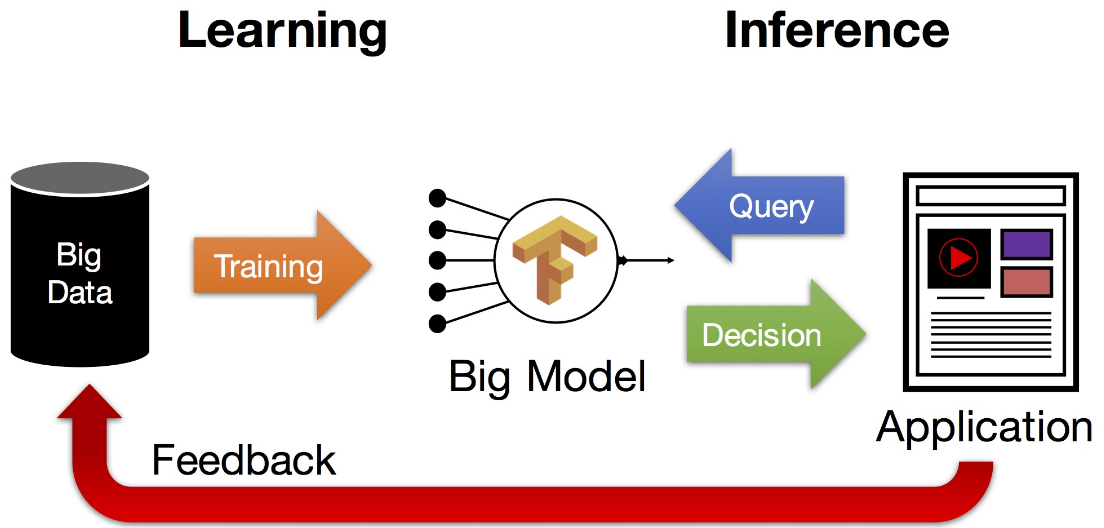
Scalable Asynchronous Gradient Descent Optimization for Out-of-Core Models

Chengjie Qin[1], Martin Torres[2], and **Florin Rusu**[2]

[1]GraphSQL, Inc.
[2]University of California Merced

August 31, 2017

Machine Learning (ML) Is Booming



https://ucbrise.github.io/cs294-rise-fa16/prediction_serving.html

- General frameworks with ML libraries: Hadoop's Mahout, Spark's MLLib, GraphLab
- Specialized ML systems: Vowpal Wabbit, SystemML, SimSQL, TensorFlow
- **In-Database ML: MADlib, Bismarck, GLADE**

ML for Generalized Linear Models

- Model is d-dimensional vector $\vec{w}$, $d \geq 1$
- Training data $\vec{\vec{X}}$ of N d-dimensional feature vectors $\vec{x}_i$ and their corresponding label y_i, $1 \leq i \leq N$
- Objective function (or loss): $\Lambda(\vec{w}) = min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} f(\vec{w}, \vec{x}_i; y_i)$
- **Find model $\vec{w}$ that minimizes objective function based on training data**

Logistic Regression (LR)

- $\Lambda_{LR}(\vec{w}) = \sum_{i=1}^{N} \log\left(1 + e^{-y_i \vec{w} \cdot \vec{x}_i}\right)$

Low-Rank Matrix Factorization (LMF)

