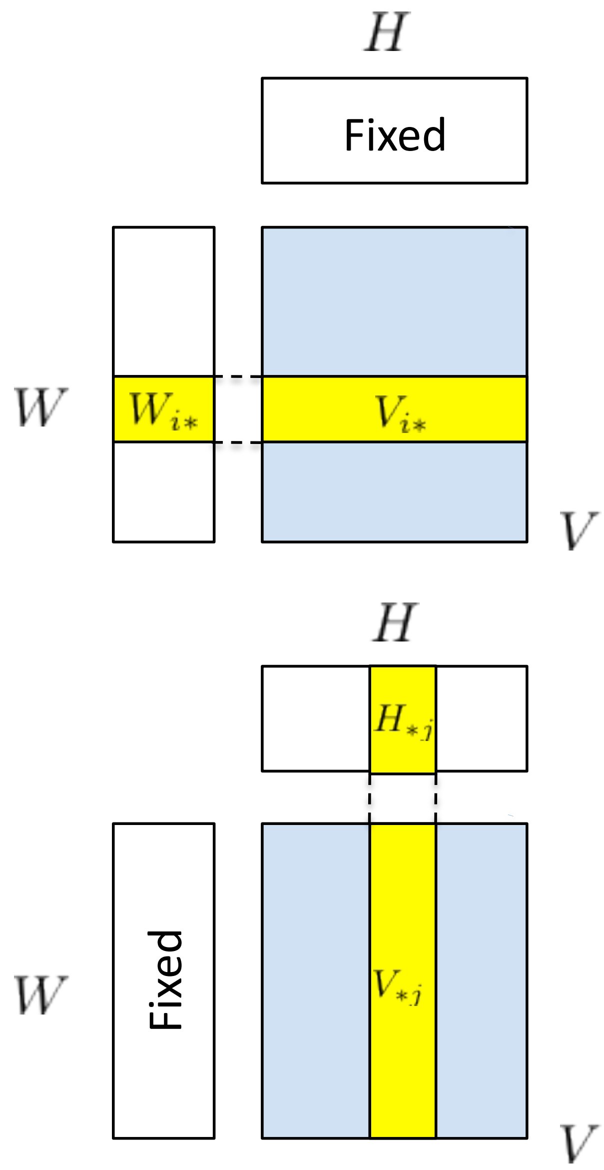
- Fix H – optimize for W



Alternating Least Squares (ALS)

Alternate

- Fix H – optimize for W
- Fix W – optimize for H



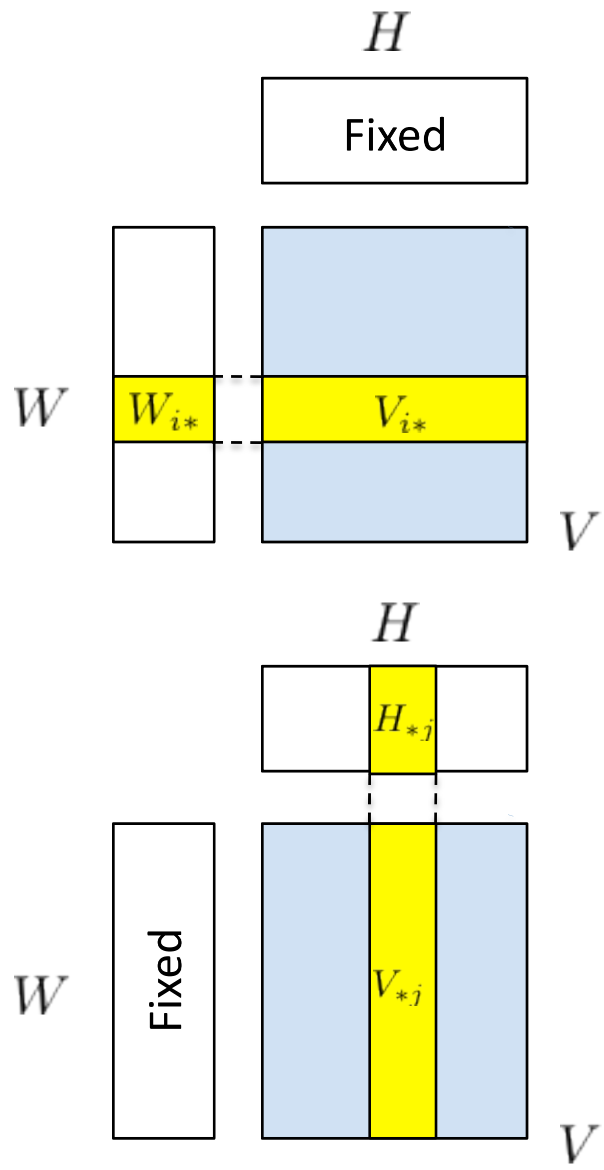
Alternating Least Squares (ALS)

Alternate

- Fix H – optimize for W

- Fix W – optimize for H

For each user/movie: solve a least squares problem

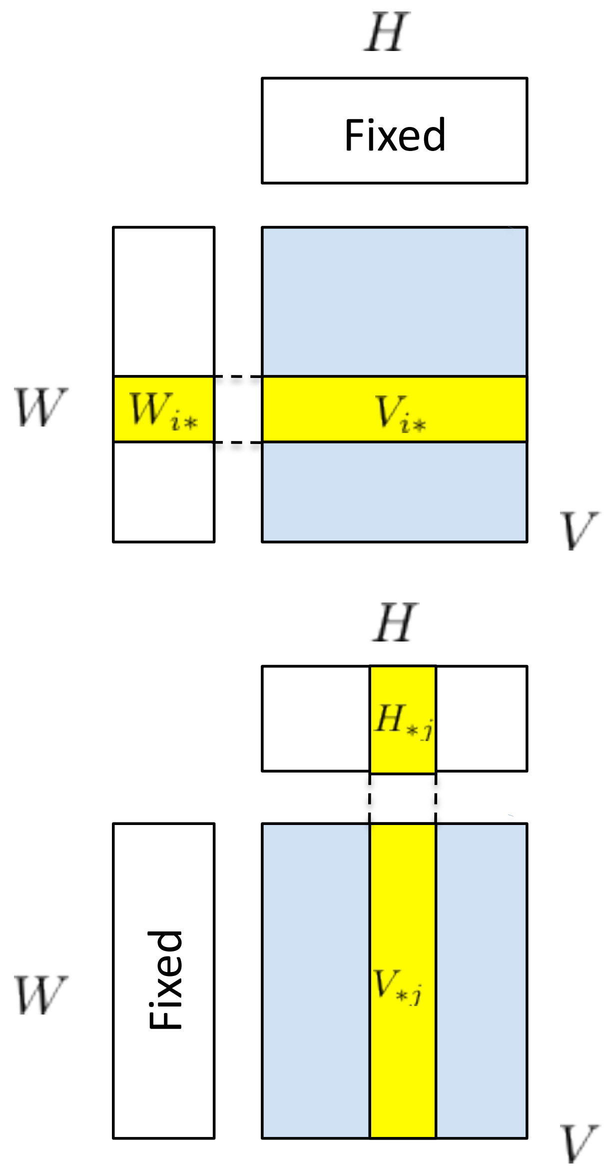


Alternating Least Squares (ALS)

Alternate

- Fix H – optimize for W
- Fix W – optimize for H

For each user/movie: solve a least squares problem



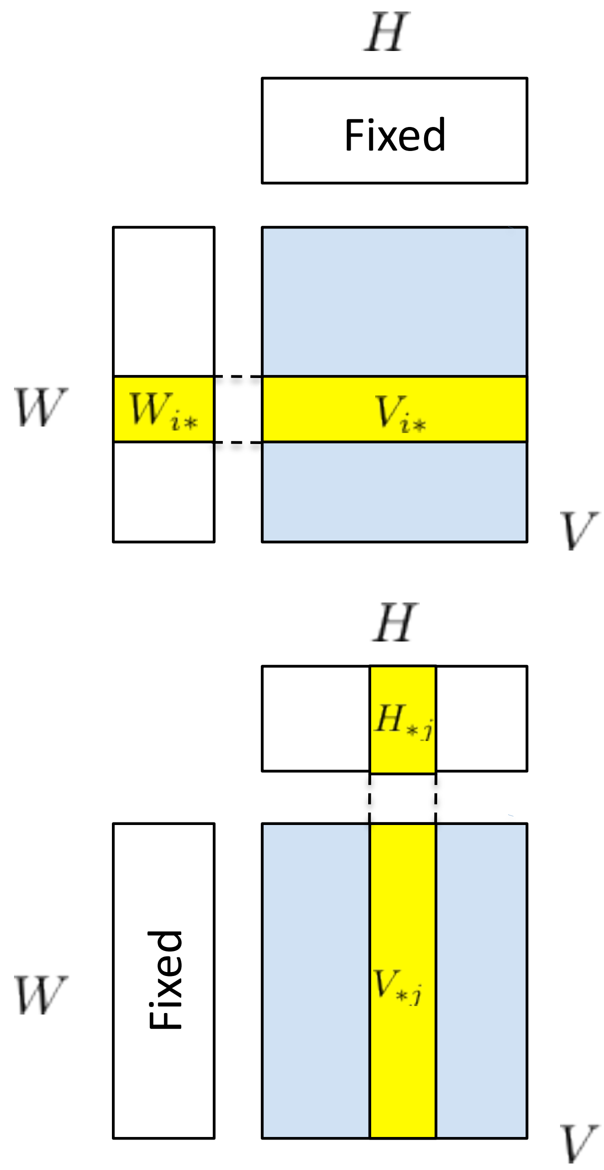
Distributed ALS similar to [Zhou08]
Difference: on each node multiple threads
instead of multiple processes

Alternating Least Squares (ALS)

Alternate

- Fix H – optimize for W
- Fix W – optimize for H

For each user/movie: solve a least squares problem



Distributed ALS similar to [Zhou08]
Difference: on each node multiple threads instead of multiple processes

- Slow (cubic in rank)
- Memory intensive (stores data matrix twice)

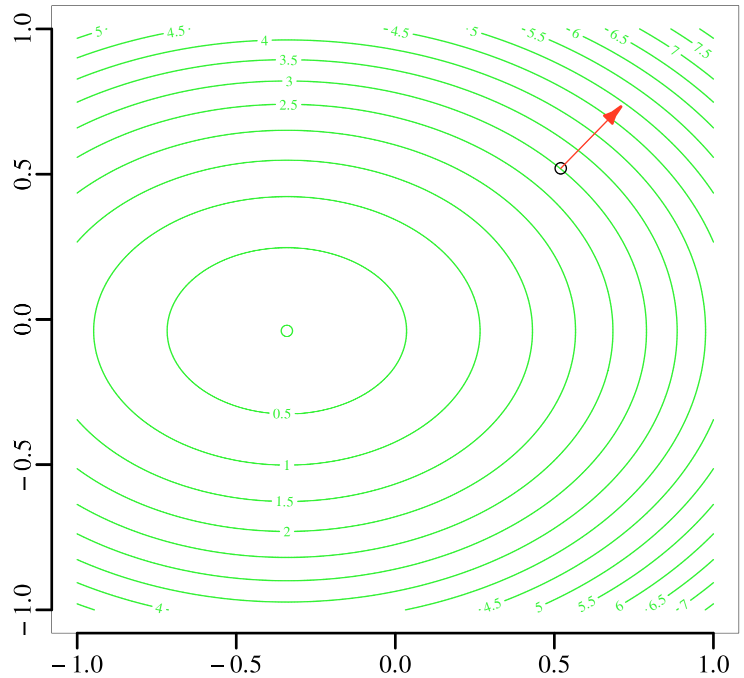
Outline

- Motivation
- Algorithms
  - Distributed Alternating Least Squares
  - **Distributed SGD-based algorithms**
    - Asynchronous SGD
    - DSGD-MR
    - DSGD++
- Experimental Results
- Summary

Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

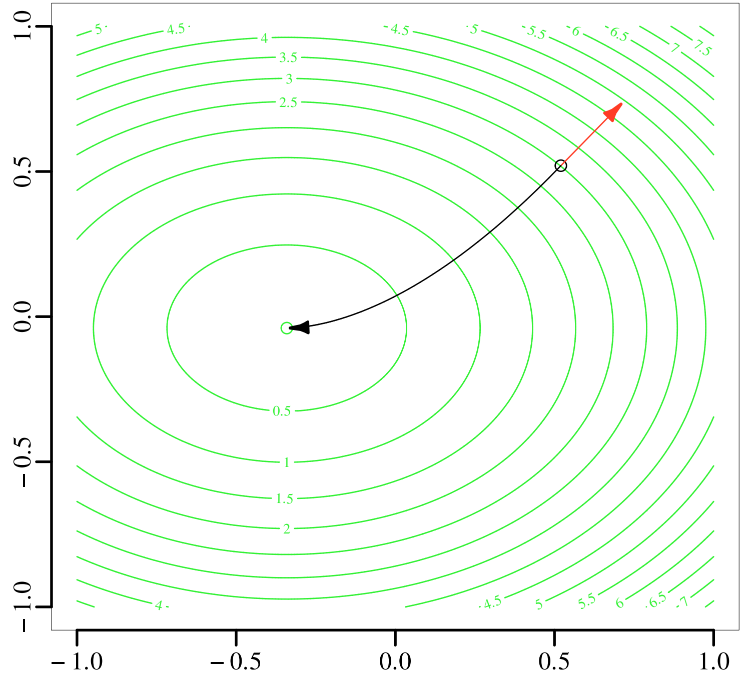
- Pick a starting point θ_0



Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

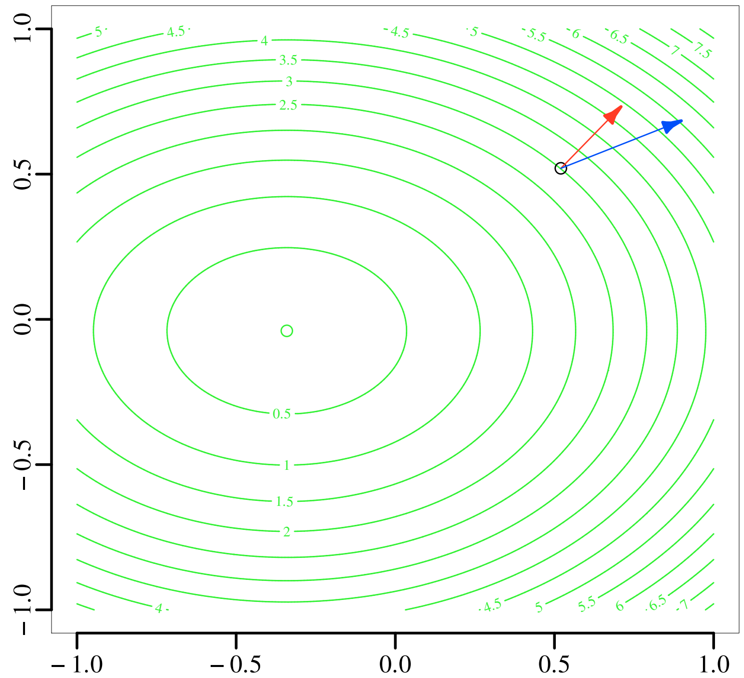
- Pick a starting point θ_0



Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

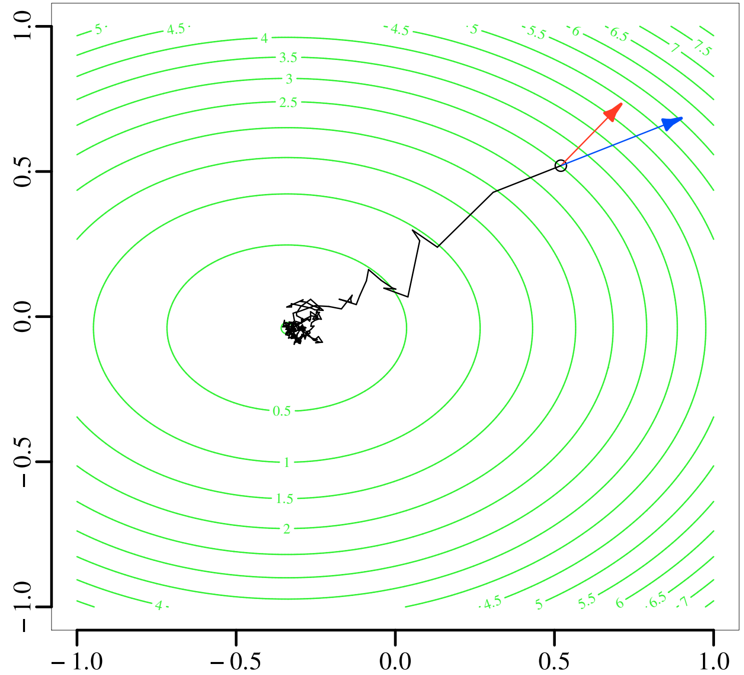
- Pick a starting point θ_0
- Approximate gradient $\widehat{L}'(\theta_n)$



Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

- Pick a starting point θ_0
- Approximate gradient $\widehat{L}'(\theta_n)$
- Jump "approximately" downhill

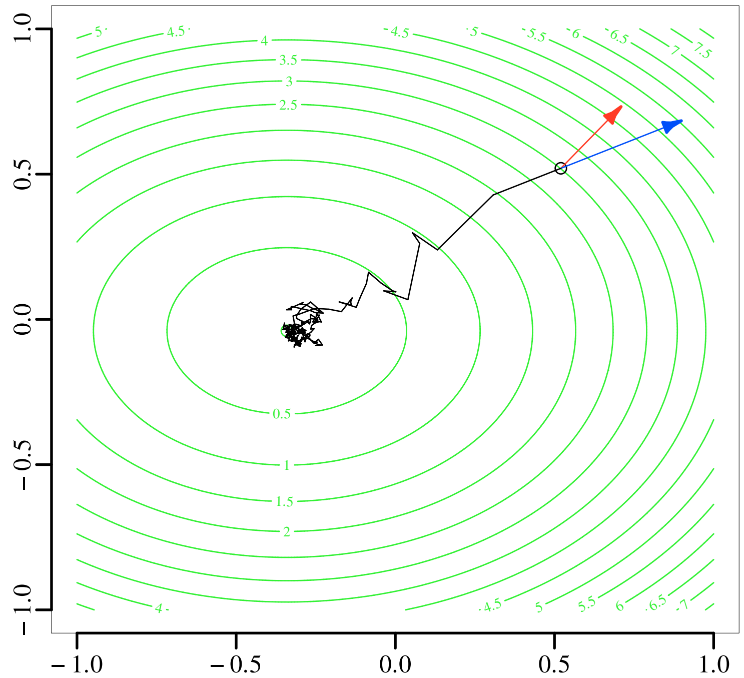


Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

- Pick a starting point θ_0
- Approximate gradient $\widehat{L}'(\theta_n)$
- Jump “approximately” downhill
- Stochastic difference equation

$$\theta_{n+1} = \theta_n - \varepsilon_n \widehat{L}'(\theta_{n+1})$$



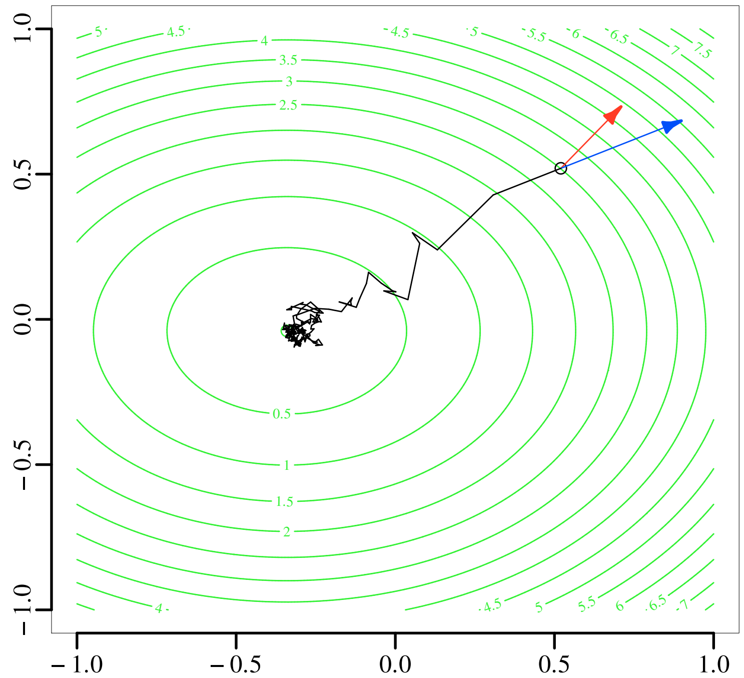
Stochastic Gradient Descent(SGD)

Goal: Find minimum θ^* of function L

- Pick a starting point θ_0
- Approximate gradient $\widehat{L}'(\theta_n)$
- Jump "approximately" downhill
- Stochastic difference equation

$$\theta_{n+1} = \theta_n - \varepsilon_n \widehat{L}'(\theta_{n+1})$$

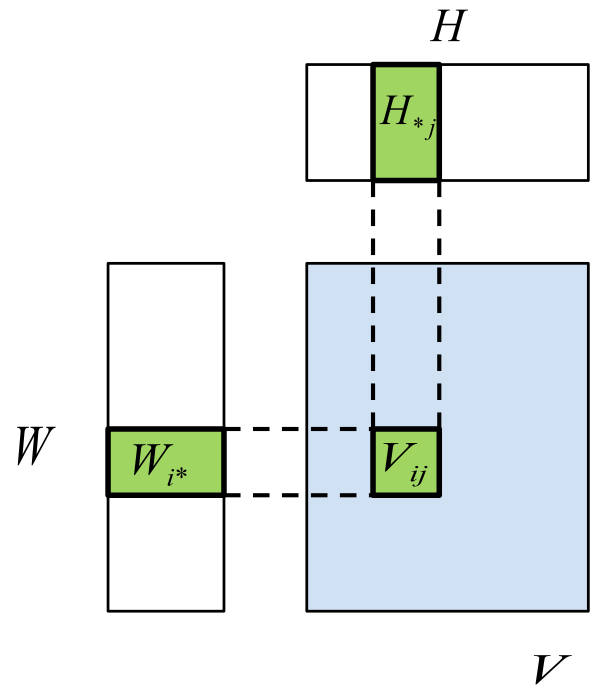
- Under certain conditions, asymptotically approximates (continuous) gradient descent



SGD for Matrix Completion

$$L = \sum_{(i,j)\in Z} \boxed{(V_{ij} - [WH]_{ij})^2}$$

Local loss

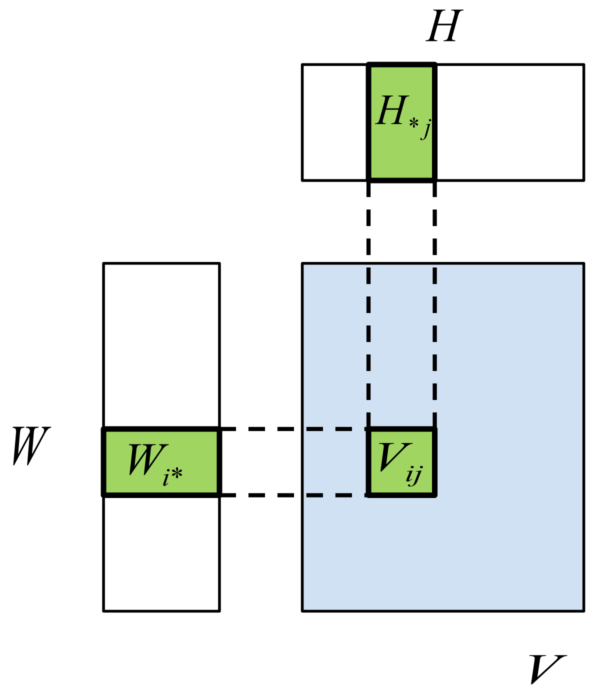


SGD for Matrix Completion

$$L = \sum_{(i,j)\in Z} \boxed{(V_{ij} - [WH]_{ij})^2}$$

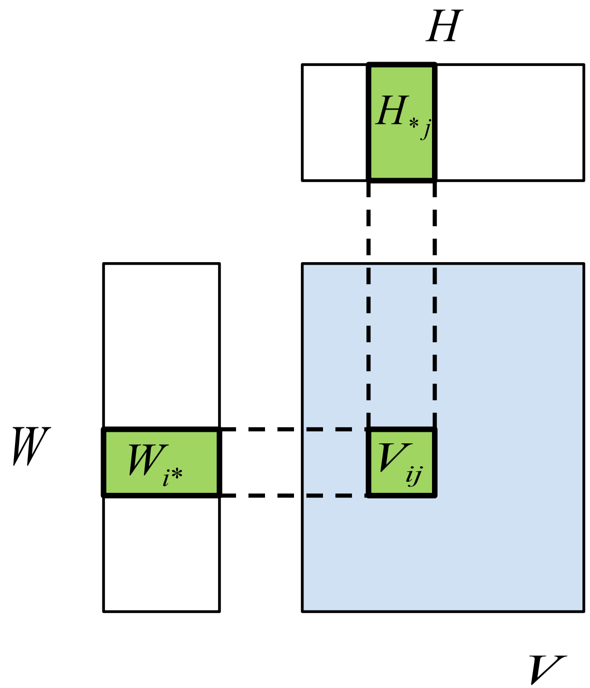
Local loss

- Estimate gradient based on **single** training point
- Scale up by # training points N



