Exploiting iterative-ness for parallel ML computations

Henggang Cui

Alexey Tumanov, Jinliang Wei, Lianghong Xu, Wei Dai, Jesse Haber-Kucharsky, Qirong Ho, Gregory R. Ganger, Phillip B. Gibbons (Intel), Garth A. Gibson, Eric P. Xing

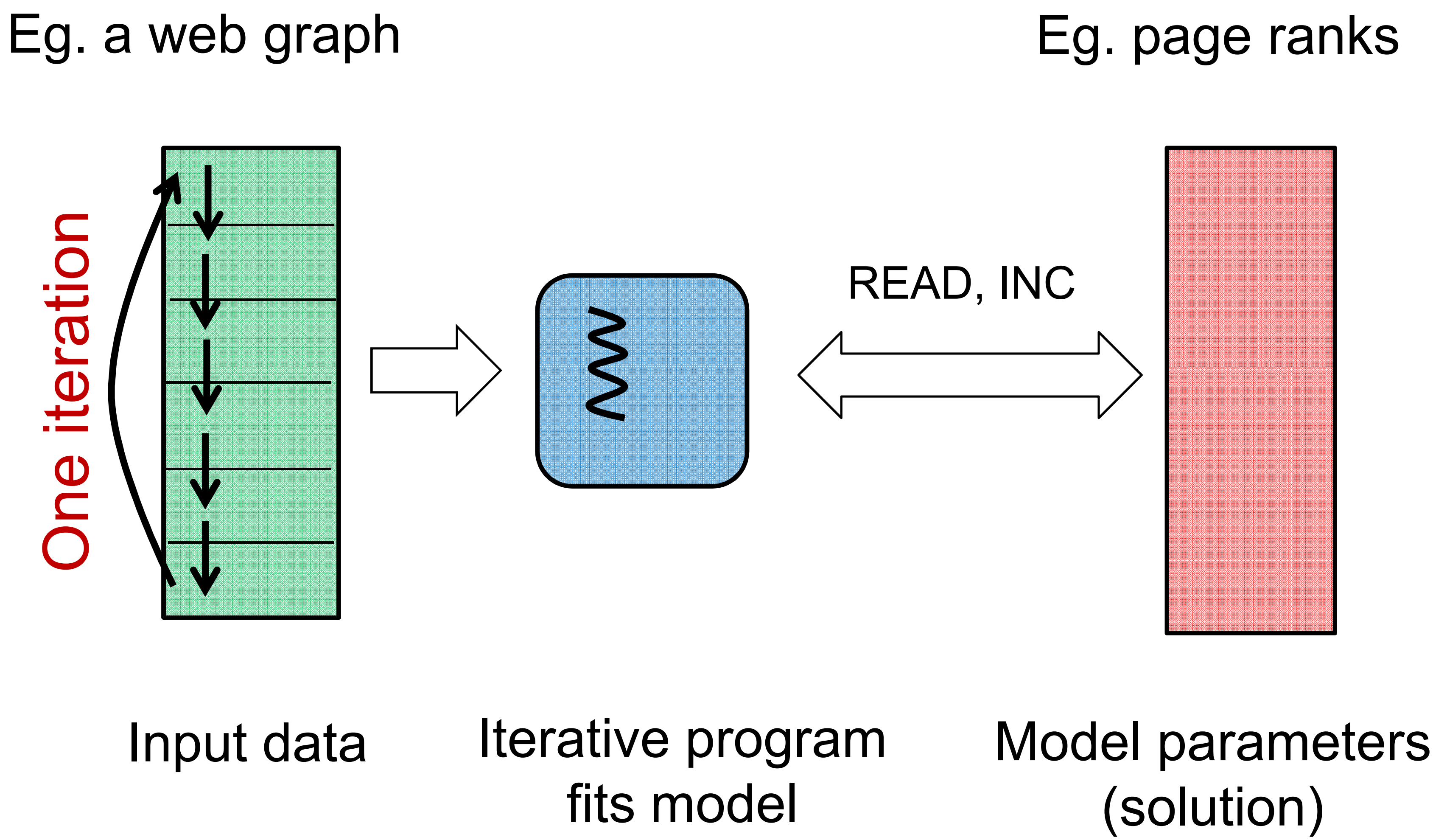
PARALLEL DATA LABORATORY

Carnegie Mellon University

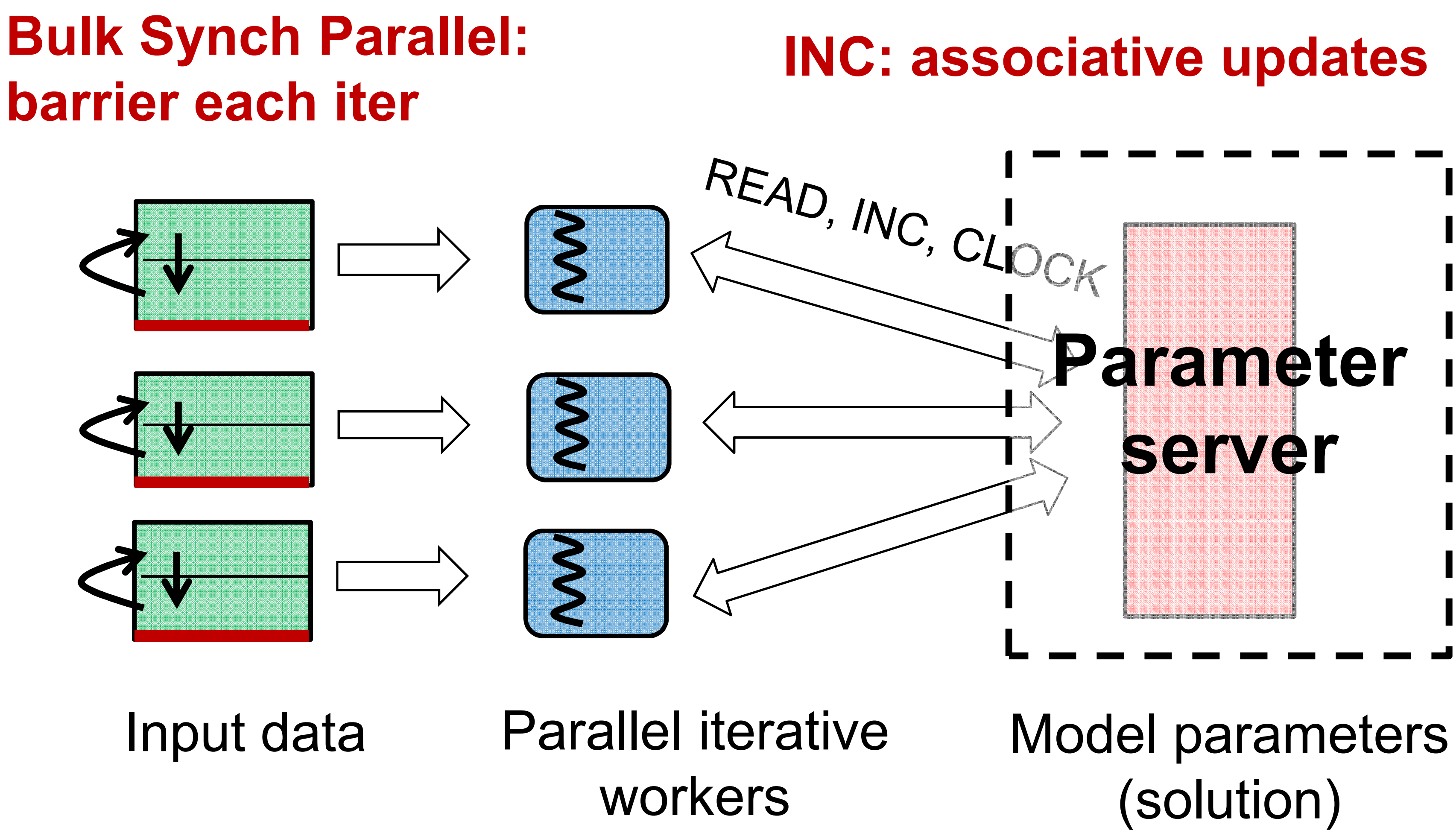
One slide overview

- Iterativeness arises in some ML apps
  - Consequence: repeated data operation sequences
- Repeating pattern can be exploited
  - Detect with minor effort
    - Either in a real or a "virtual" iteration
  - Specialize structures and policies to known pattern
    - Data partitioning, prefetching, lock avoidance, pre-marshalled structures, etc.
- Next
  - Parallel machine learning
  - PageRank as one example