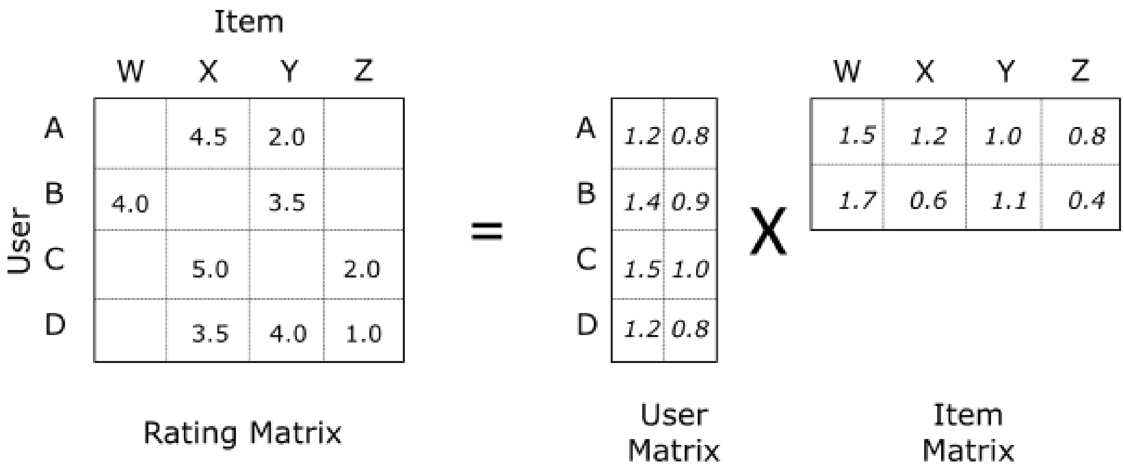
- $\Lambda_{LMF}(L, R) = \frac{1}{2} \sum_{(i,j) \in M} \left(\vec{L}_i^T \cdot \vec{R}_j - M_{ij}\right)^2$

Agenda

- **Big Model Analytics**
- Gradient Descent Optimization
- Scalable HOGWILD! for Big Models
- Experimental Results
- Conclusions

Big Model Example 1

Recommender Systems



http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify

- Spotify applies low-rank matrix factorization (LMF) to 24 million users and 20 million songs which is **4.4 billion features** at a relatively small rank of 100

Big Model Example 2

Text Analytics

Full sentence	It does not, however, control whether an exaction is within Congress's power to tax.
Unigrams	"It"; "does"; "not,"; "however,"; "control"; "whether"; "an"; "exaction"; "is"; "within"; "Congress's"; "power"; "to"; "tax."
Bigrams	"It does"; "does not,"; "not, however,"; "however, control"; "control whether"; "whether an"; "an exaction"; "exaction is"; "is within"; "within Congress's"; "Congress's power"; "power to"; "to tax."
Trigrams	"It does not"; "does not, however"; "not, however, control"; "however, control whether"; "control whether an"; "whether an exaction"; "an exaction is"; "exaction is within"; "is within Congress's"; "within Congress's power"; "Congress's power to"; "power to tax."

N-gram features of a sentence

- For the English Wikipedia corpus, a feature vector with **25 billion unigrams and 218 billion bigrams** can be constructed [S. Lee, J. K. Kim et al., "On Model Parallelization and Scheduling Strategies for Distributed Machine Learning", NIPS 2015]

Big Model Motivation

In the Cloud . . .

- There is always enough memory
- You can always add more servers

ML in IoT, Edge, and Fog Environments

- Push processing to the devices acquiring the data which have rather scarce resources
- Data transfer is not a viable alternative for bandwidth and privacy reasons
- Secondary storage (disk, SSD, or flash) is plentiful

Support for Big Models

Existing ML Systems Do Not Support Big Models

- Model is an in-memory container data structure, e.g., `vector` or `map`
- Model is array attribute in a single-column table (at most 1 GB in PostgreSQL) and in-memory state of a UDA (User-Defined Aggregate)

Parameter Server [M. Li et al., OSDI 2014]

- Partition the model across the distributed shared memory of multiple servers, with each server storing a sufficiently small model partition that fits in its local memory
- Complex partitioning, replication, and synchronization

Dot-Product Join [C. Qin and F. Rusu, SSDBM 2017]

- Serial dot-product computation between sparse matrix and massive dense vector
- Range-based model partitioning independent of training data
- Training data reordering at chunk-level