SGD for Matrix Completion

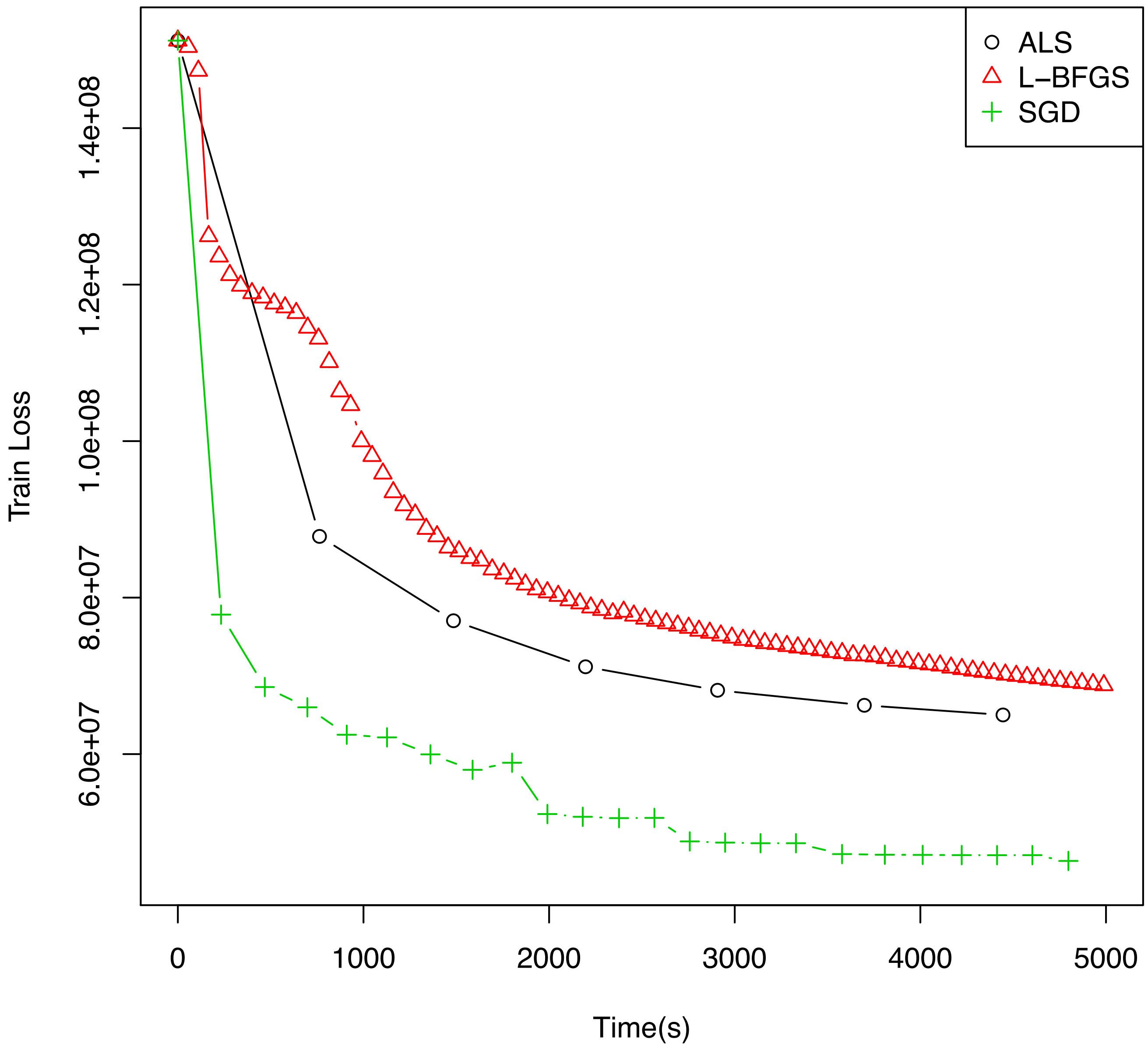
$$L = \sum_{(i,j)\in Z} \underbrace{(V_{ij} - [WH]_{ij})^2}_{\text{Local loss}}$$



- Estimate gradient based on **single** training point
- Scale up by # training points N

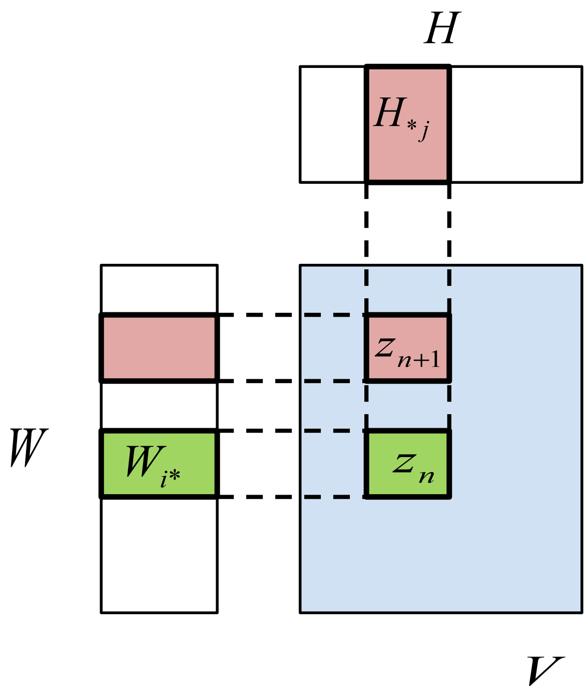
- SGD epoch:
  1. Pick a random training point
  2. Compute approximate gradient
  3. Update W_{i*} and H_{*j}
  4. Repeat N times

Netflix Single-Core



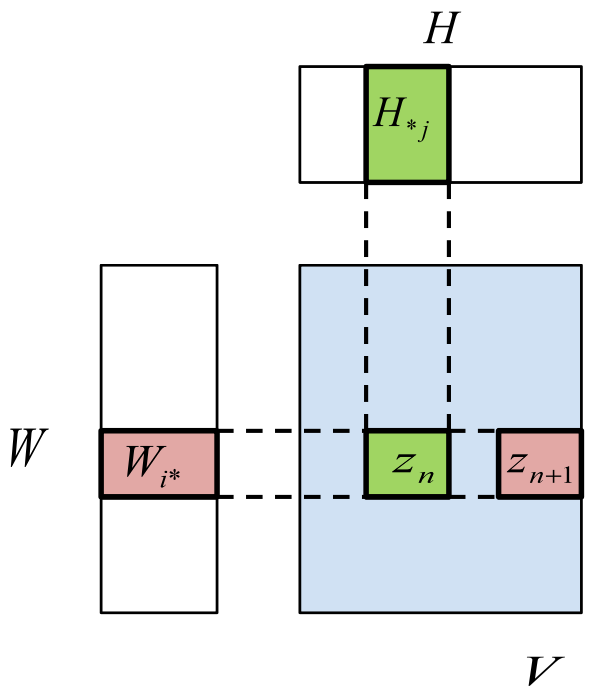
Problem Structure

SGD steps depend on each other



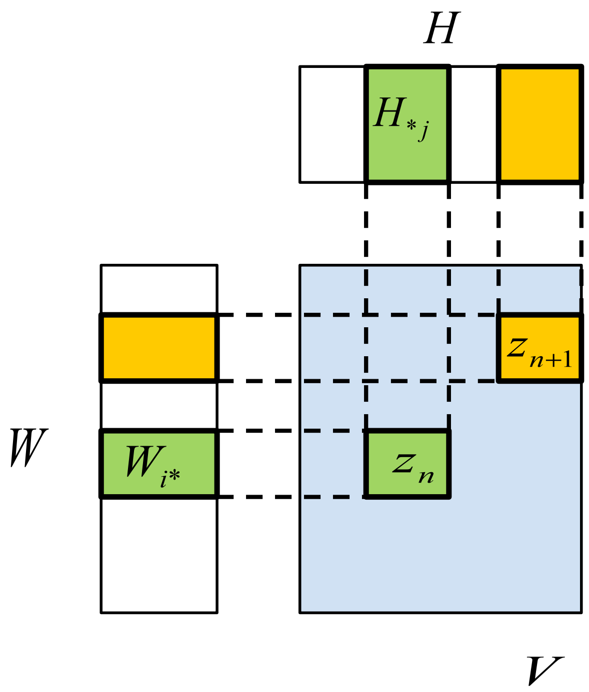
Problem Structure

SGD steps depend on each other



Problem Structure

SGD steps depend on each other

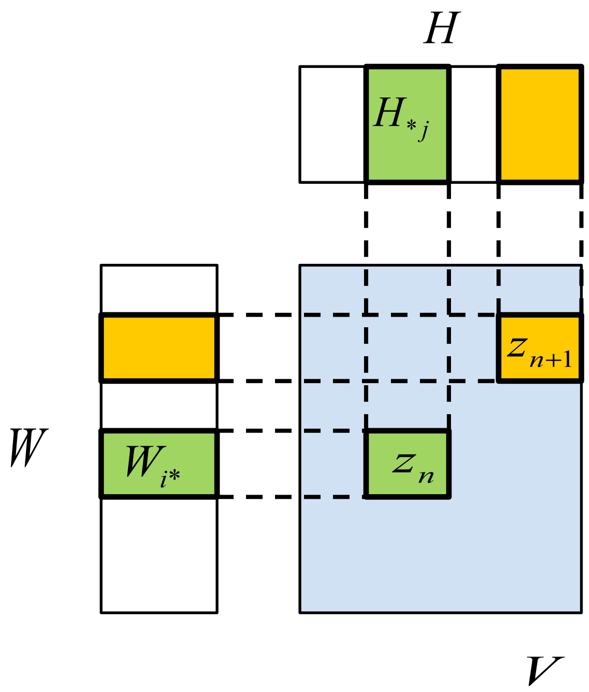


But not all steps are dependent

Shared-memory, parallel SGD:
Efficient and simple

Problem Structure

SGD steps depend on each other



But not all steps are dependent

Shared-memory, parallel SGD:
Efficient and simple

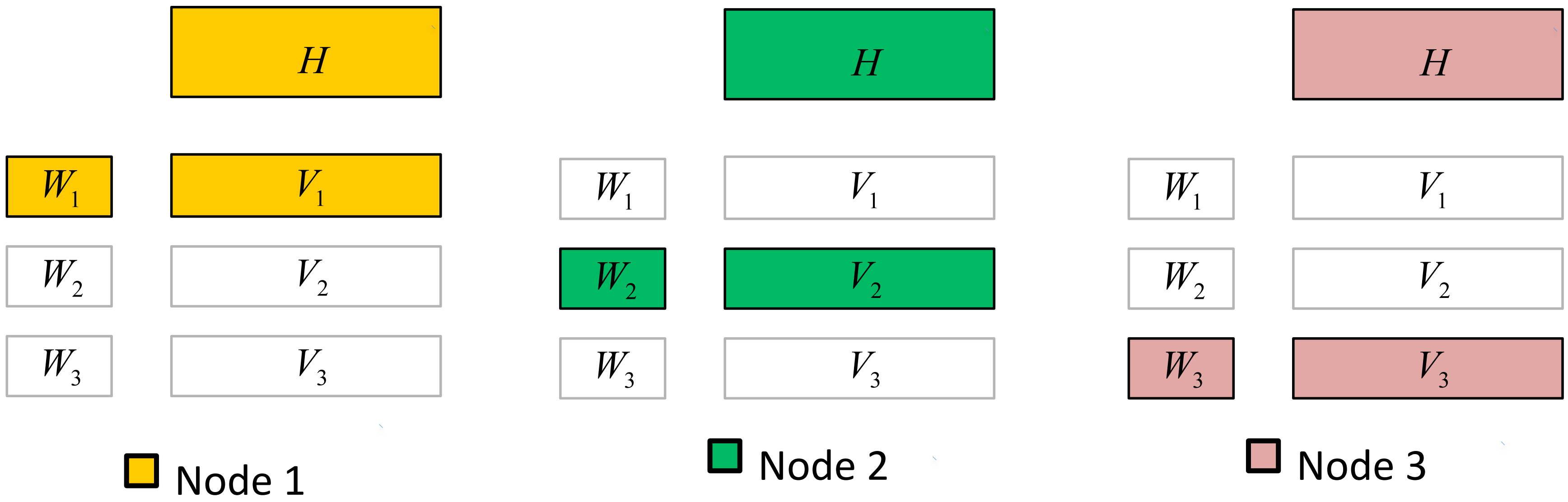
Parallel SGD slow for larger problems.