Parallel machine learning

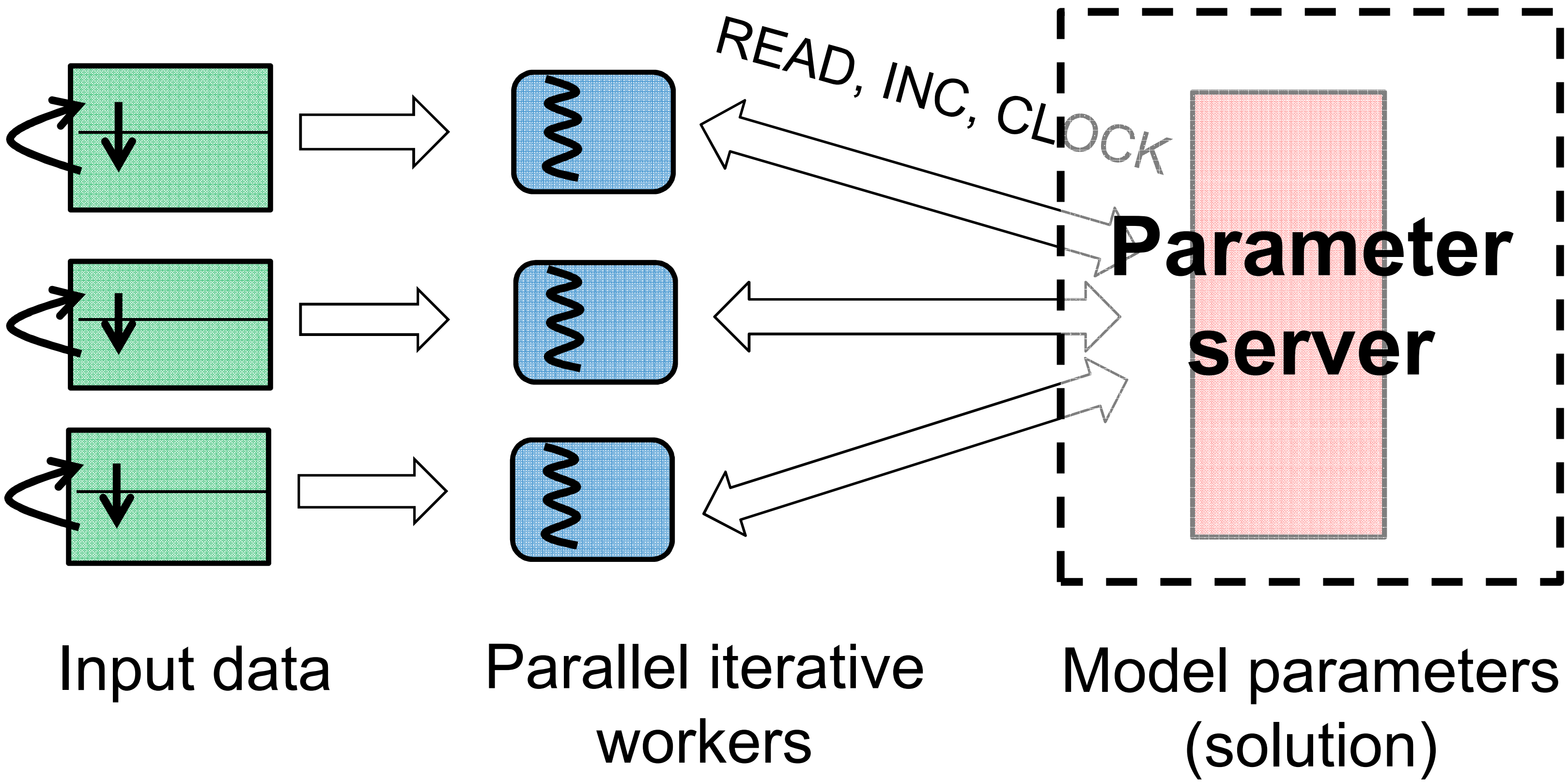


Parallel machine learning



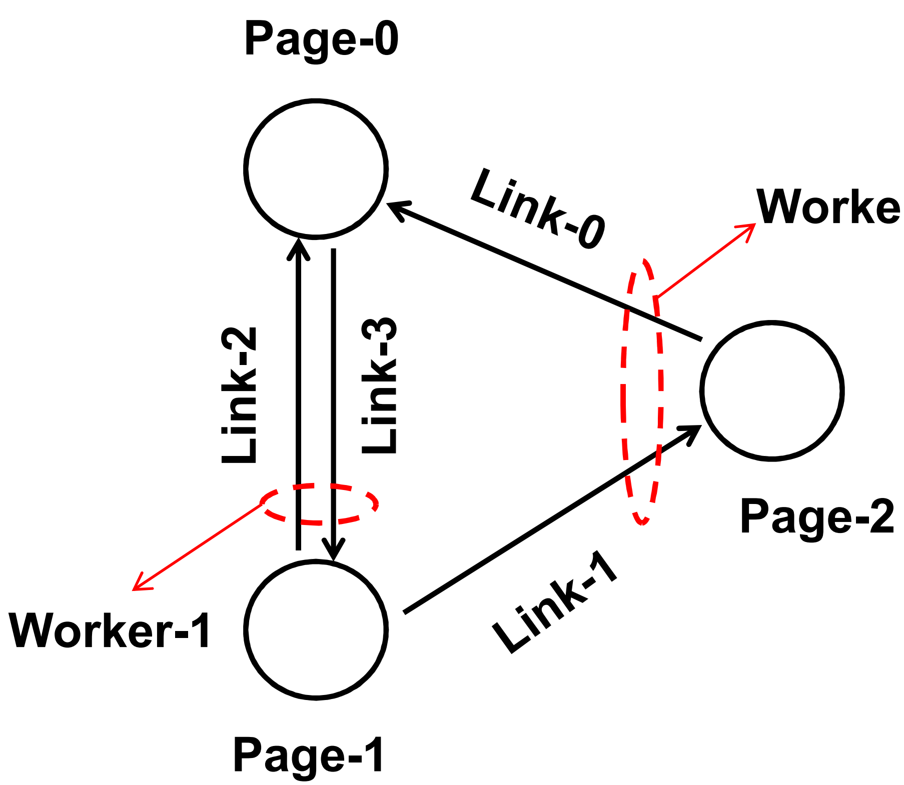
Parallel machine learning

Goal: improve performance by exploiting iterativeness



Example: PageRank

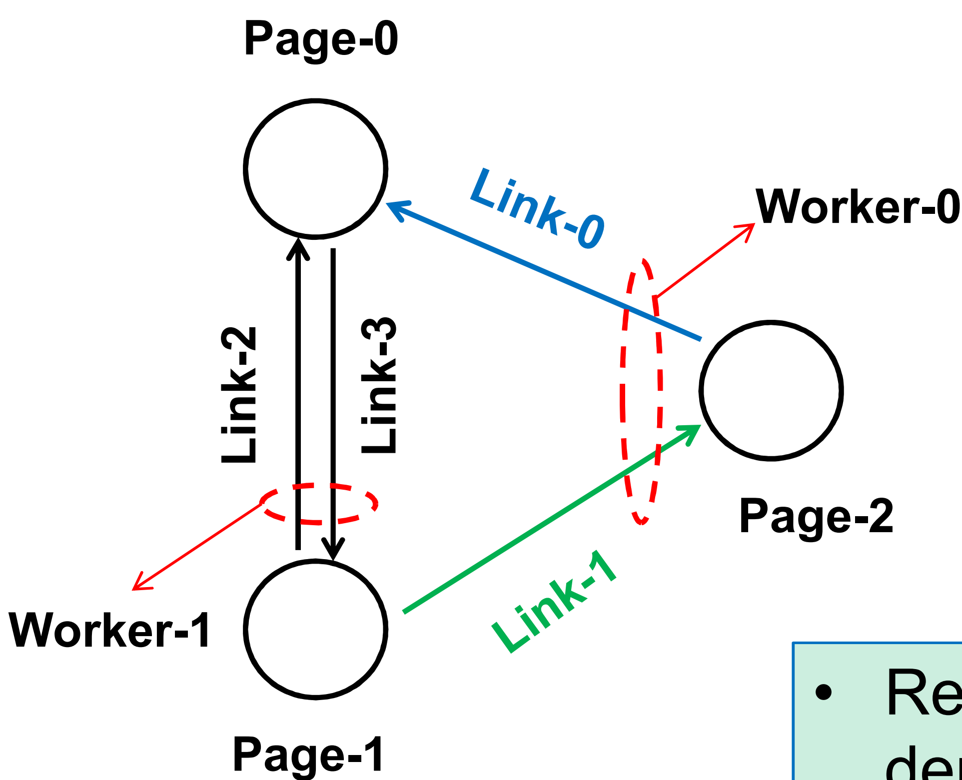
Input data: a set of links, stored locally
Parameter data: ranks of pages, stored in parameter server




```
All ranks set to some value
LOOP
  FOREACH link from i to j
    read Rank(i)
    Rank(j) += change of Rank(i)
  ENDFOREACH
WHILE NOT CONVERGE
```

Example: PageRank

Input data: a set of links, stored locally
Parameter data: ranks of pages, stored in parameter server



Worker-0:

```
LOOP
  # Link-0
  READ page[2].rank
  INC page[0].rank
  # Link-1
  READ page[1].rank
  INC page[2].rank
  CLOCK
WHILE NOT CONVERGE
```

- Repeated operation sequence depends only on input data
  - Does not depend on ranks

Repeated operation sequences

- Many examples of ML applications
  - Including Topic Modeling and Collaborative Filtering
- Knowledge of repeated operation sequence can be exploited to improve efficiency
  - 50x speed up for PageRank
- Talk outline
  - Ways to obtain per-iter operation sequences
  - Optimizations with pre-knowledge of operations
  - Experiment results

Obtain per-iter operation sequences

- Parameter server operations
  - READ
  - INC
  - CLOCK
    - Can be thought of as barrier

```
LOOP
  READ page[2].rank
  INC page[0].rank
  READ page[1].rank
  INC page[2].rank
  CLOCK
WHILE NOT CONVERGE
```

- Two ways of obtaining it
  - Gather in the first iteration
  - Gather in a *"virtual iteration"*

Gather in the first iteration

```
// Original
load_data()
init_param_vals()
do {
  do_iteration()
} while (not stop)
```

```
// Gather in first iter
load_data()
init_param_vals()
do {
  if (first iteration)
    ps.start_gather()
  do_iteration()
  if (first iteration)
    ps.finish_gather()
} while (not stop)
```

Carnegie Mellon
Parallel Data Laboratory

Gather in the first iteration

+ Little programmer effort
  - Only need to annotate iteration boundaries

- Considerable performance overhead
  - The first iteration runs without optimizations
  - More cost to apply the optimizations
    - States from the first iteration need to be migrated

Gather in a *virtual* iteration

Just to remind you

```
// Gather in first iter
load_data()
init_param_vals()
do {
  if (first iteration)
    ps.start_gather()
  do_iteration()
  if (first iteration)
    ps.finish_gather()
} while (not stop)
```

```
// Gather in virtual iter
load_data()
ps.start_gather(virtual)
do_iteration()
ps.finish_gather()
init_param_vals()
do {
  do_iteration()
} while (not stop)
```

- Operations between start_gather(virtual) and finish_gather() are recorded but return *without any action. Nearly free.*

Gather in a *virtual* iteration

− Programmer needs to be more careful

- do_iteration() needs to work with *virtual* READ/INC
  - Computation must be independent of param value