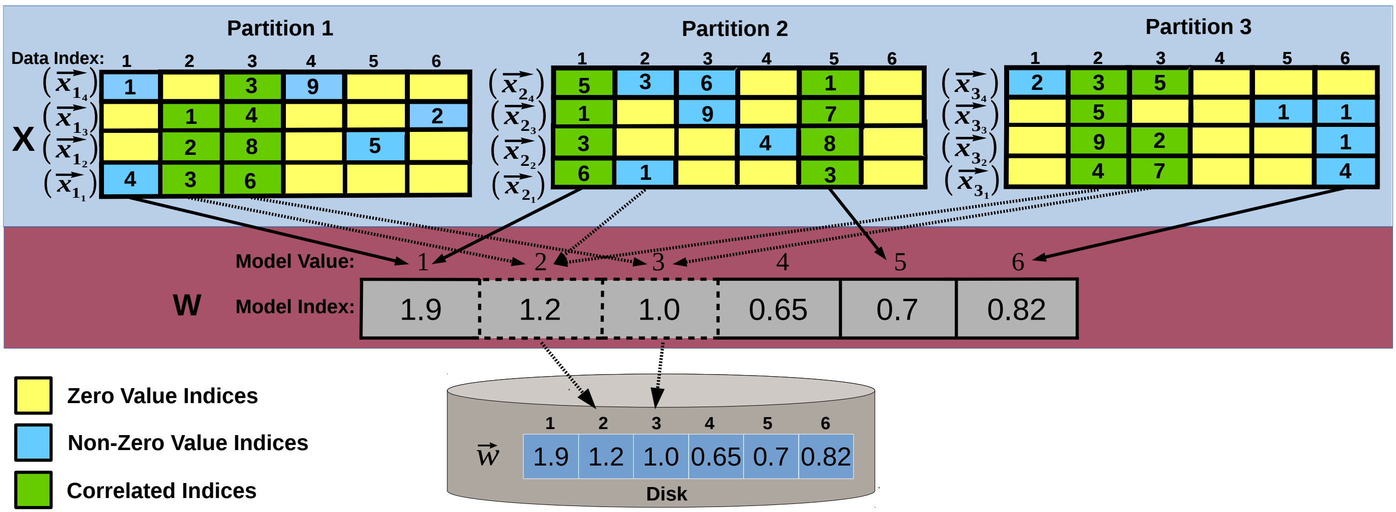
In-Database Big Model ML

Problem

- Provide scalable in-database support for Big Model ML in a single multi-core server with attached storage, i.e., disk or SSD

Challenge

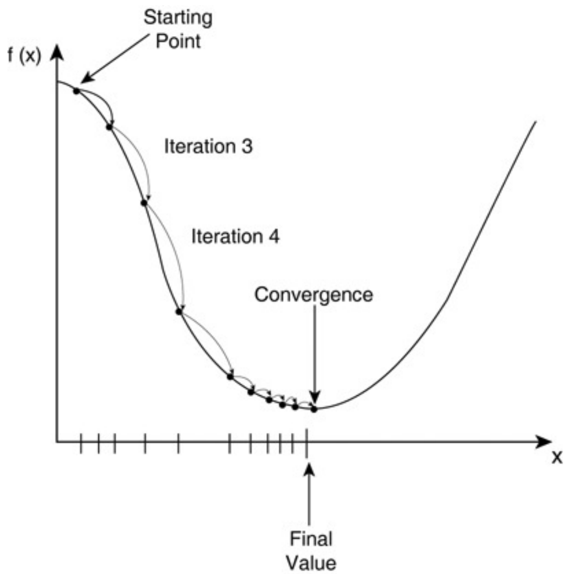
- Access to model is unpredictable and there are many accesses for each training example
- Highly-concurrent READ/WRITE model accesses across partitions (worker threads)



Agenda

- Big Model Analytics
- **Gradient Descent Optimization**
- Scalable HOGWILD! for Big Models
- Experimental Results
- Conclusions

Gradient Descent Optimization



http://www.yaldex.com/game-development/1592730043_ch18lev1sec4.html

$$min_{\vec{w}\in\mathbb{R}^d}\left\{\Lambda(\vec{w})=\sum_{i=1}^{N} f\left(\vec{w},\vec{x}_i;y_i\right)\right\}$$

$$\vec{w}^{(k+1)}=\vec{w}^{(k)}-\alpha^{(k)}\nabla\Lambda\left(\vec{w}^{(k)}\right)$$

$\alpha^{(k)}$ is step size or learning rate

$\vec{w}^{(0)}$ is the starting point (random)

$$\nabla\Lambda(\vec{w})=\left[\frac{\partial\Lambda(\vec{w})}{\partial w_1},\ldots,\frac{\partial\Lambda(\vec{w})}{\partial w_d}\right] \text{ is the gradient}$$

$$\frac{\partial\Lambda_{LR}(\vec{w})}{\partial w_i}=\sum_{i=1}^{N}\left(-y_i\frac{e^{-y_i\vec{w}\cdot\vec{x}_i}}{1+e^{-y_i\vec{w}\cdot\vec{x}_i}}\right)\vec{x}_i$$