Outline

- Motivation
- Algorithms
  - Distributed Alternating Least Squares
  - **Distributed SGD-based algorithms**
    - **Asynchronous SGD**
    - DSGD-MR
    - DSGD++
- Experimental Results
- Summary

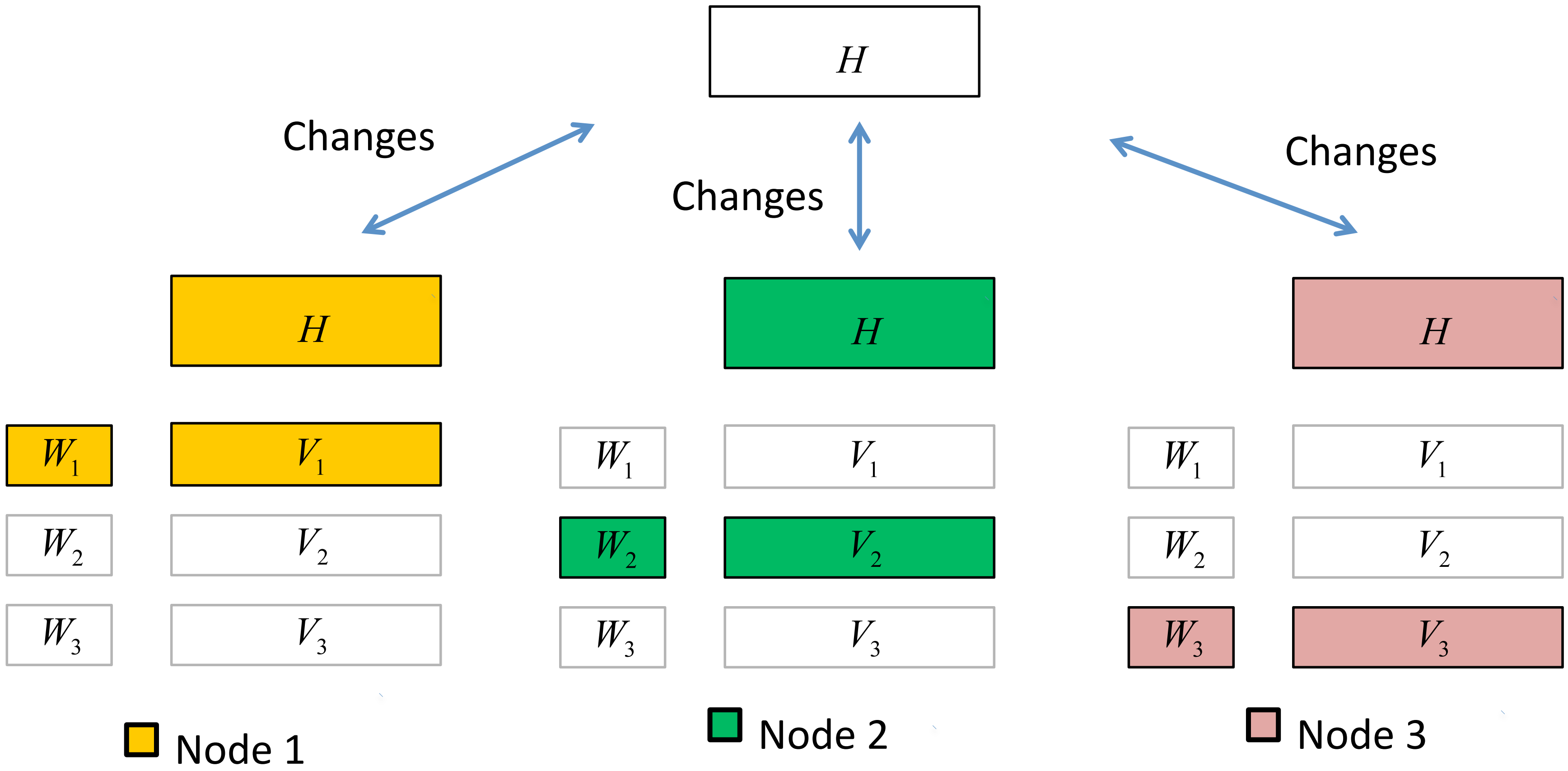
Asynchronous SGD (ASGD)

Each node works on a local copy of the movies matrix H.



Asynchronous SGD (ASGD)

Each node works on a local copy of the movies matrix H.
Local copies are synchronized continuously.

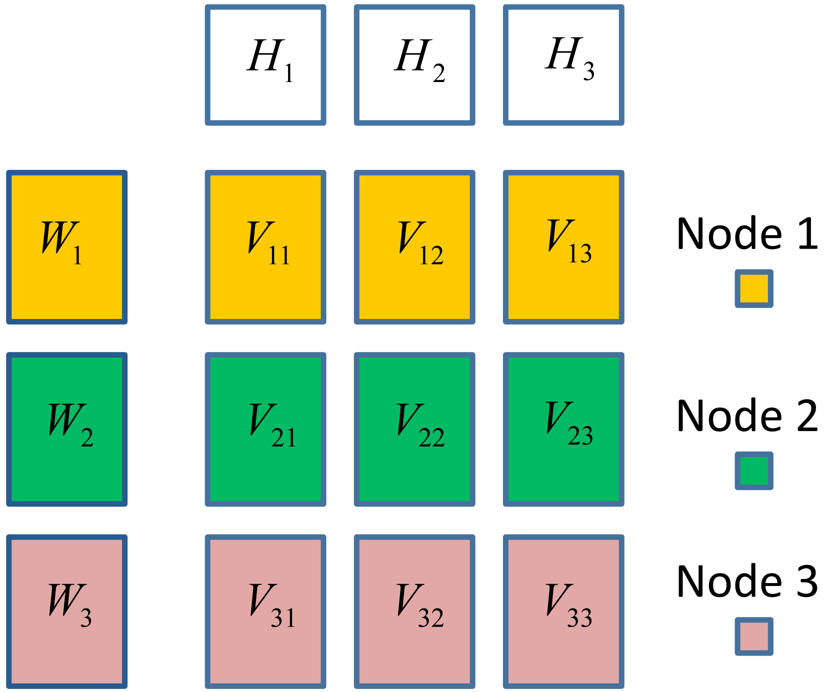


Outline

- Motivation
- Algorithms
  - Distributed Alternating Least Squares
  - **Distributed SGD-based algorithms**
    - Asynchronous SGD
    - **DSGD-MR**
    - DSGD++
- Experimental Results
- Summary

Distributed SGD-MapReduce

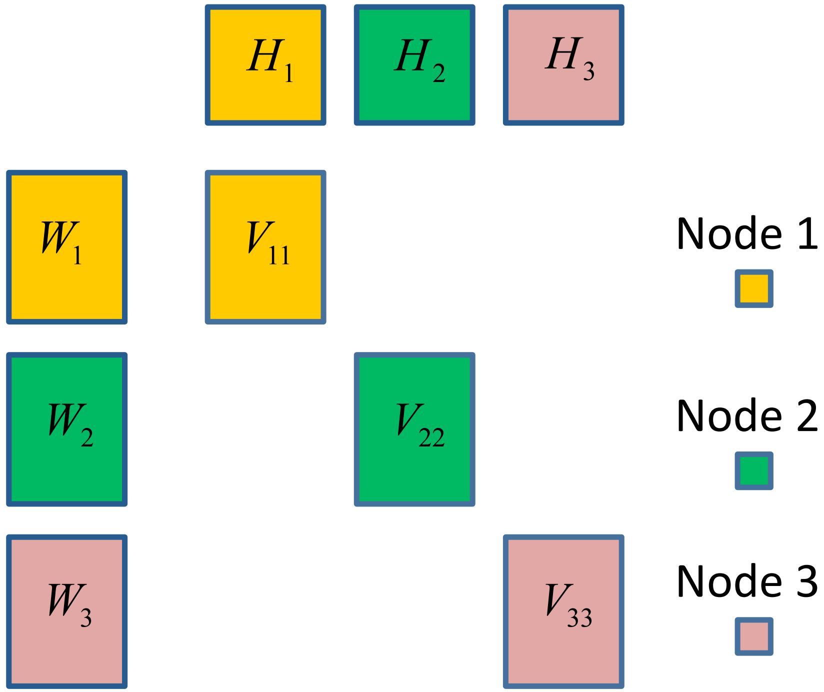
Block and distribute V



Distributed SGD-MapReduce

Block and distribute V

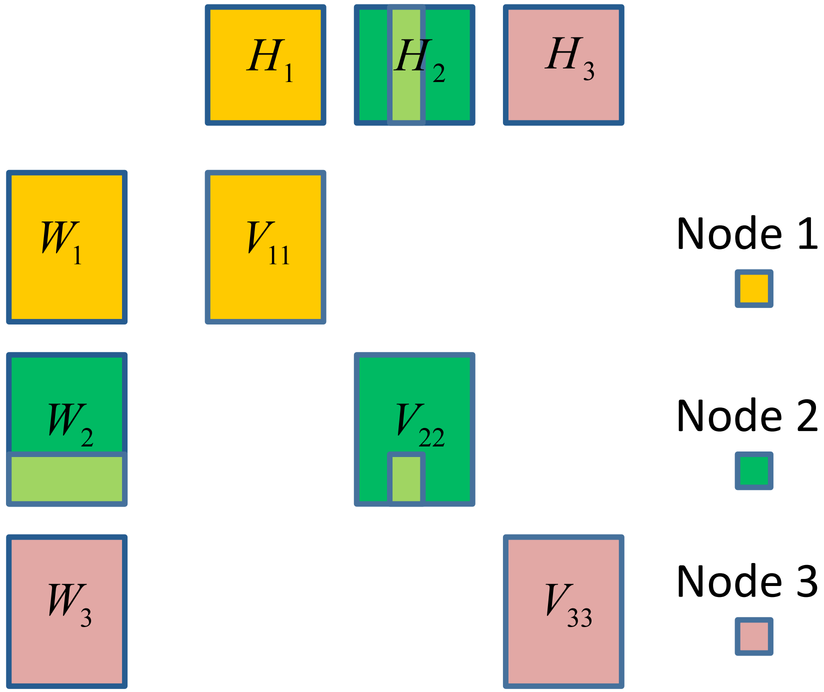
1. Pick a “diagonal”



