STRADS-AP: Simplifying Distributed Machine Learning Programming without Introducing a New Programming Model

Jin Kyu Kim¹, Abutalib Aghayev¹, Garth A. Gibson¹,²,³, Eric P. Xing¹,⁴

¹Carnegie Mellon University, ²Vector Institute, ³University of Toronto, ⁴Petuum Inc.
Distributed ML Programming is Difficult

Distributed ML development flow

- New Model
- Sequential Algorithm
- Sequential Programs (i.e. Python, Java, R, Matlab, ...)
- Distributed Framework Programs (i.e. Spark/Hadoop/GraphLab/PS, ..)

Our goal is to simplify conversion of sequential ML programs into distributed ML programs almost mechanically.

- A Straightforward task
- Sequential thinking
- Sequential programming model

- Difficult and time consuming task
- Parallel thinking
- Non sequential programming model
Structure of Targeting Applications

Create and initialize data structures D for input data
Create and initialize data structures P for model parameters
// ... run transformations on input data or parameter if necessary
Create and initialize hyper parameters V to control training

(a) Pretraining part
for(i=0; i<max_iter; i++) { // outer loop
  for(j=0; j<N; j++) { // inner loop
    // Computations for optimization happens here
    Read a part of input data D
    Read hyper parameters V and loop indexes i, j
    Read/writes to a part of model parameters P
  }
}

change hyper parameters
if(stop condition is true)
  break;

(b) Training part

define data structures for input data and parameters

Inner loop repeats the same update routine that accesses data structures above

Computation type in inner loop body
- Asynchronous compute. i.e. sampling
- Synchronous compute. i.e. SGD

This inner loop is source of parallelism

This structure is common across a wide range of ML algorithms
Case Study: SGDMF Model/Algorithm

(1) Matrix Factorization (MF) Model

- Popular model for product recommendation
- Many optimization algorithms available
  i.e. SGD-MF, ALS-MF, CCD-MF,

Objective: \[
\min_{W \in \mathbb{R}^{m \times k}, H \in \mathbb{R}^{n \times k}} \sum_{(i,j) \in \Omega} (A_{ij} - w_i^T h_j)^2
\]

(2) Stochastic Gradient Descent (SGD) MF Algorithm

1. \( A \): a set of ratings. Each rating contains (i: user id, j: product id, r: rating)
2. \( W: M \times K \) matrix; initialize \( W \) randomly
3. \( H: N \times K \) matrix; initialize \( H \) randomly
4. for each rating \( r \) in \( A \)
5. \( \text{err} = r.r - W[r.i]H[r.j] \)
6. \( \Delta W = \gamma \cdot (\text{err} \cdot H[r.j] - \lambda \cdot W[r.i]) \)
7. \( \Delta H = \gamma \cdot (\text{err} \cdot W[r.i] - \lambda \cdot H[r.j]) \)
8. \( W[r.i] += \Delta W \)
9. \( H[r.j] += \Delta H \)

- SGDMF is asynchronous computation
  i.e. Ratings are processed sequentially, and output of processing a rating is immediately visible for the next rating processing.
Case Study: Sequential SGDMF

1: \( A \): a set of ratings. Each rating contains \( (i: \text{user id}, j: \text{product id}, r: \text{rating}) \)
2: \( W: M \times K \) matrix; initialize \( W \) randomly
3: \( H: N \times K \) matrix; initialize \( H \) randomly
4: for each rating \( r \) in \( A \)
5: \[ \text{err} = r.r - W[r.i]H[r.j] \]
6: \[ \Delta W = \gamma \cdot (\text{err} \cdot H[r.j] - \lambda \cdot W[r.i]) \]
7: \[ \Delta H = \gamma \cdot (\text{err} \cdot W[r.i] - \lambda \cdot H[r.j]) \]
8: \( W[r.i] += \Delta W \)
9: \( H[r.j] += \Delta H \)

(2) Stochastic Gradient Descent (SGD) MF Algorithm

```c
struct rate{int i, int j, float r};
typedef rate T1;
typedef array<float, K> T2;
vector<T1> A = LoadRatings(Datafile_Path);
vector<T2> W(M); RandomInit(W);
vector<T2> H(N); RandomInit(H);
float gamma(.01f), lambda(.1f);
for(int i=0; i<maxiter; i++){
  for(int j=0; j<A.size(); j++){
    const T1 &r = A[j];
    T2 err = r - W[r.i]*H[r.j];
    T2 Wd = gamma*(err*W[r.i]-lambda*H[r.j]);
    T2 Hd = gamma*(err*H[r.j]-lambda*W[r.i]);
    W[r.i] += Wd;
    H[r.j] += Hd;
  }
}
```

(3) Sequential program

- Straightforward code based on the SGD-MF algorithm
Case Study: Dist. SGDMF on STRADS-AP

(3) Sequential program

Note the similarity between these two code snippets!