+ Better performance

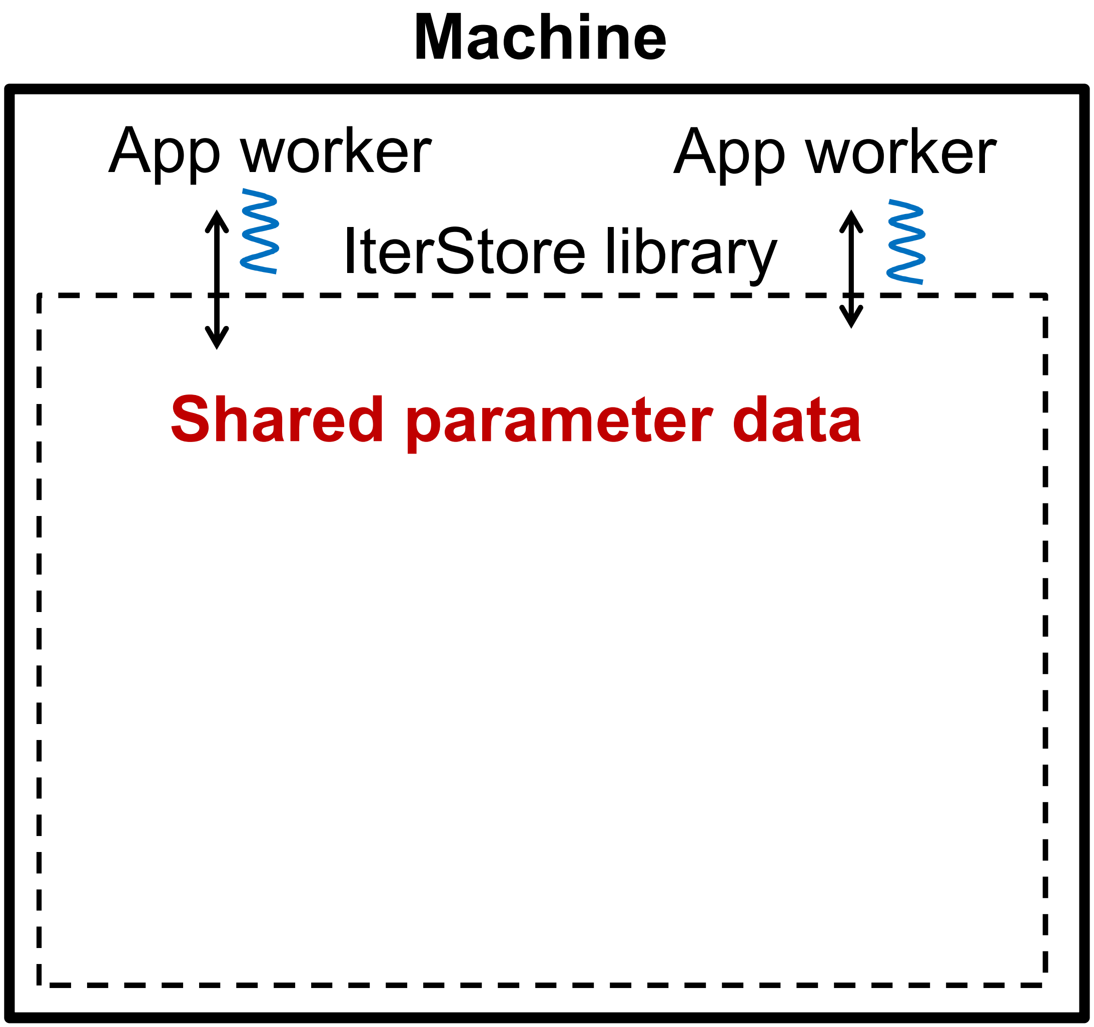
- No operations performed during virtual iteration
- No state migration
- All real iterations run at optimized speed

Optimizations on informed access

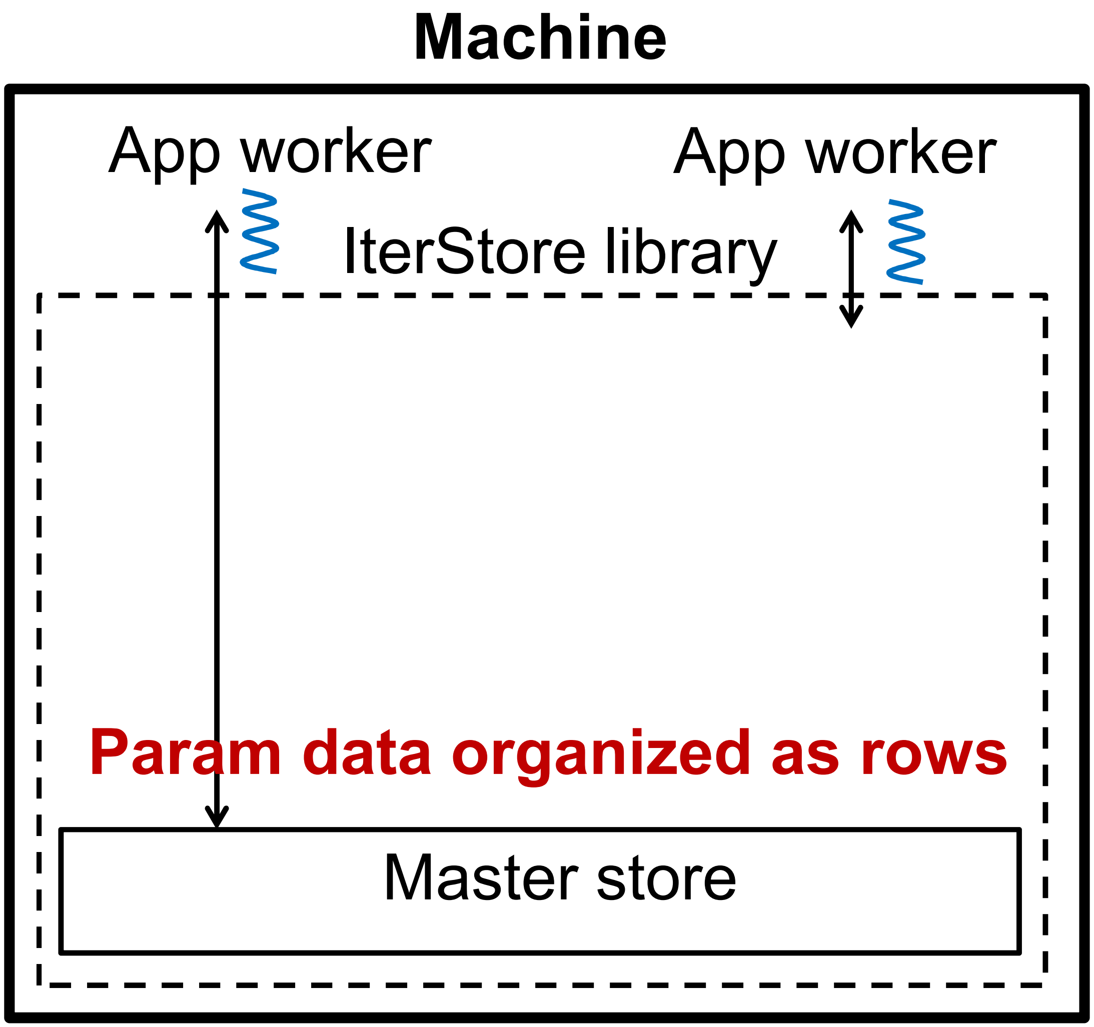
- Optimizations applied at finish_gather()
  1. Cross-machine parameter data placement
  2. Prefetching
  3. Static cache policies
  4. More efficient static data structures
  5. NUMA-aware memory management

- Prototyped on IterStore
  - A "*parameter server library*"
  - An improved version of LazyTable