Distributed SGD-MapReduce

Block and distribute V

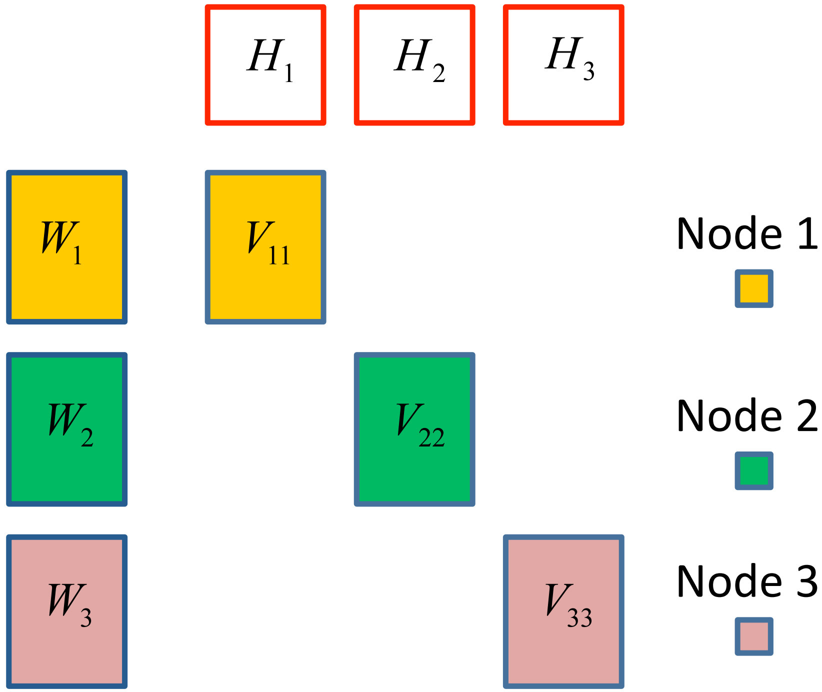
1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)



Distributed SGD-MapReduce

Block and distribute V

1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Write back the results

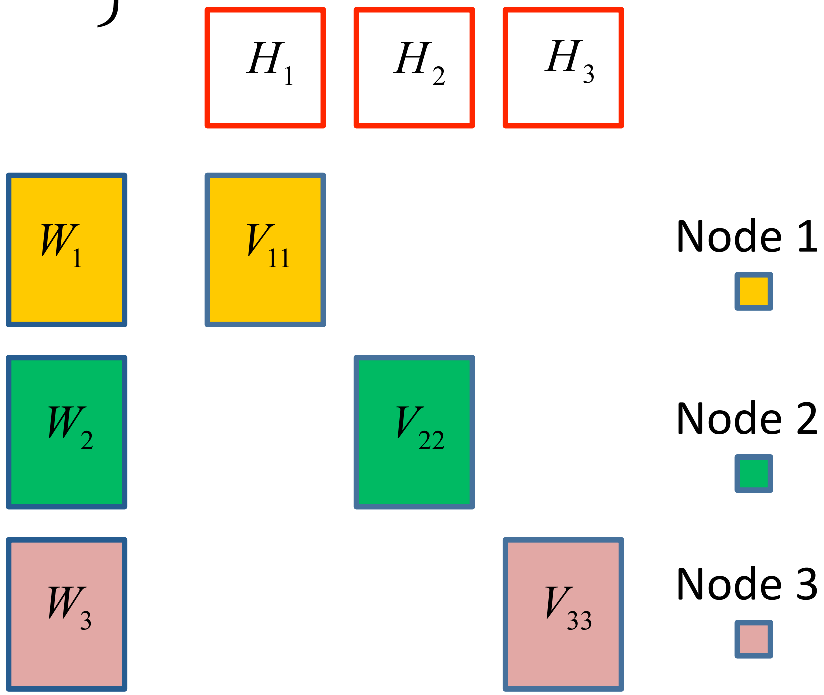


Distributed SGD-MapReduce

Block and distribute V

1. Pick a "diagonal"
2. Run SGD on the diagonal (in parallel)
3. Write back the results

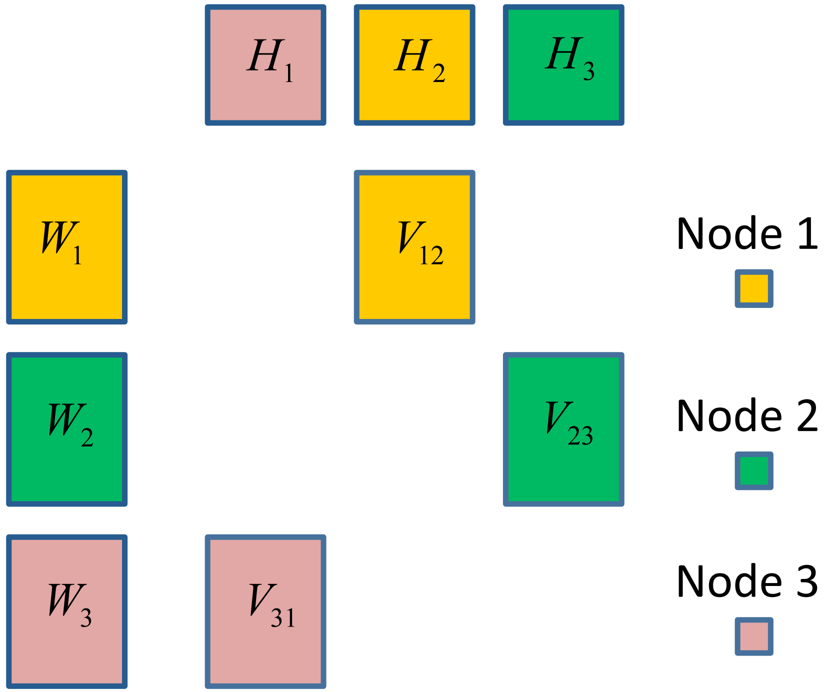
1 subepoch
1 Map Job



Distributed SGD-MapReduce

Block and distribute V

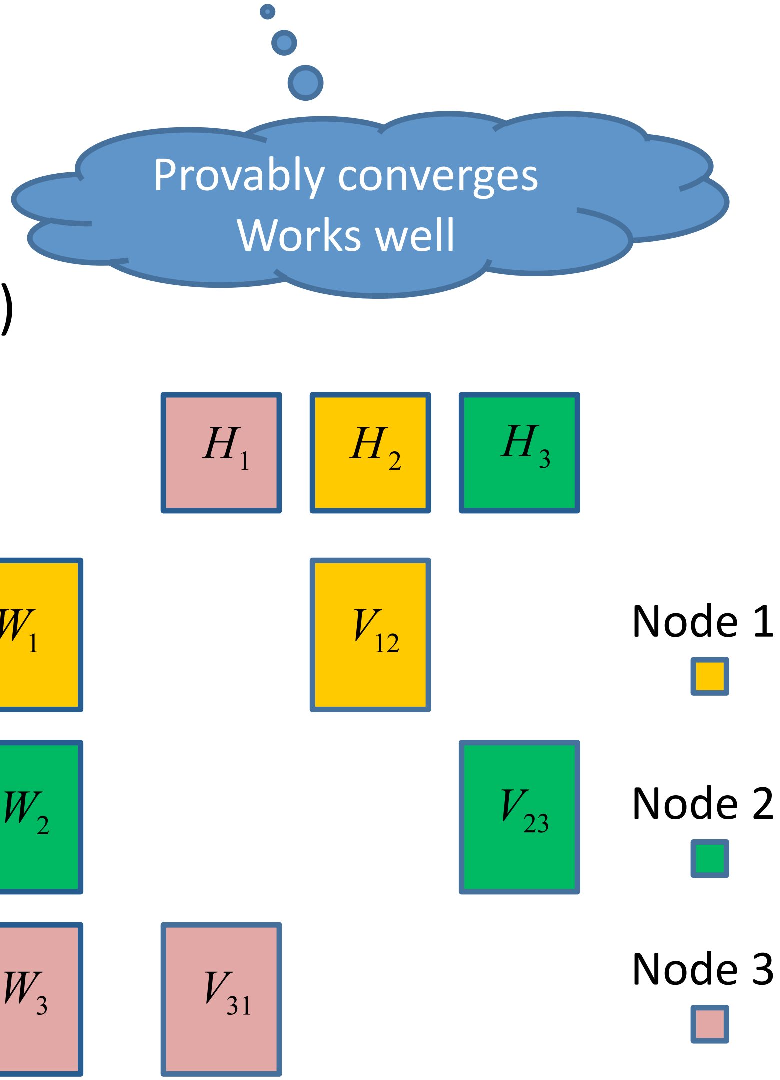
1. Pick a "diagonal"
2. Run SGD on the diagonal (in parallel)
3. Write back the results
4. Move to the next "diagonal"



Distributed SGD-MapReduce

Block and distribute V

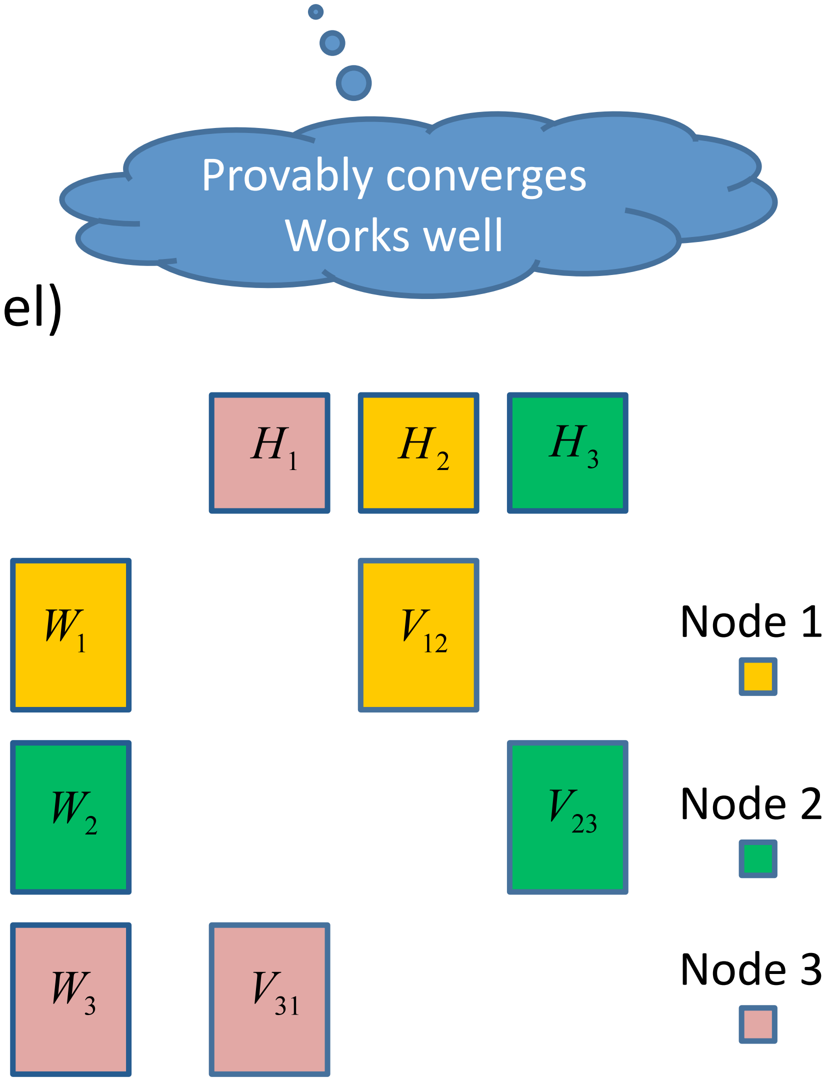
1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Write back the results
4. Move to the next “diagonal”



Distributed SGD-MapReduce

Block and distribute V

1. Pick a “diagonal”
2. Run SGD on the diagonal (in parallel)
3. Write back the results
4. Move to the next “diagonal”



DSGD-MR drawbacks:

- Repeatedly reads/writes from/to disk
- Synchronous
- No overlapping of communication and computation
- No shared memory