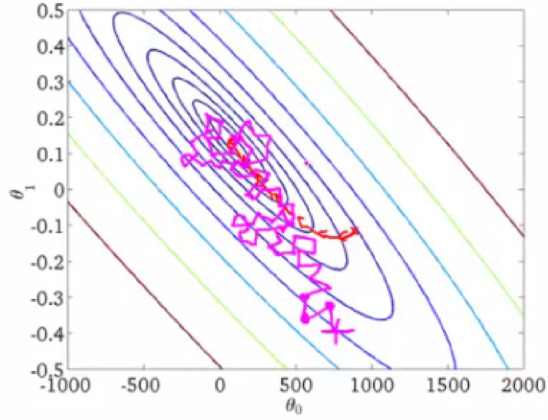
$$\frac{\partial\Lambda_{LMF}(L,R)}{\partial\vec{L}_{i'}}=\sum_{(i',j)\in M}\left(\vec{L}_{i'}^{T}\cdot\vec{R}_j-M_{i'j}\right)\vec{R}_j^{T}$$

- Convergence to minimum guaranteed for convex objective function

Stochastic Gradient Descent (SGD)

Input: $\{(\vec{x}_i, y_i)\}_{1 \le i \le N}$, f, ∇f, $\vec{w}^{(0)}$, $\alpha^{(0)}$
Output: $\vec{w}^{(k-1)}$

Let $k = 1$
while (**true**) **do**
  if convergence($\{\Lambda(\vec{w}^{(l)})\}_{0 \le l < k}$) **then break**
  for each example $(\vec{x}_{\eta^{(k)}}, y_{\eta^{(k)}})$ **do**
    Approximate gradient: $\nabla f\left(\vec{w}^{(k)}, \vec{x}_{\eta^{(k)}}; y_{\eta^{(k)}}\right)$
    $\vec{w}^{(k)} = \vec{w}^{(k)} - \alpha^{(k)} \nabla f\left(\vec{w}^{(k)}, \vec{x}_{\eta^{(k)}}; y_{\eta^{(k)}}\right)$
  end for
  Update step size $\alpha^{(k)}$
  Let $k = k + 1$
end while
return $\vec{w}^{(k-1)}$



http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

HOGWILD!: Parallel Asynchronous SGD

HOGWILD! Main Loop

for $i = 1$ **to** N **do in parallel**
  $\vec{w} \leftarrow \vec{w} - \alpha^{(k)} \nabla f(\vec{w}, \vec{x}_i; y_i)$

- Parallelize inner loop while ignoring the sequential nature of SGD [F. Niu et al., NIPS 2011]
- Single model shared across threads
- Lock-free access to model: **data races**
  - 2 READ: gradient and model update
  - 1 WRITE: model update
- Convergence is preserved for sparse data
- Hardware cache coherence limits speedup [S. Sallinen et al., IPDPS 2016]

Naive Extension to Big Models

for $i = 1$ **to** N **do in parallel**
  for each non-zero feature $j \in \{1, \ldots, d\}$ in $\vec{x}_i$ **do**
    get $\vec{w}[j]$
    compute $\nabla f(\vec{w}[j], \vec{x}_i; y_i)$
  end for
  $\vec{w} \leftarrow \vec{w} - \alpha^{(k)} \nabla f(\vec{w}, \vec{x}_i; y_i)$
  for each non-zero feature $j \in \{1, \ldots, d\}$ in $\vec{x}_i$ **do**
    put $\vec{w}[j]$
  end for
end for