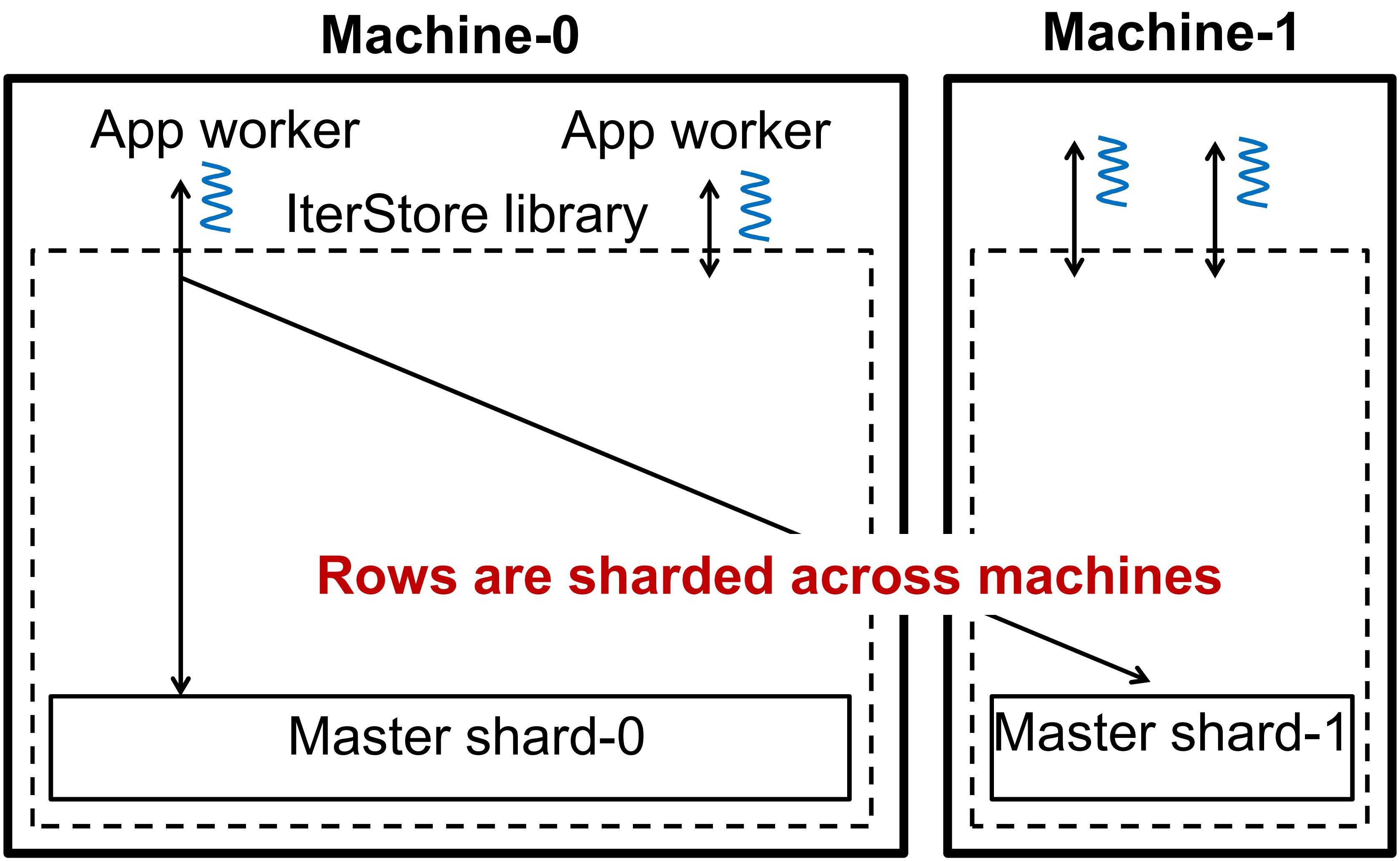
IterStore architecture



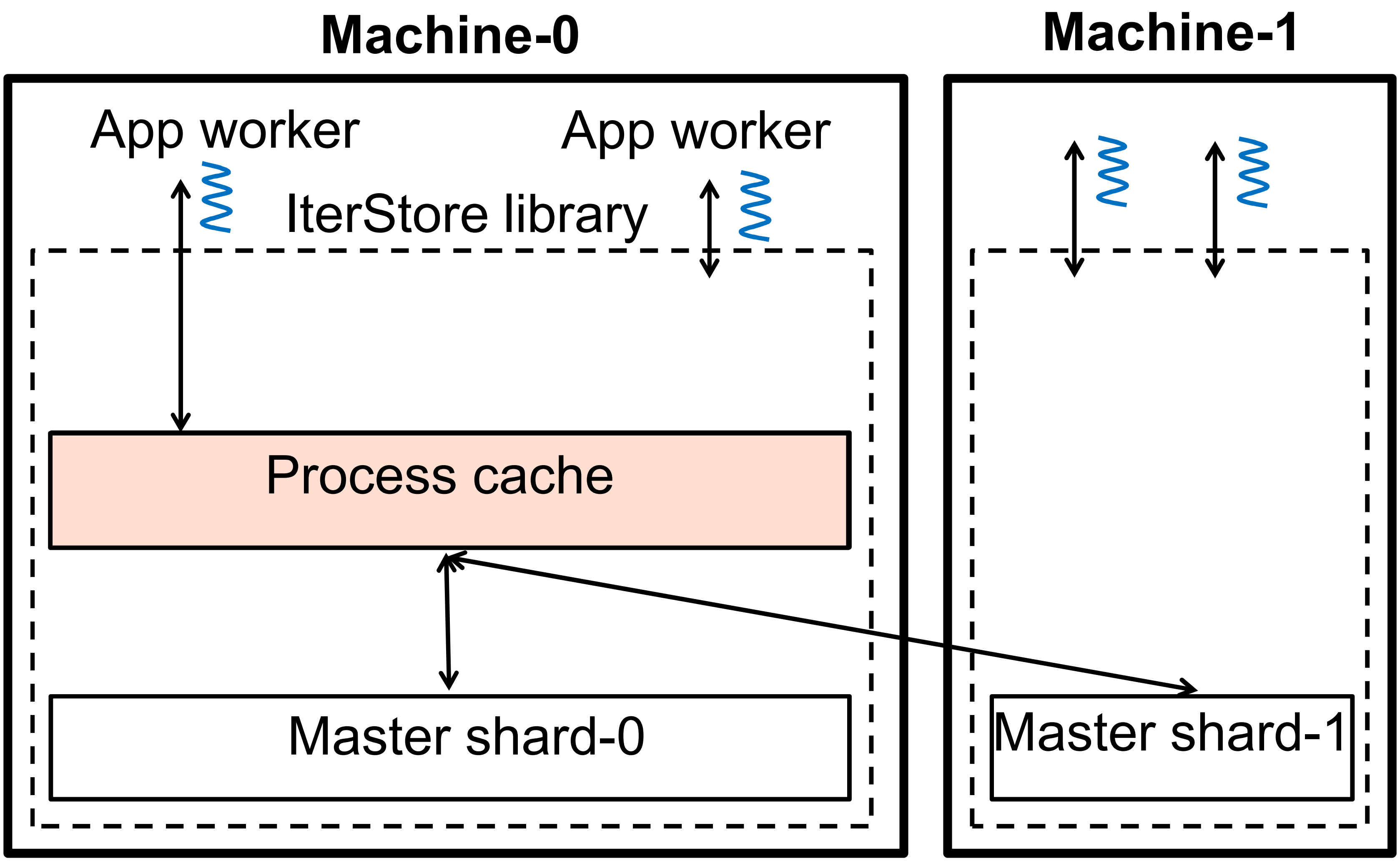
IterStore architecture



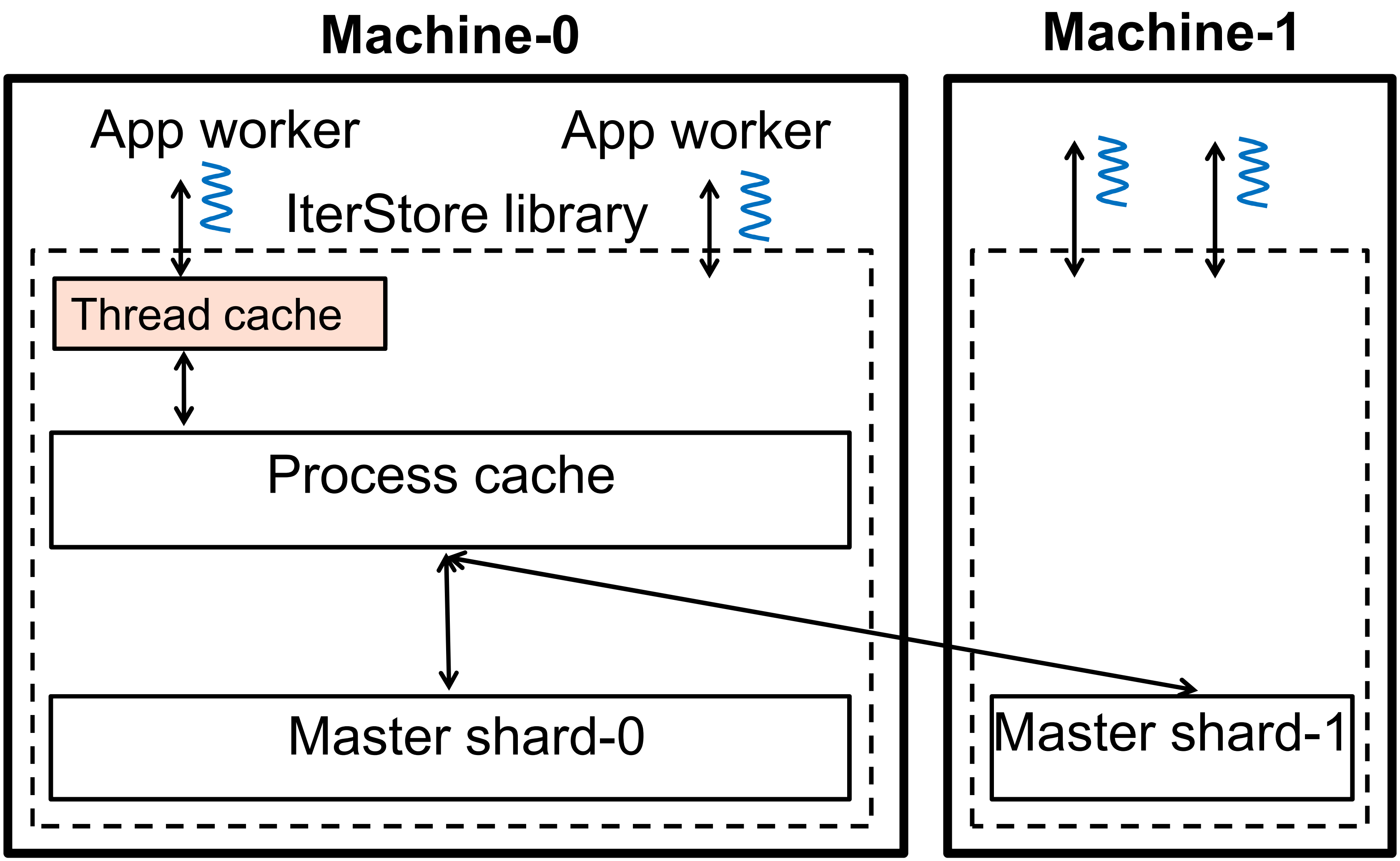
IterStore architecture



IterStore architecture

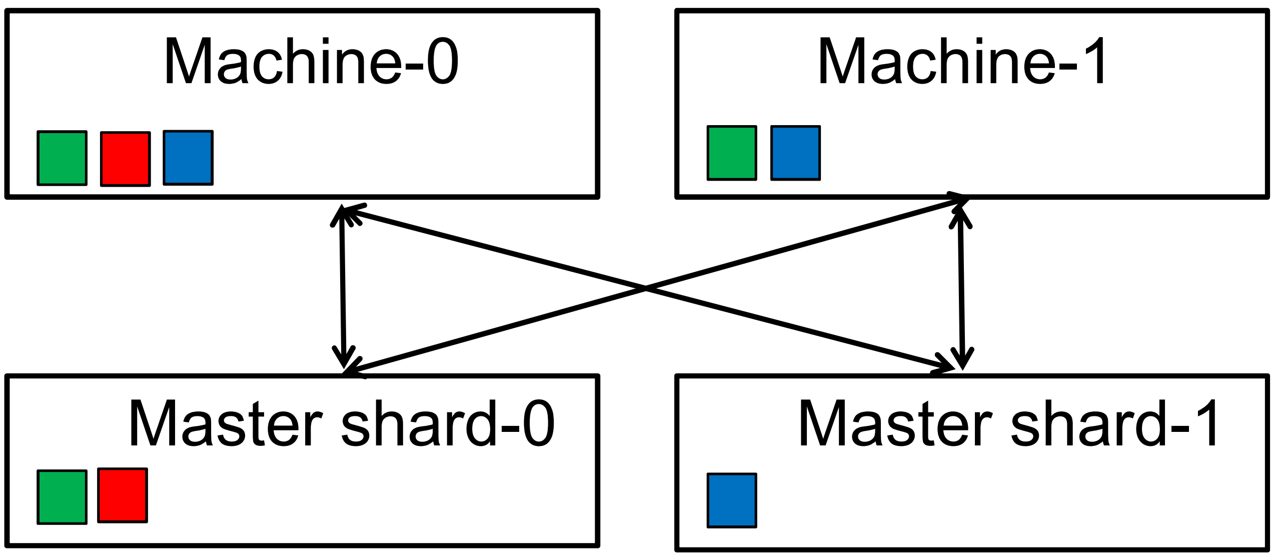


IterStore architecture



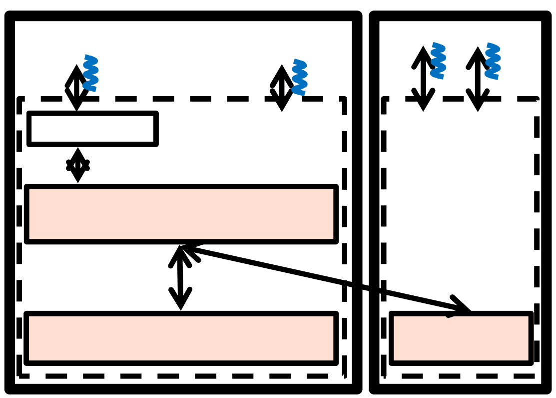
1: Parameter data placement

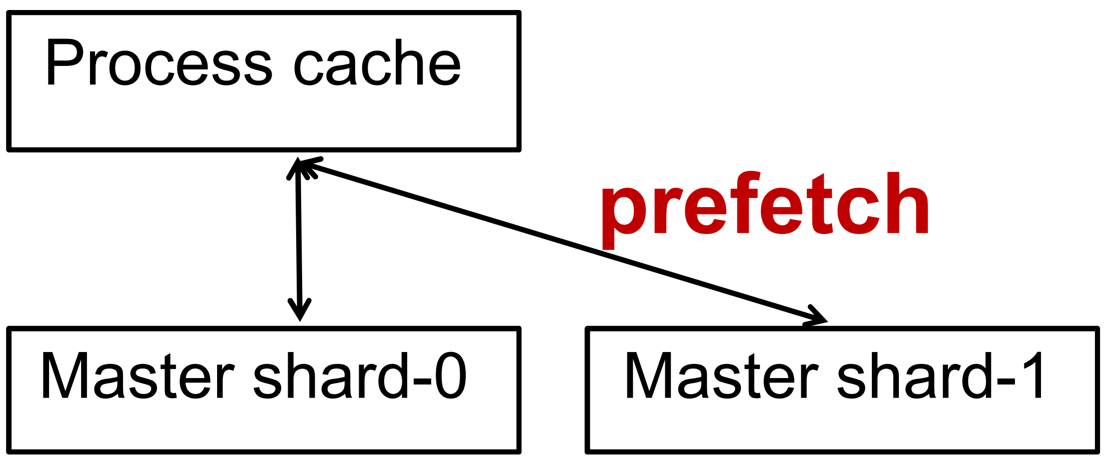
- Cross-machine parameter data placement
  - Store each row at the machine accessing it most
  - Balance the load for rows without clear affinity



2: Prefetching

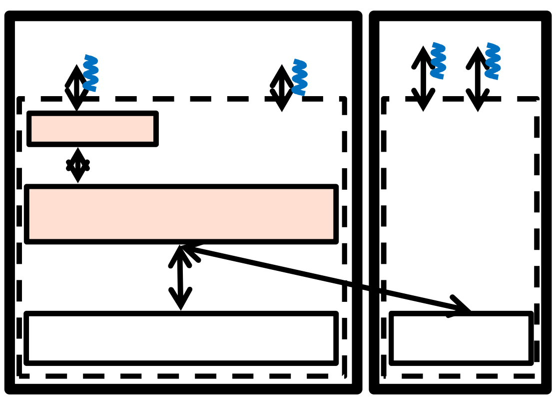
- Prefetching
  - Prefetch to process cache at the beginning of clock
    - – Rows expected to be read in the clock
  - Fetched in a single message

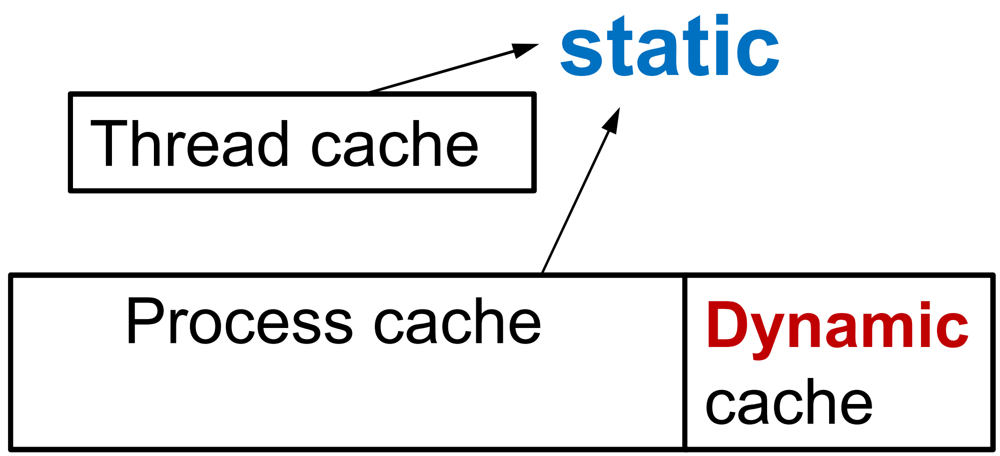