Outline

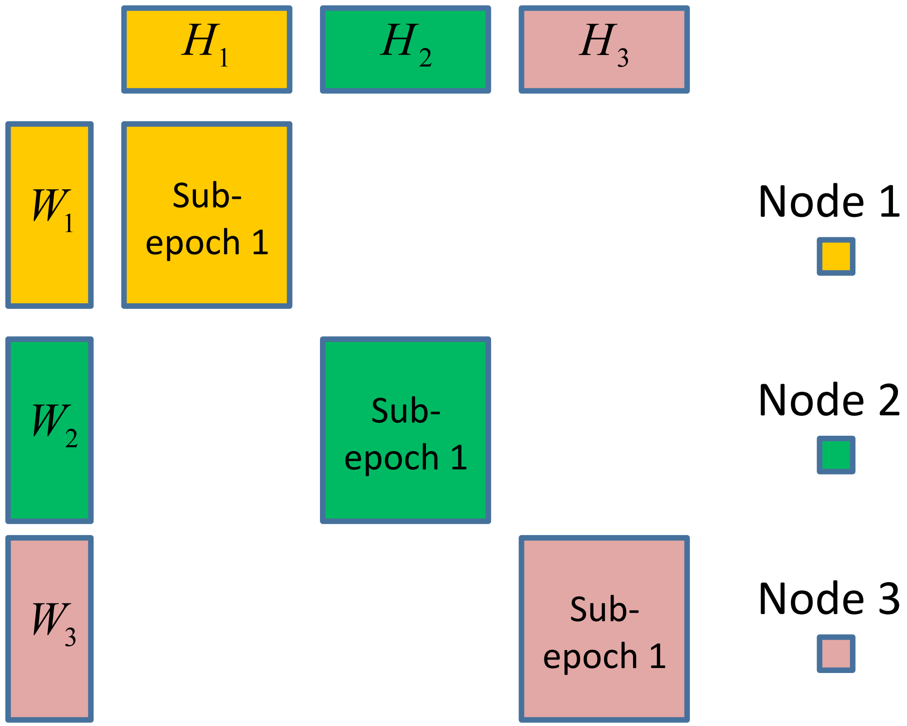
- Motivation
- Algorithms
  - Distributed Alternating Least Squares
  - **Distributed SGD-based algorithms**
    - Asynchronous SGD
    - DSGD-MR
    - **DSGD++**
- Experimental Results
- Summary

DSGD++: Direct communication between nodes

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- Repeatedly reads/writes from/to disk
- Synchronous
- No overlapping of communication and computation
- No shared memory

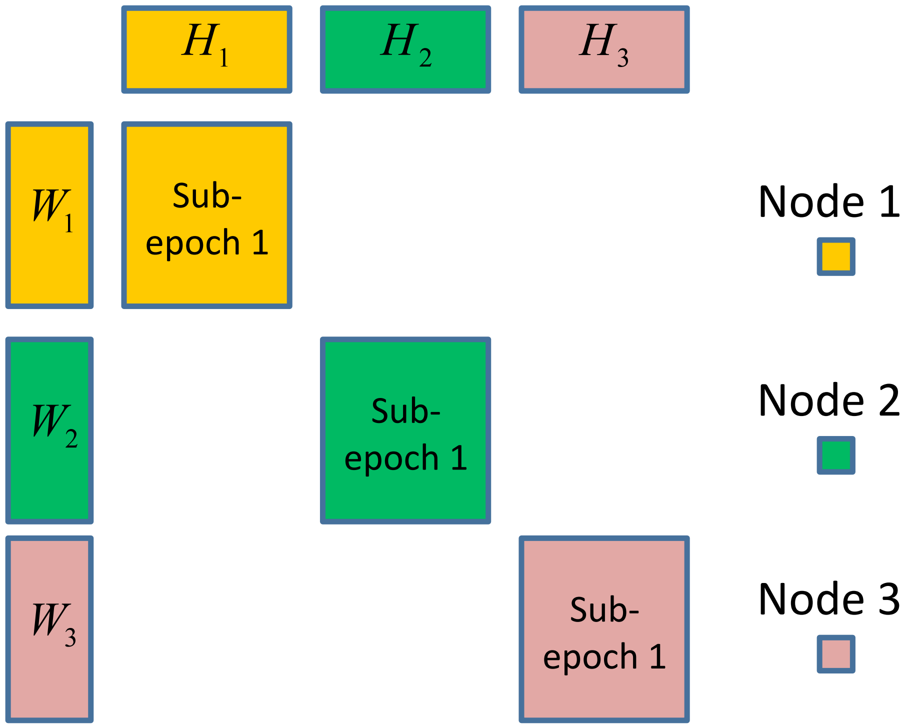


DSGD++: Direct communication between nodes

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- Synchronous
- No overlapping of communication and computation
- No shared memory

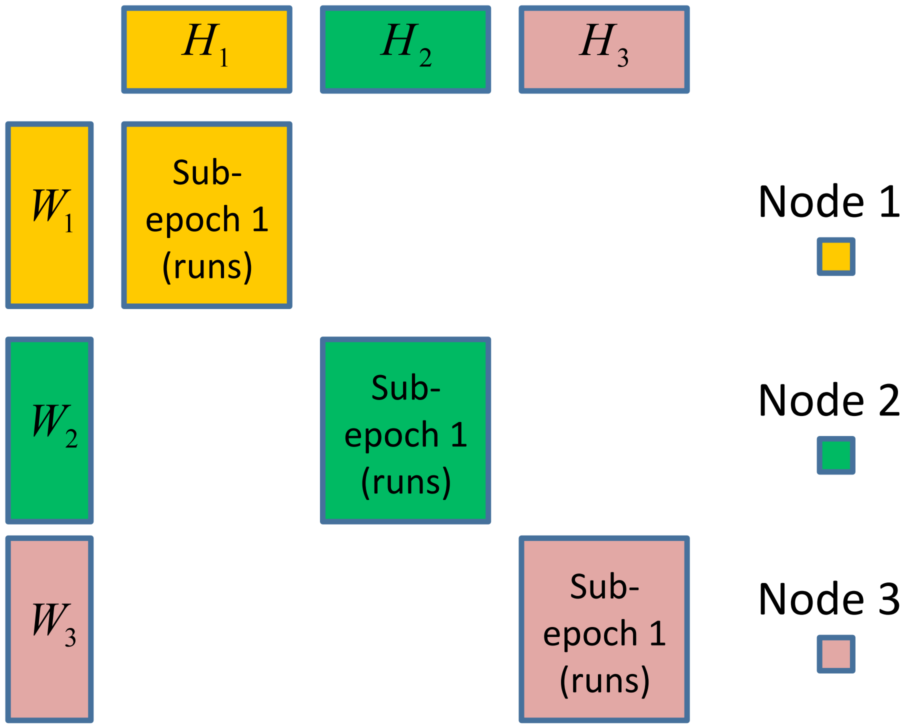


DSGD++: Overlap Subepochs

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

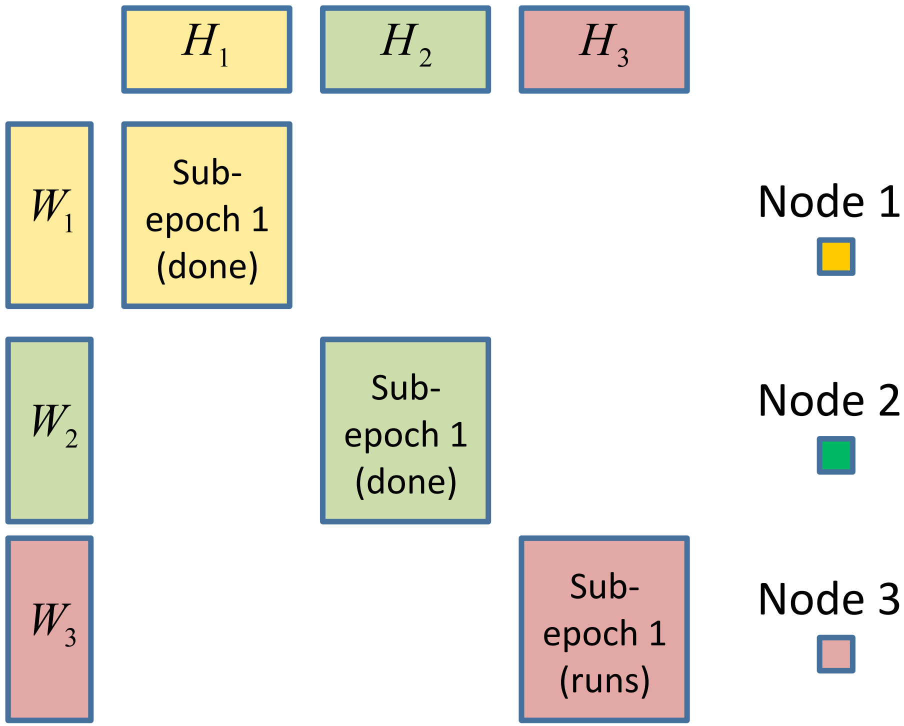
- ~~Repeatedly reads/writes from/to disk~~
- Synchronous
- No overlapping of communication and computation
- No shared memory



DSGD++: Overlap Subepochs

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:
- ~~Repeatedly reads/writes from/to disk~~
- Synchronous
- No overlapping of communication and computation
- No shared memory

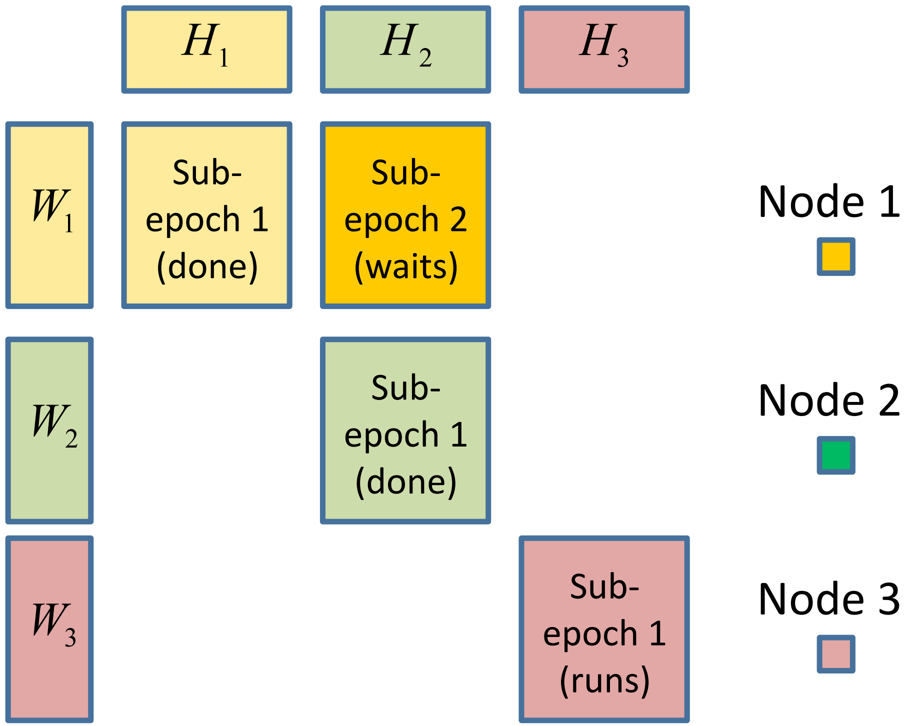


DSGD++: Overlap Subepochs

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- Synchronous
- No overlapping of communication and computation
- No shared memory

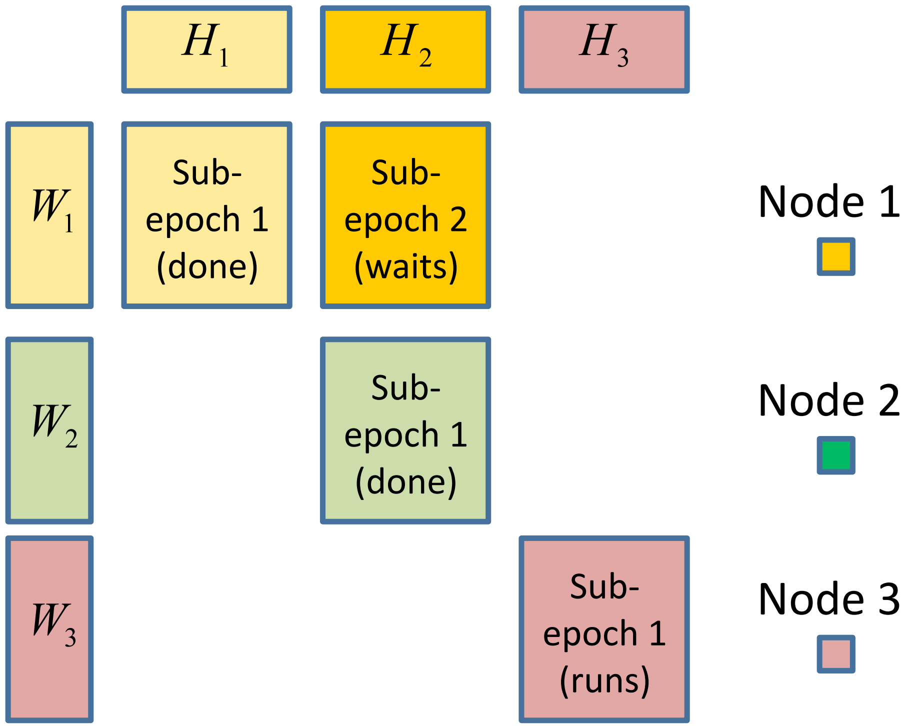


DSGD++: Overlap Subepochs

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- Synchronous
- No overlapping of communication and computation
- No shared memory