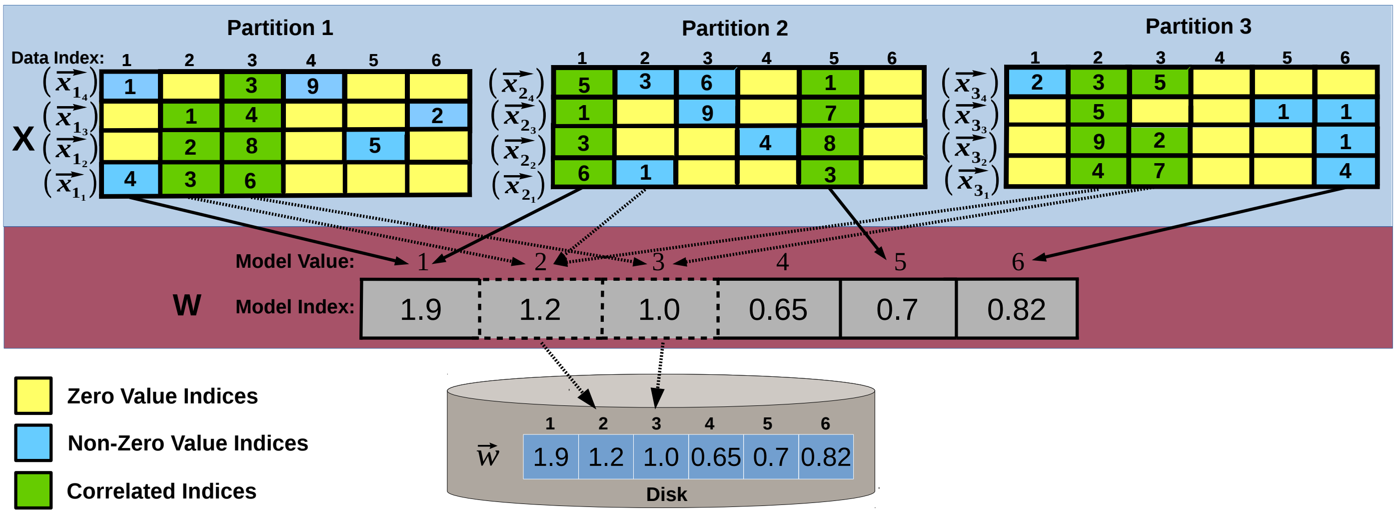
- Model is stored in disk key-value store with get/put interface, e.g., (Hyper)LevelDB

Agenda

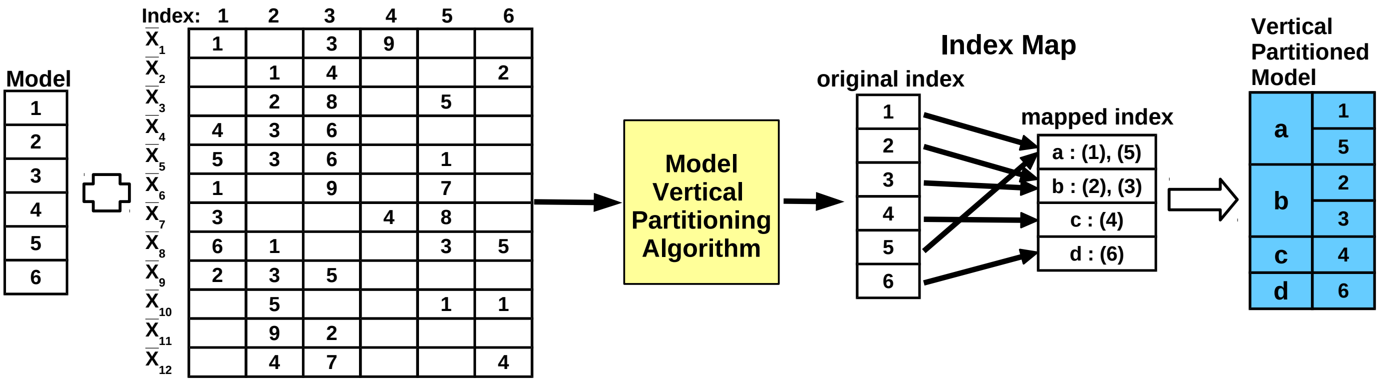
- Big Model Analytics
- Gradient Descent Optimization
- **Scalable HOGWILD! for Big Models**
- Experimental Results
- Conclusions

Scalable HOGWILD! for Big Models

- **Offline** model vertical partitioning
  - Correlate indices to reduce number of `get` calls inside an example
- **Online** asynchronous model access sharing
  - Vertical partition traversal to reduce number of `get` calls inside a partition
  - Push-based model sharing to reduce number of `get` calls across partitions
  - Partition-level model update (mini-batch) to reduce number of `put` calls