3: Static cache policies

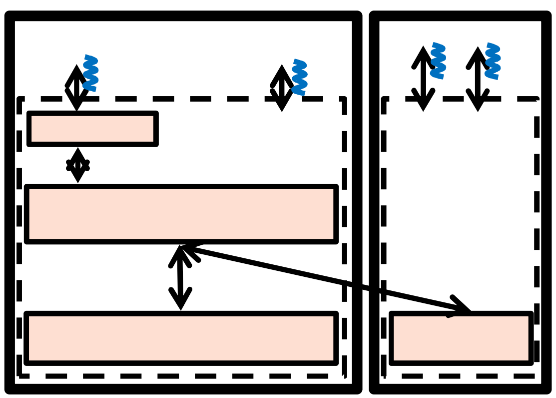
- Static cache policies
  - Decide rows to be cached based on access sequence
  - Cache rows with higher utilities
  - Never evict rows, no cache eviction overhead
  - Use a 2nd (dynamic) cache for items not in static cache





4: Static data structures

- Static hash map
  - Immutable index
    - No global lock needed for index concurrency
  - Entries stored in a contiguous block of memory
    - Can be sent in a single message without marshalling

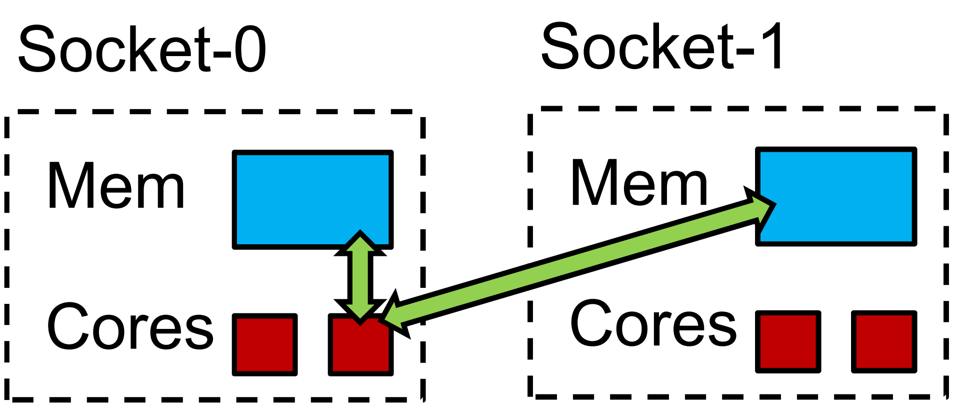


- Thread cache and master shared
  - Hash maps
  - Each accessed by one thread
- Process cache
  - Concurrent hash map

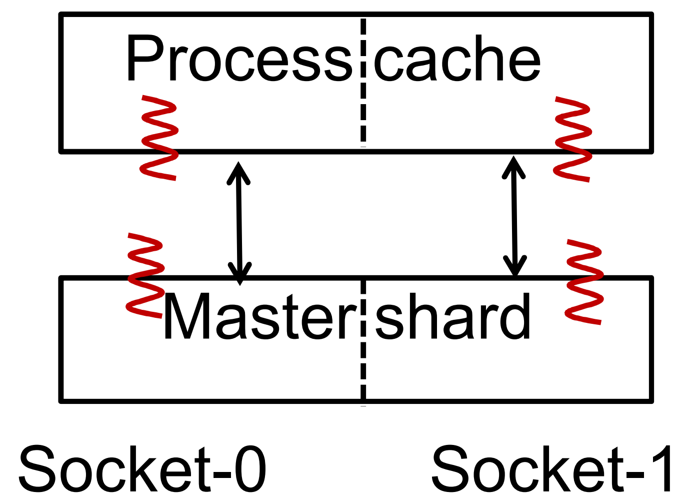
Carnegie Mellon
Parallel Data Laboratory