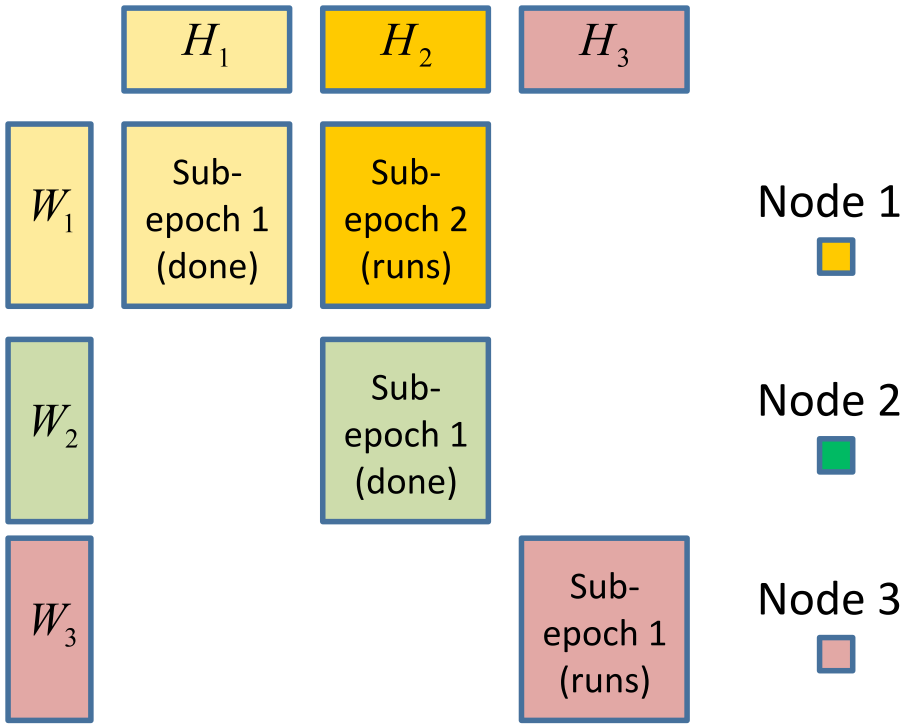


DSGD++: Overlap Subepochs

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- No overlapping of communication and computation
- No shared memory

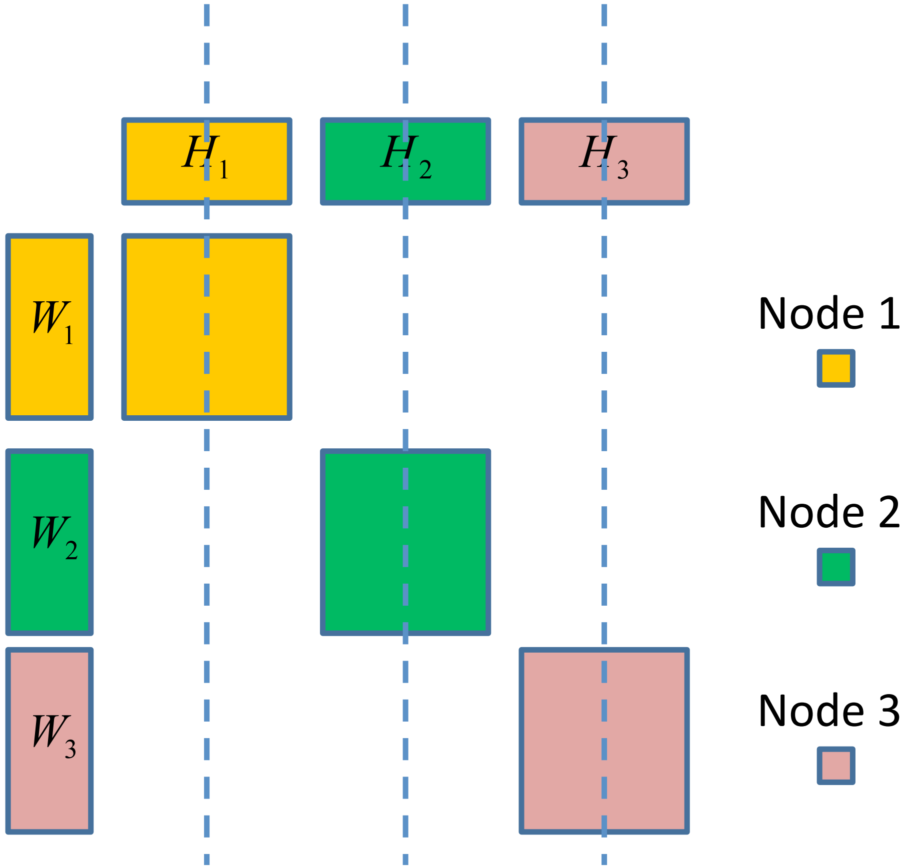


DSGD++: Overlay computation and communication

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- No overlapping of communication and computation
- No shared memory

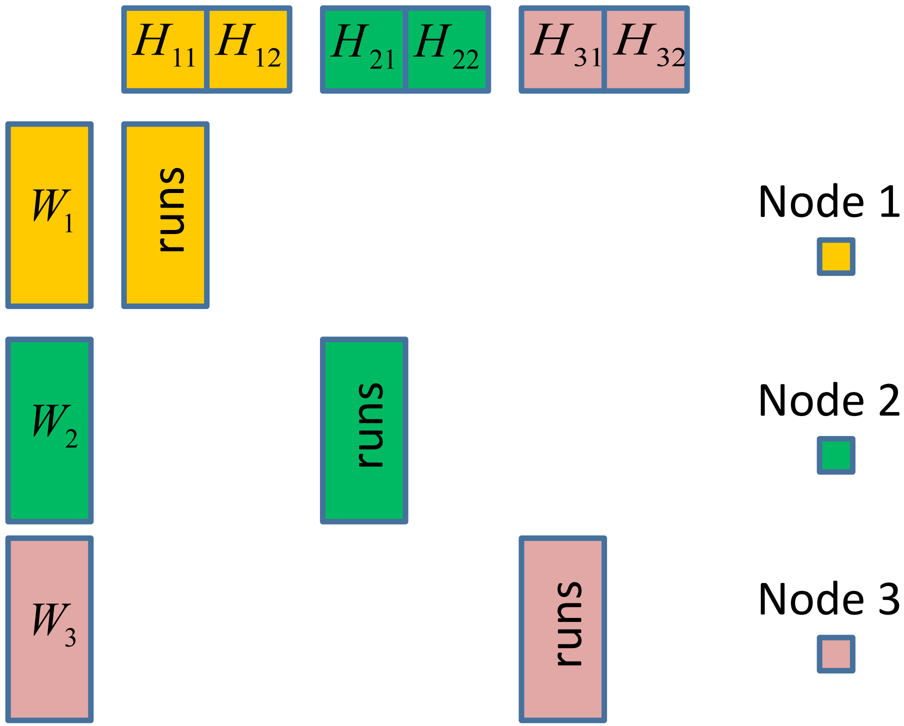


DSGD++: Overlay computation and communication

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- No overlapping of communication and computation
- No shared memory

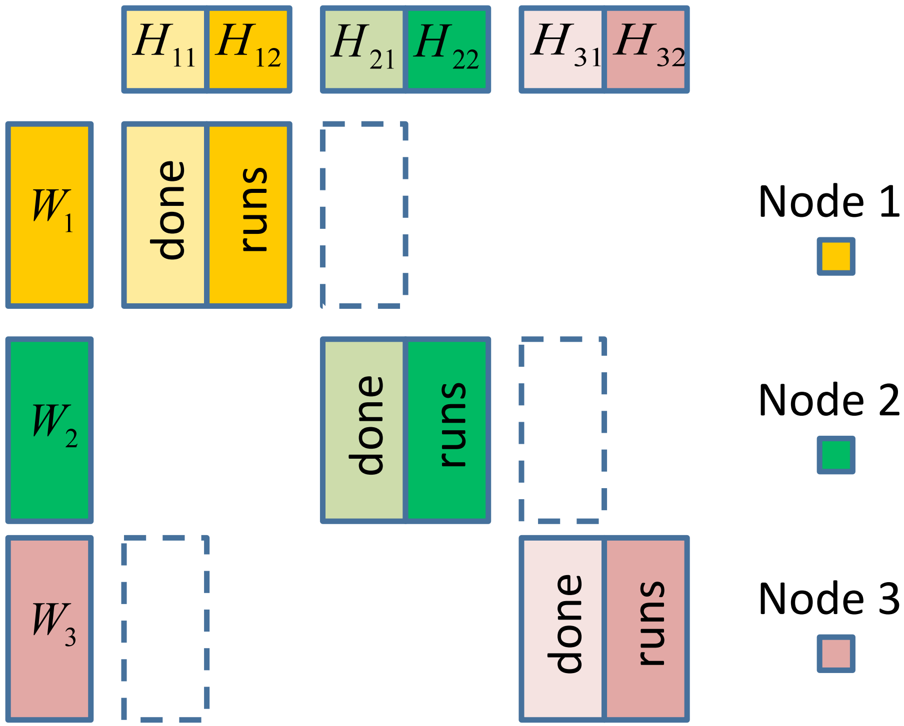


DSGD++: Overlay computation and communication

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- No overlapping of communication and computation
- No shared memory

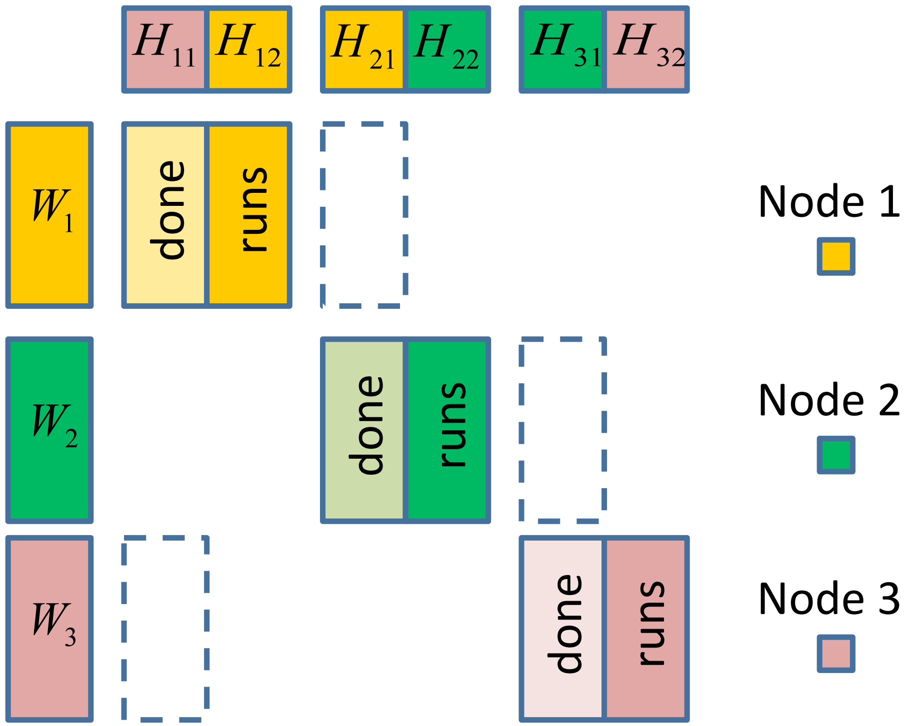


DSGD++: Overlay computation and communication

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- ~~No overlapping of communication and computation~~
- No shared memory