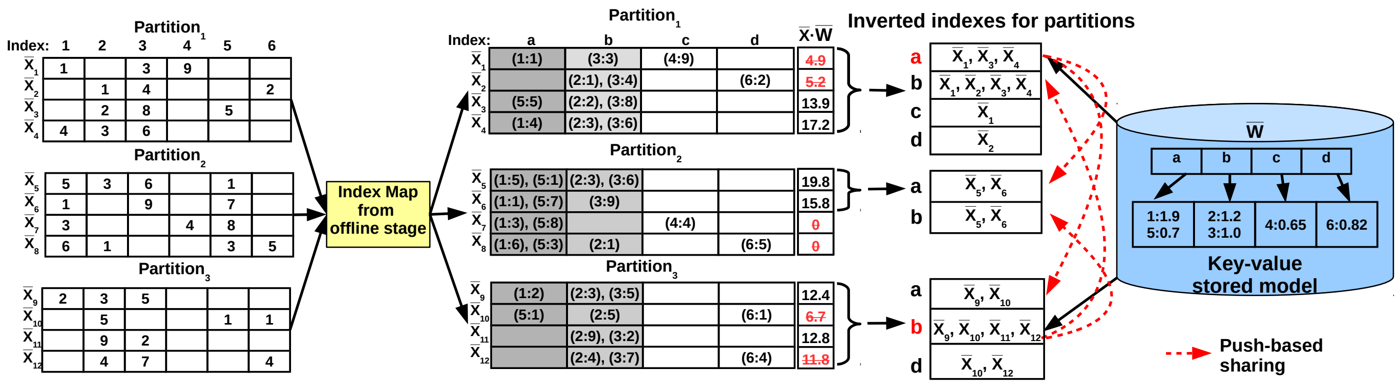


Offline Model Partitioning



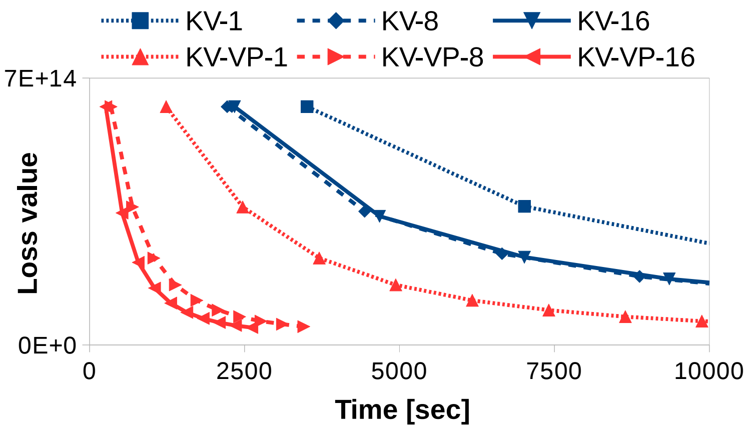
- Composite key-value storage scheme
  - Two-level values in (key, value) intermediate representation
- Model vertical partitioning algorithm
  - Vertical partitioning in physical design: examples are workload; model is relation
  - Bottom-up greedy strategy with precomputed affinity matrix
  - Cost model combines seek time and payload access time
  - Sampling & frequency-based pruning

Online Asynchronous Model Access Sharing

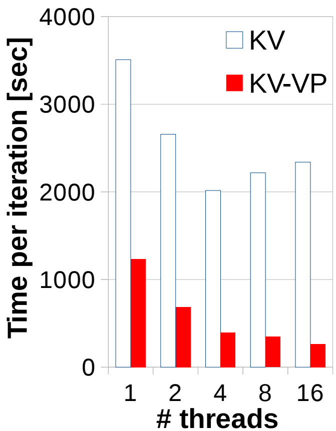


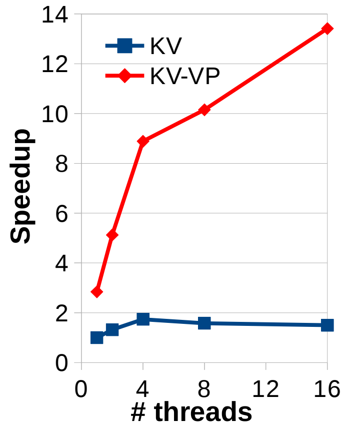
- On-the-fly example mapping into composite key-value scheme
- Vertical partition traversal
  - Incremental dot-products in gradient computation
- Push-based sharing of model indices across partitions
  - Key-to-vector inverted index (cuckoo hash table): single request across partition
- Partition-level (mini-batch) model updates preceded by get call