5: NUMA memory management

- NUMA effect in multi-socket machines
  - Lower latency to access local memory
- Partition cache and master store structures
  - Place each partition local to managing threads



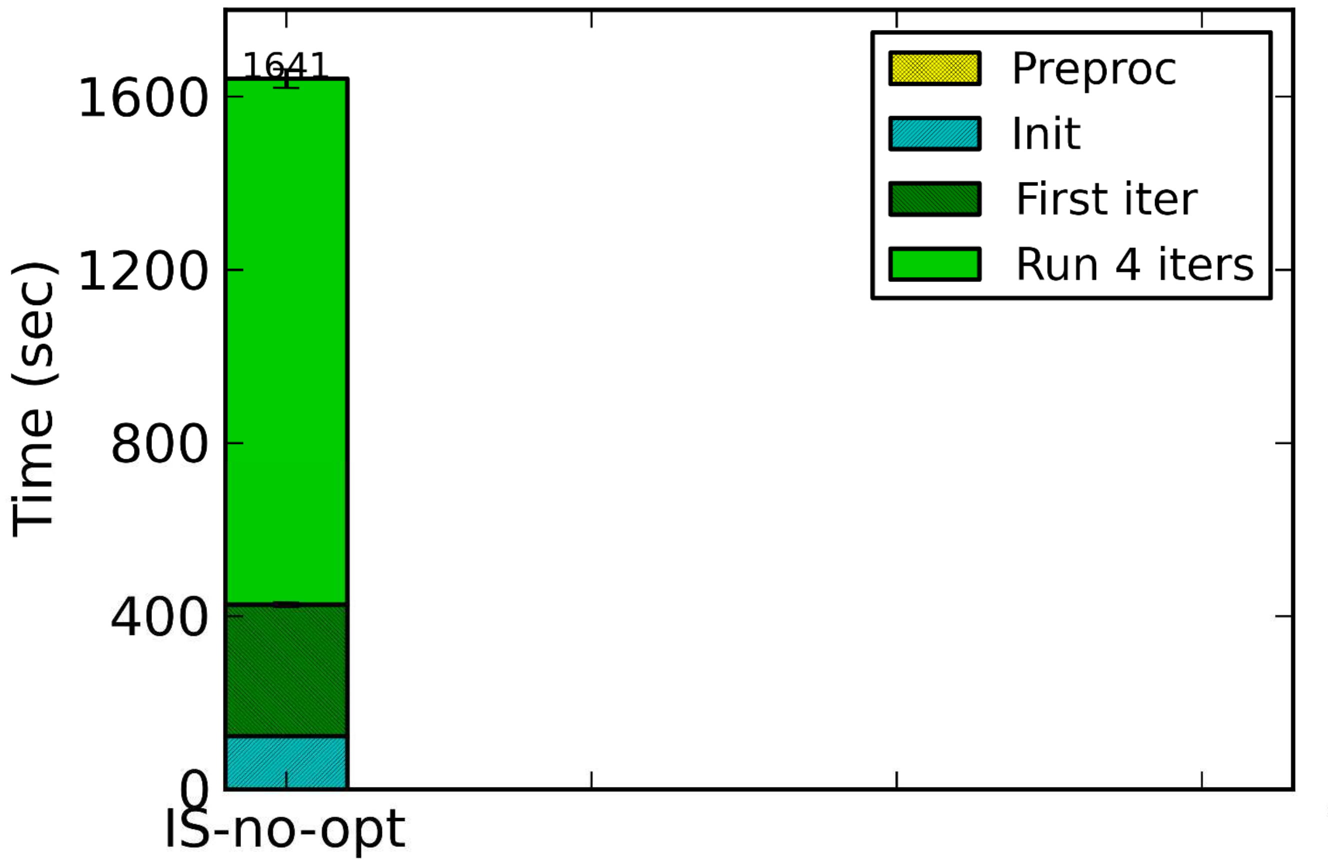
Faster to access local memory



Experiment setup

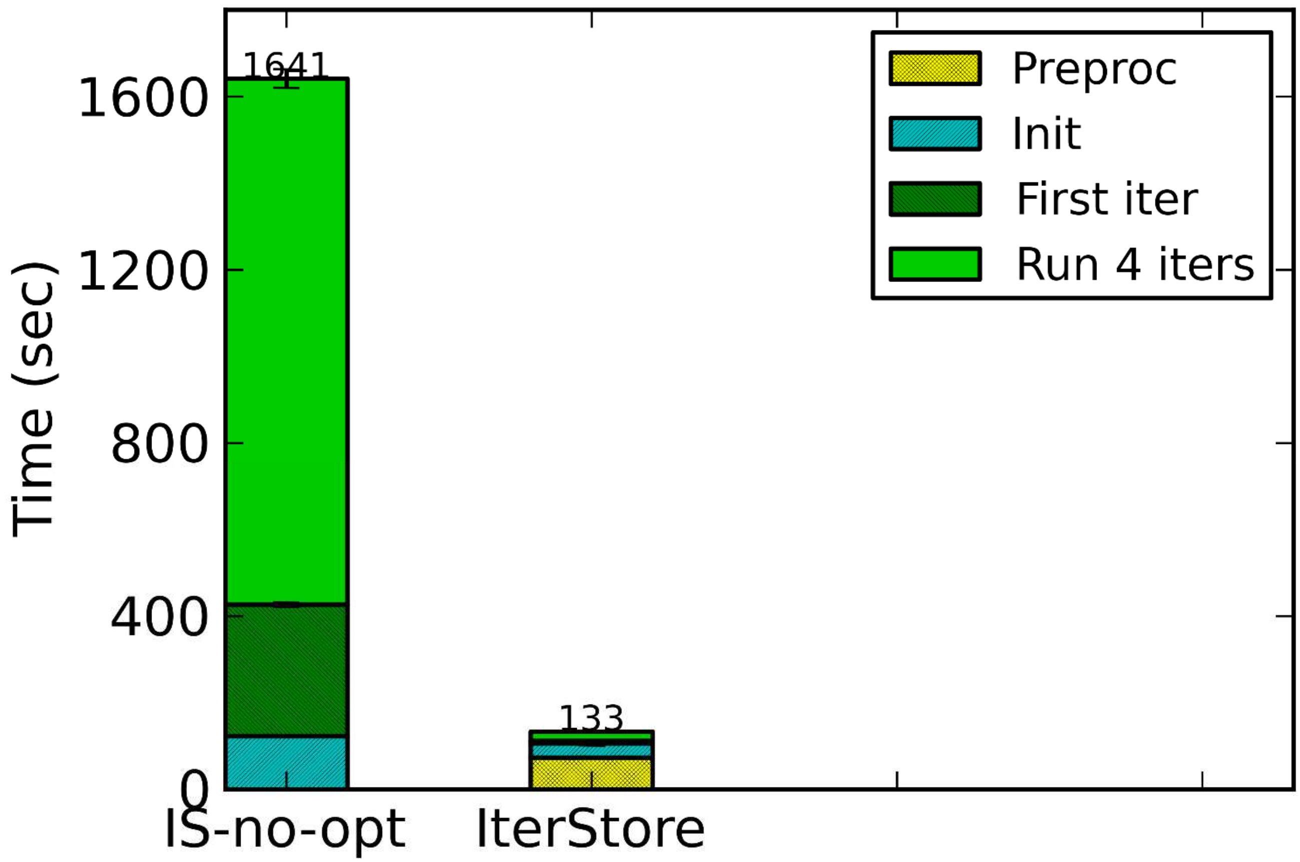
- Cluster information
  - 8 machines, each with 64 cores & 128GB RAM
  - 64 application worker threads per machine
- Application benchmarks
  - PageRank:
    `twitter-graph` (40m nodes, 1.5b edges)
  - Collaborative Filtering:
    `netflix` (480k-by-18k sparse matrix)
  - Topic Modeling:
    `nytimes` (100m tokens, 300k docs)