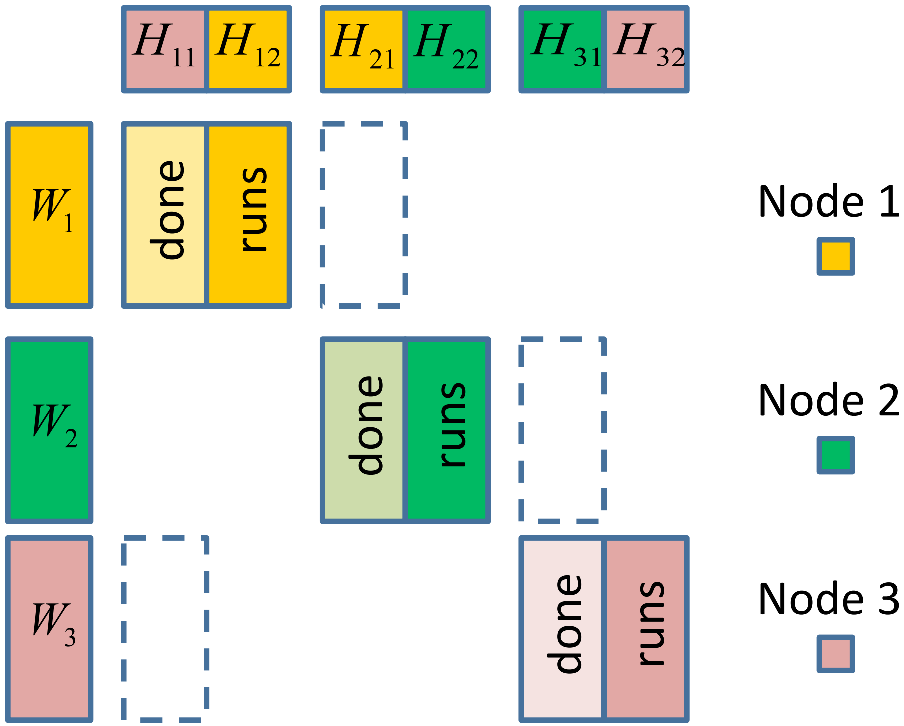


DSGD++: Overlay computation and communication

How to do better in a shared-nothing environment?

DSGD-MR drawbacks:

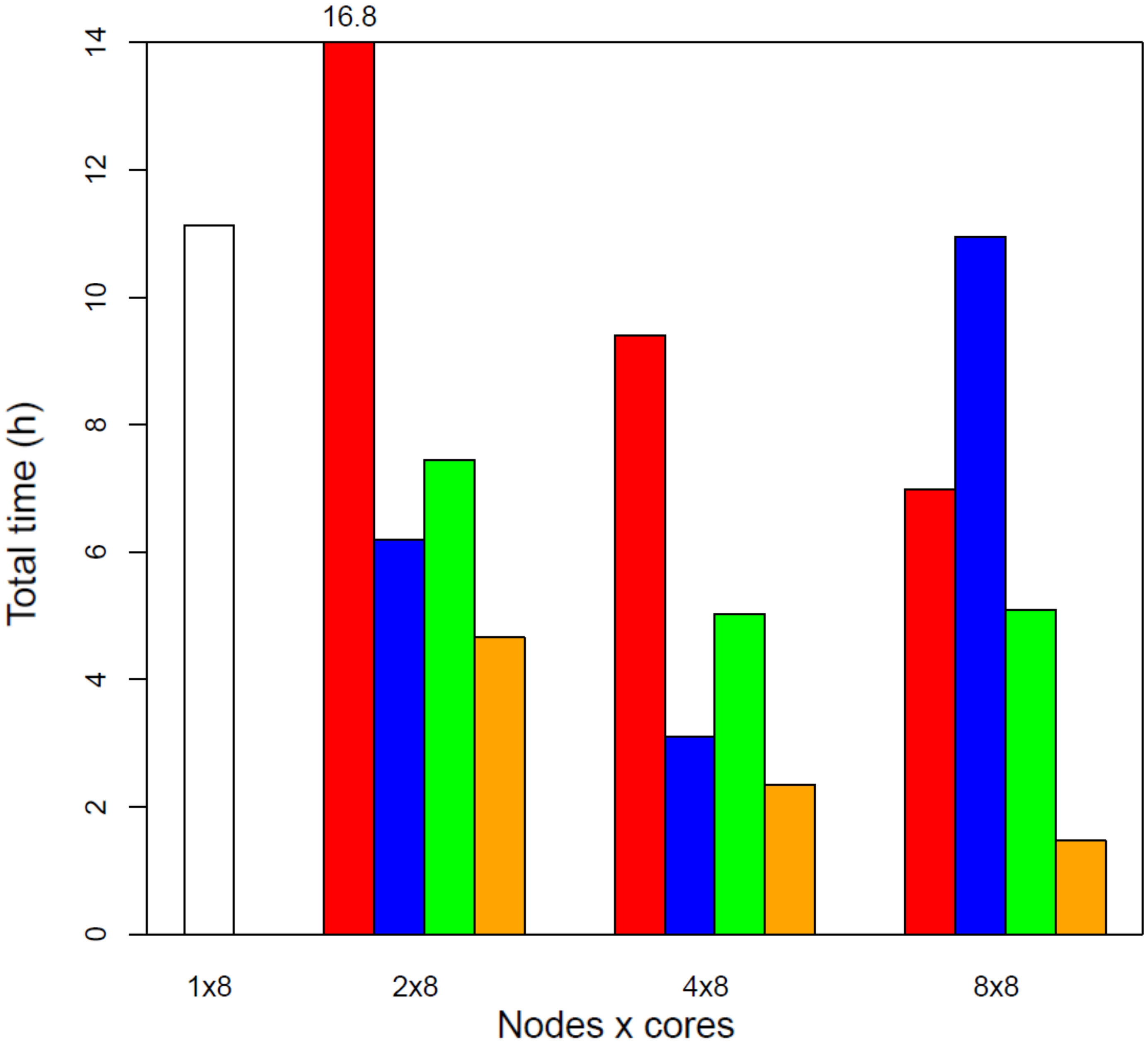
- ~~Repeatedly reads/writes from/to disk~~
- ~~Synchronous~~
- ~~No overlapping of communication and computation~~
- ~~No shared memory~~



Outline

- Motivation
- Algorithms
  - – Distributed Alternating Least Squares
  - – Distributed SGD-based algorithms
    - Asynchronous SGD
    - DSGD-MR
    - DSGD++
- **Experimental Results**
- Summary

Large Data

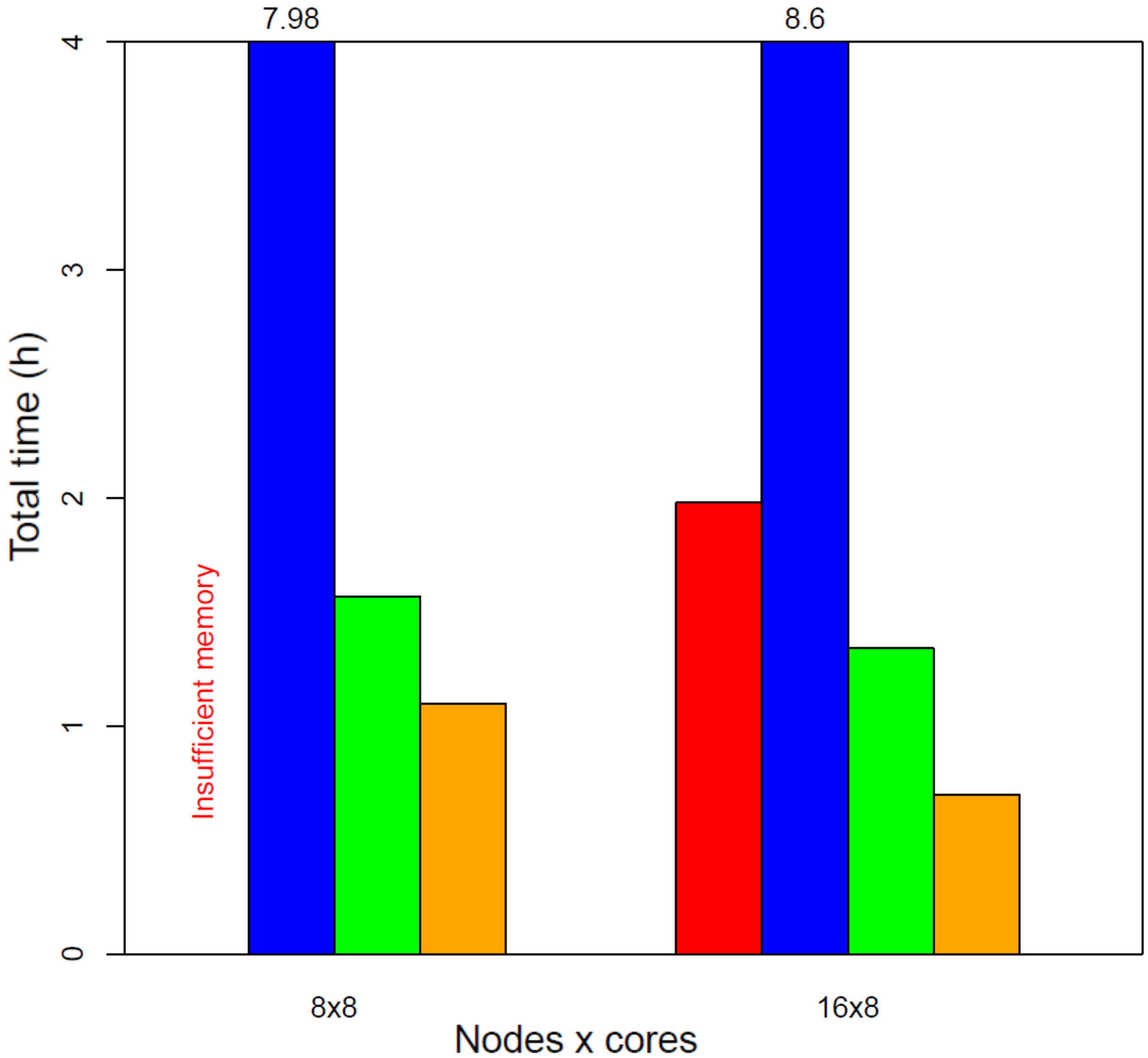


Dataset: 10M x 1M,
1B entries

- Parallel SGD
- Distributed ALS
- Asynchronous SGD
- DSGD−MapReduce
- DSGD++

DSGD++:
- Fastest
- Best scalability

Very Large Data



Dataset: 10M x 1M,
10B entries

- Parallel SGD
- Distributed ALS
- Asynchronous SGD
- DSGD−MapReduce
- DSGD++

DSGD++:
faster on 8 nodes
than any other
method on 16 nodes

Outline

- Motivation
- Algorithms
  - Distributed Alternating Least Squares
  - Distributed SGD-based algorithms
    - Asynchronous SGD
    - DSGD-MR
    - DSGD++
- Experimental Results
- **Summary**

Summary

- Existing distributed algorithms for matrix completion mainly designed for MapReduce
- Distributed algorithms for a shared-nothing environment :
  - Direct communication of nodes
  - Asynchronous
  - Overlay computation and communication
  - Multi-threading
- DSGD++:
  - Scales better
  - Can reach superlinear speed-ups
  - Low memory footprint
  - 10M x 1M with 10B entries: ~40min on 16 nodes

Summary

- Existing distributed algorithms for matrix completion mainly designed for MapReduce
- Distributed algorithms for a shared-nothing environment :
  - Direct communication of nodes
  - Asynchronous
  - Overlay computation and communication
  - Multi-threading
- DSGD++:
  - Scales better
  - Can reach superlinear speed-ups
  - Low memory footprint
  - 10M x 1M with 10B entries: ~40min on 16 nodes

Thank you
Questions?