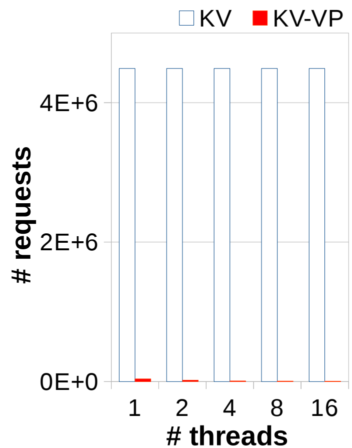
Experimental Results (1)



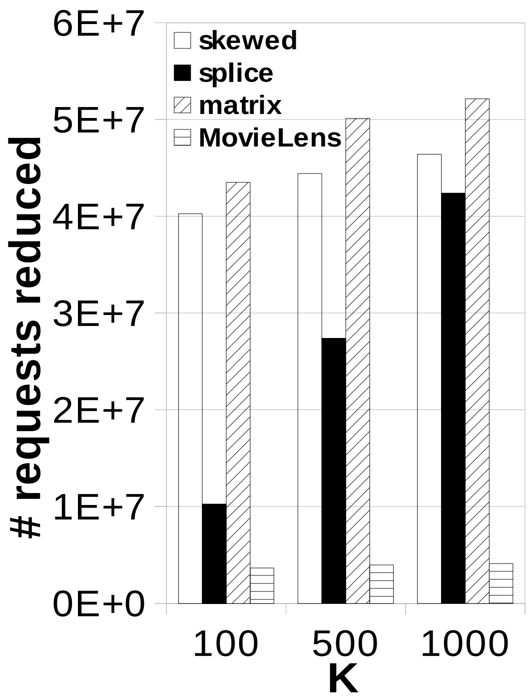
Naive (KV)
Scalable (KV-VP)

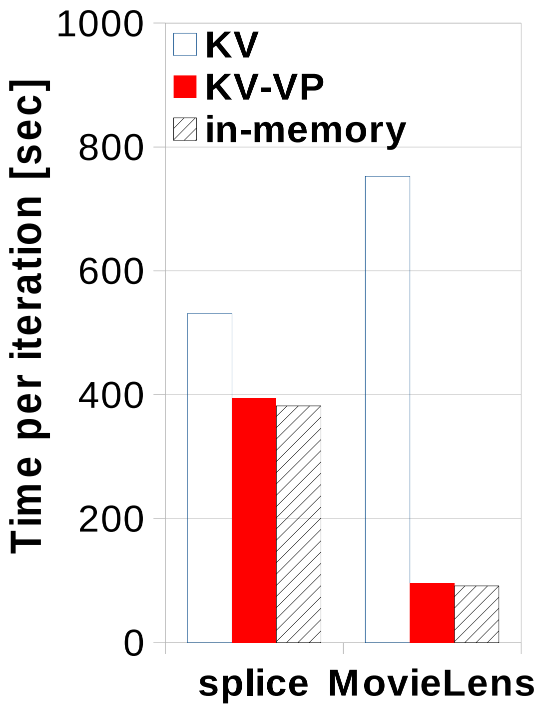






Experimental Results (2)





Conclusions

- Design a scalable model and data-parallel framework for parallelizing stochastic optimization algorithms over big models: offline model partitioning and asynchronous online training
- Formalize model partitioning as vertical partitioning and design a scalable frequency-based model vertical partitioning algorithm
- Devise an asynchronous method to traverse vertically the training examples in all the data partitions
- Design a push-based model sharing mechanism for incremental gradient computation based on partial dot-products
- Implement the entire framework using User-Defined Aggregates (UDA) which provides generality across databases
- Evaluate the framework for three analytics tasks over synthetic and real datasets
- Results prove the scalability and reduced overhead incurred by model partitioning and key-value store

Thank you.

Questions ???