Overall performance: PR, 5 iters



PageRank

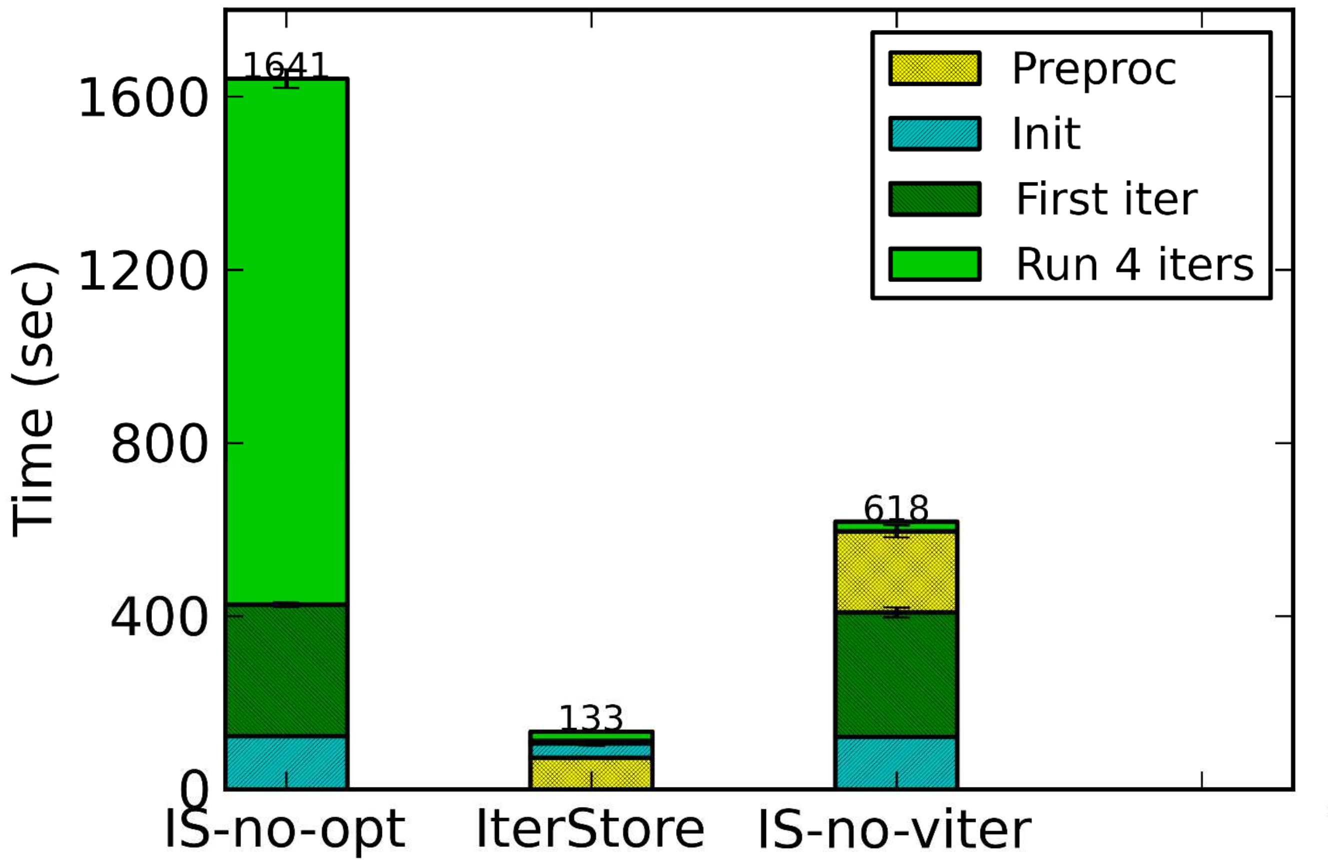
Overall performance: PR, 5 iters



PageRank

12x speed up on overall time,
50x speed up on per iteration time

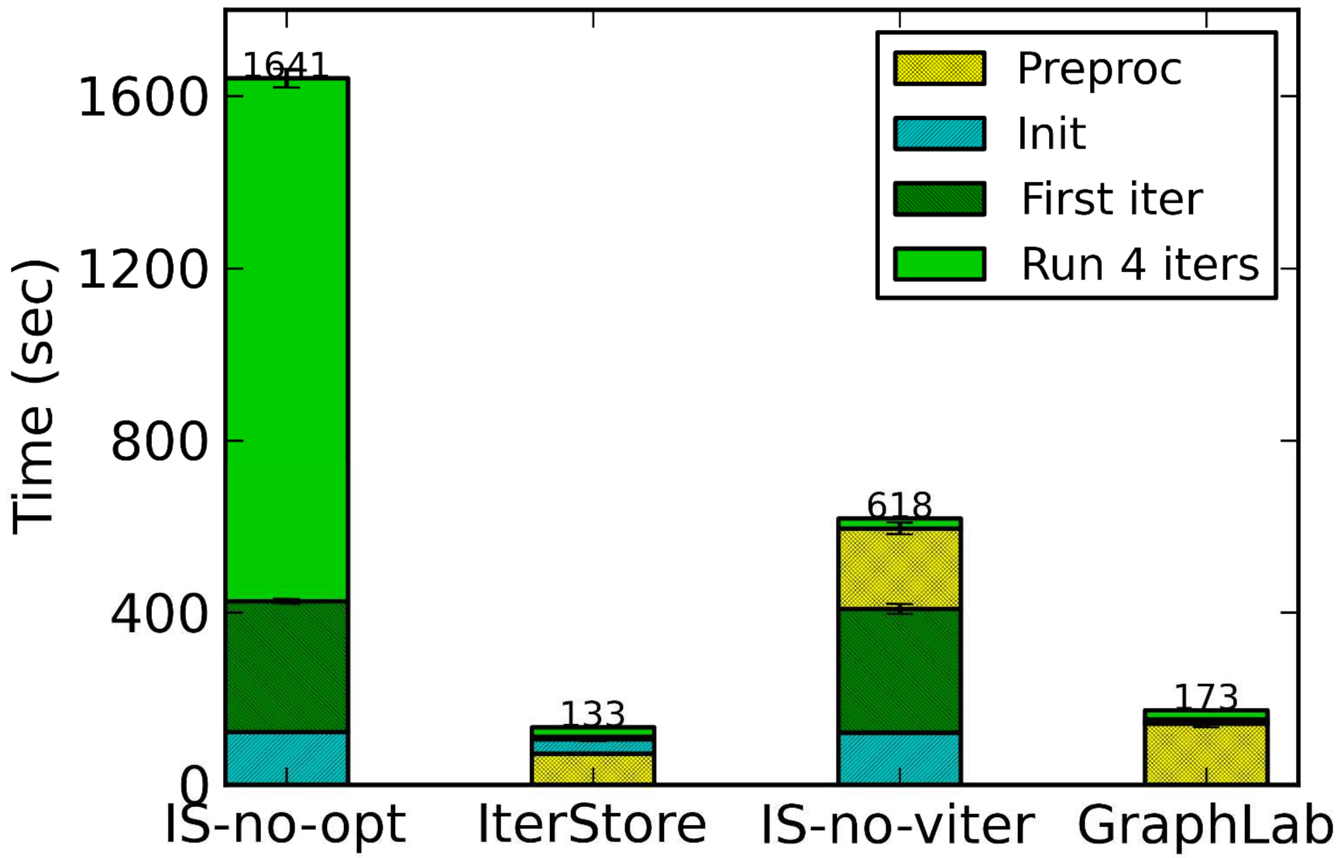
Overall performance: PR, 5 iters



PageRank

Virtual-iter gathering performs better than first-iter gathering

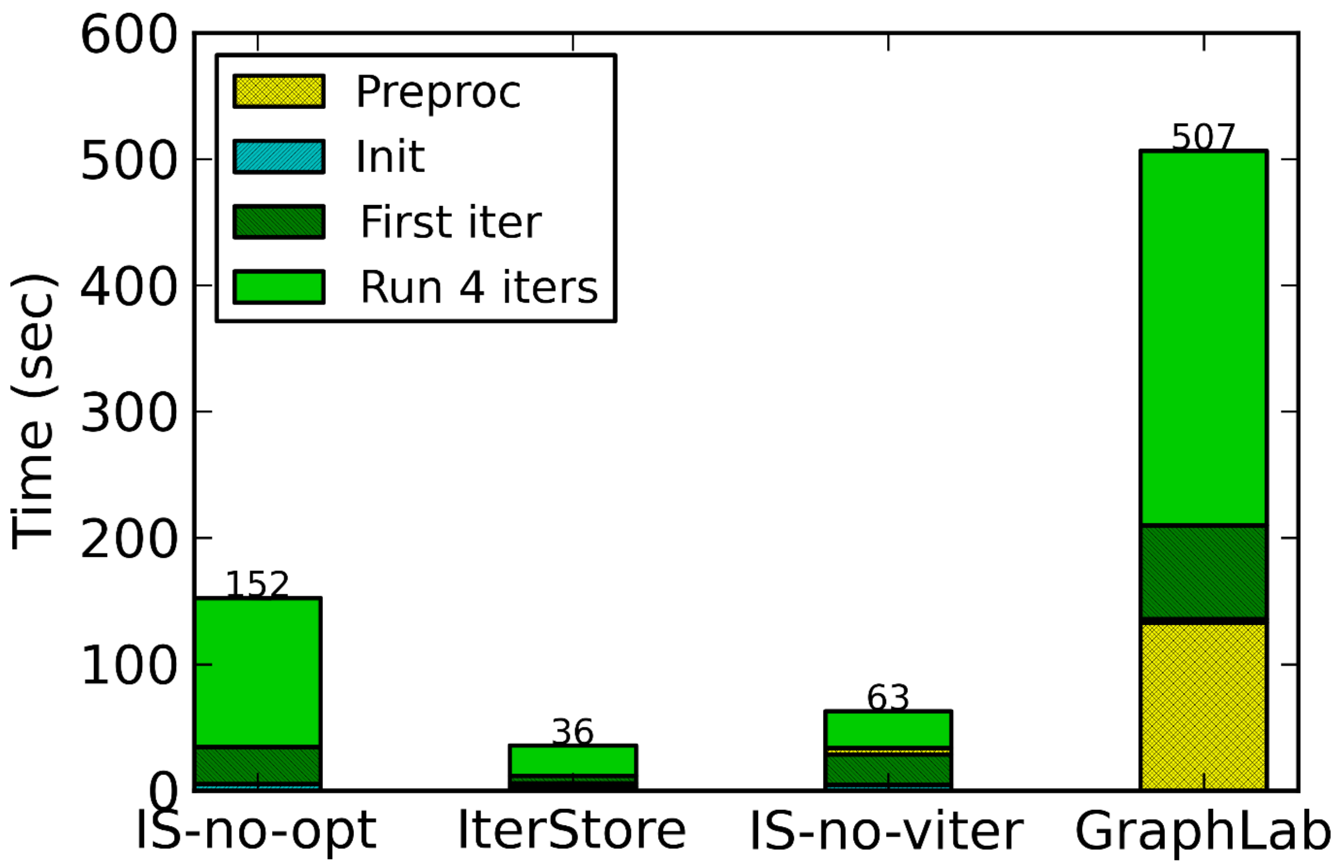
Overall performance: PR, 5 iters



PageRank

Faster than GraphLab even on PageRank

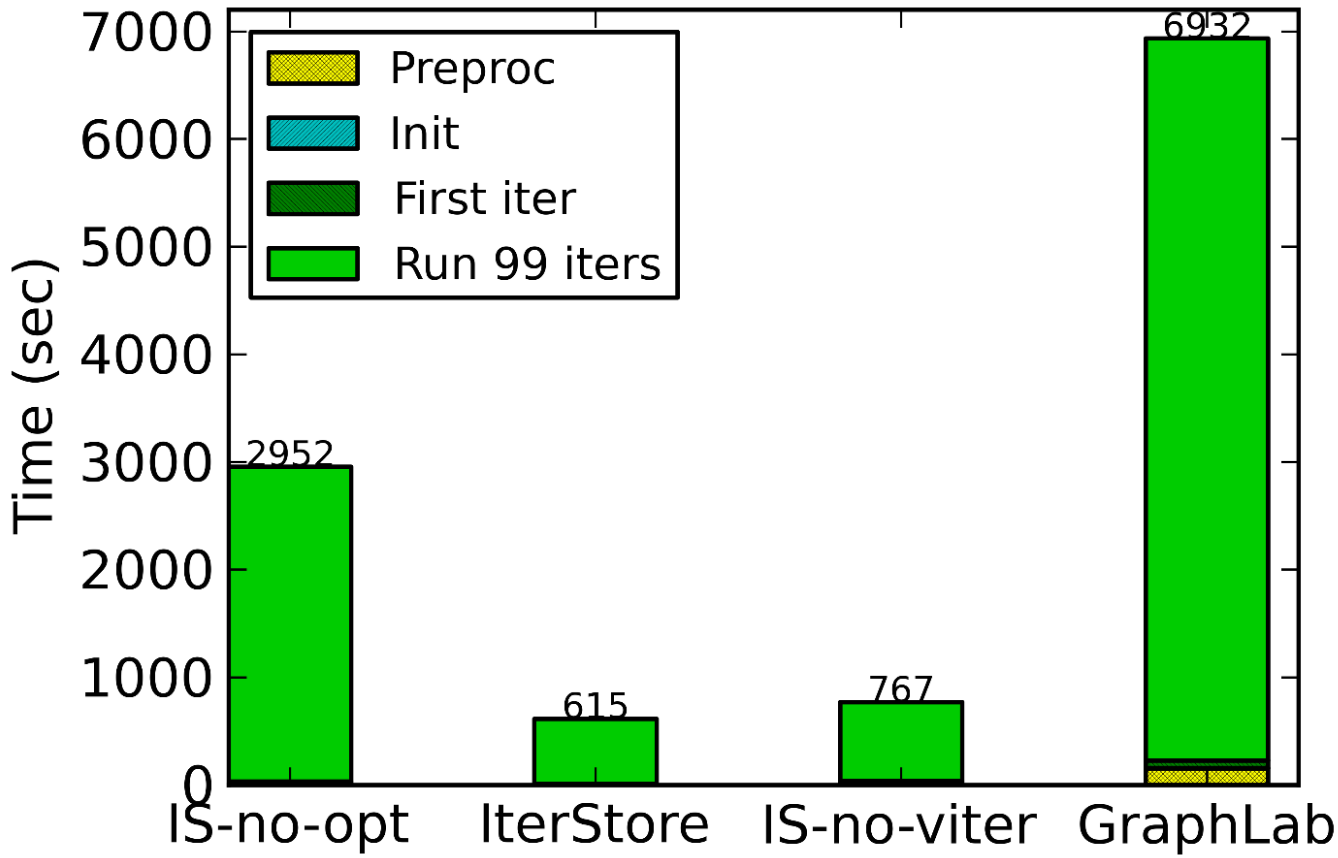
Overall performance: CF, 5 iters



Collaborative Filtering

More speed up over GraphLab

Overall performance: CF, 100 iters



Collaborative Filtering

Preprocessing time is amortized over more iterations

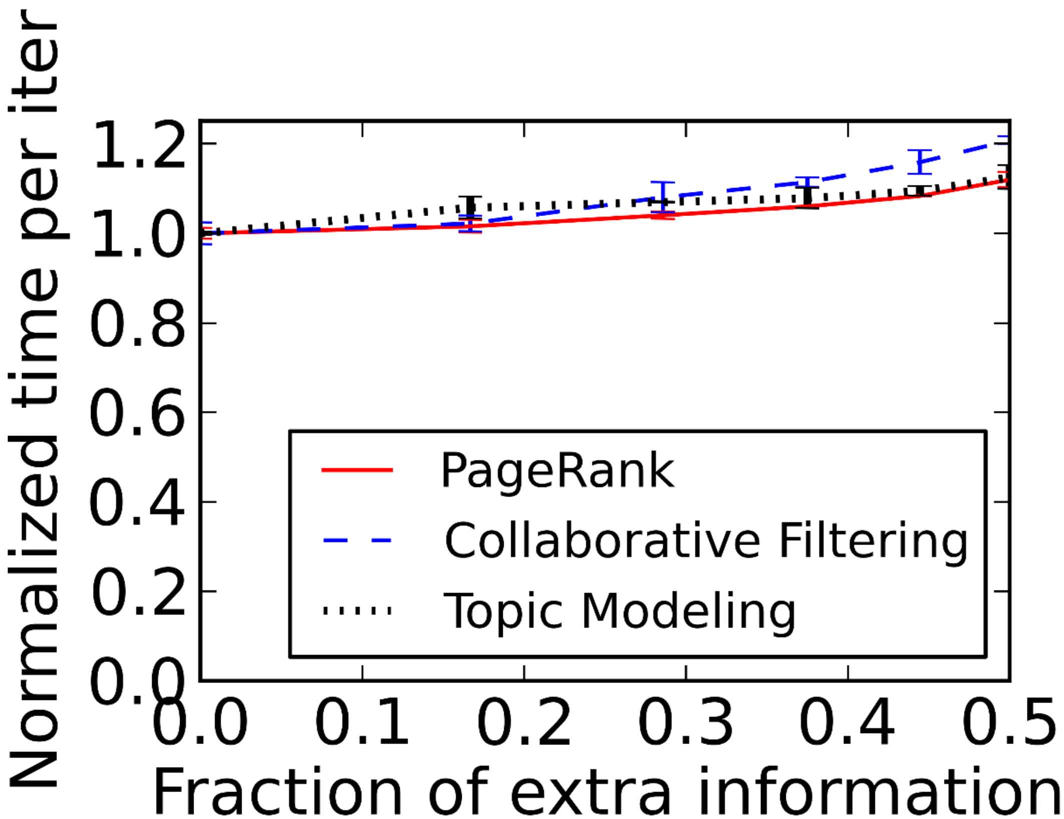
Sensitivity to information accuracy

- Inaccurate information can be caused by
  - Work migration
  - Skipped work due to parameter convergence
- Experiment method
  - Keep real operation sequences fixed
    - Report **more** operations than performed
    - Report **less** operations than performed
  - Compare normalized time per iteration
    - No inaccuracy as the baseline

Sensitivity to information accuracy

Report **more** operations than performed

- Can be caused by work migration or skipped work

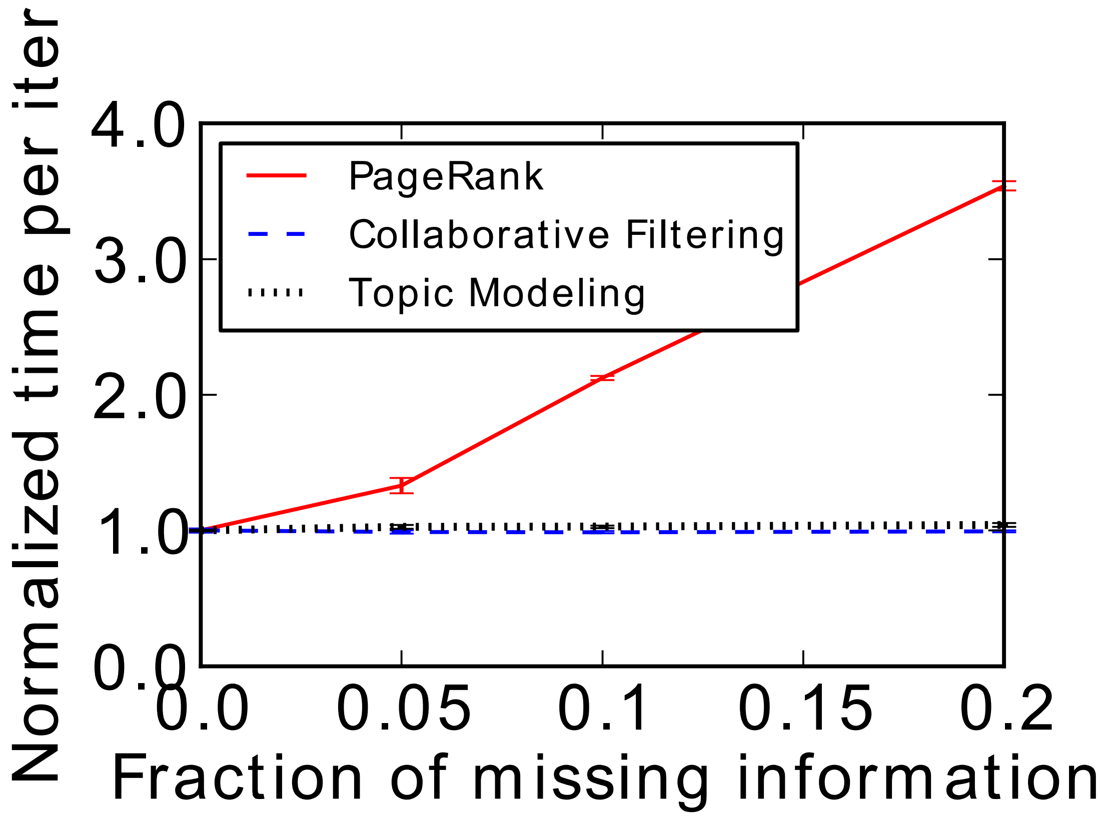


All are insensitive to extra information

Sensitivity to information accuracy

Report **less** operations than performed

- Can be caused by work migration



CF and TM are insensitive to missing information

Conclusion

- Many ML applications exhibit iterativeness
  - Same sequence of operations every iteration
- Systems can exploit repeated op sequences
  - Speed up real ML benchmarks by up to 50x
- Two ways of gathering such operation sequence
  - Better performance when doing virtual iteration

References

- **[Parameter Server]** A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In WSDM, 2012.
- **[LazyTable]** H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Exploiting bounded staleness to speed up big data analytics. In USENIX ATC, 2014.
- **[GraphLab]** J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: Distributed graph-parallel computation on natural graphs. In OSDI, 2012.
- **[Nytimes]** http: //archive.ics.uci.edu/ml/datasets/Bag+of+Words.
- **[Twitter graph]** H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In WWW, 2010.