(4) STRADS-AP program

reused code  modified code
Case Study: Dist. SGDMF on Spark

(3) Sequential code

This is almost a new implementation. Requires substantial time and efforts

(4) Spark-SGDMF
What increases programming complexity in Spark?

(1) Because of absence of app-level concurrency control, app code includes a scheduling code

→ App code complexity increases
→ Scheduling overhead is expensive

(2) Constraints of Spark’s functional programming

Spark constraints
- Spark RDD data structure is immutable
- parallel operator works on only a single RDD

→ W/H/A data structures should be merged via expensive join operations before parallel execution
→ Use of join operation increases code complexity and slows down performance
Case Study: SGDMF Evaluation

Application: SGDMF, Dataset: Netflix data, 100M movie ratings, 470K users, 17K movies
Cluster: 1~16 machines, each has 16 cores, Network: 20Gbps Ethernet

Training time

Compared to hand-tuned MPI: only 17% slower

OpenMP: multi core on a single machine
MPI: Message passing interface for a cloud
STRADS-Automatic Parallelization (AP)

(a) Sequential code

```c
cvector<T1> D; // input data
map<T2> P, Q; // model parameter
float alpha(0.1); // hyper parameters
for(i=0; i<max_iter; i++){
  for(j=0; j<N; j++){
    - optimization routine
    - read i, j, alpha, elements of D
    - read/write elements of P, Q
  }
  alpha *= 0.99;
}
```

(b) STRADS-AP code

```c
cvector<T1> D;
dmap<T2> P, Q;
float alpha(0.1);
for(i=0; i<N; i++){
  parallel_for(N, [i, alpha, &D, &P, &Q](int j){
    - optimization routine
    - read i, j, alpha, elements of D
    - read/write elements of P, Q
  }, ConsistencyModel);
  alpha *= 0.99;
}
```

(c) STRADS-AP compile time tool

(d) STRADS-AP runtime

(e) STRADS-AP debugging

fill the lack of C++ language’s reflection capability

Native compiler

Add Language specific augmentations

Binary code

Cluster replay

Execution log

Log execution ordering

Single node replay

Driver program

Task scheduling

Worker(s)

Worker 0

Worker 1

Worker 2

Worker 3

Client

Master

Scheduler0

Scheduler(s)

Dependency Graph

Part0

Part1

Part2

Part3

Dist. DS server0

Dist. DS server1

Dist. DS server2

DS Part0

DS Part1

DS Part2

DS Part3

Result

Schedule

10
STRADS-AP: API

STRADS-AP API consists of DDS and Loop operators

**Distributed Data Structures (DDS)**

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dvector[T]</td>
<td>Support the same API (i.e. random RW access) of sequential C++ containers</td>
</tr>
<tr>
<td>dmap[K,V]</td>
<td>→ Allows to reuse data structure and compute routine</td>
</tr>
<tr>
<td>dmultimap[K,V]</td>
<td></td>
</tr>
</tbody>
</table>

**Two Parallel Loop Operators for Synchronous and Asynchronous compute:**

<table>
<thead>
<tr>
<th>Loop Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncFor(start-id, end-id, UDF)</td>
<td>Copy the inner loop body into a C++ lambda function (UDF)</td>
</tr>
<tr>
<td>SyncFor(InputData, BatchSize, UDF, SyncOption)</td>
<td>and call a loop operator with UDF → Allows to reuse compute routine.</td>
</tr>
</tbody>
</table>
STRADS-AP System Design & Challenge

STRADS-AP Architecture

- Application code in sequential programming model
- User-defined application code

STRADS-AP API: DDS, Sync/Async loop, data processing ops

- Distributed KV store for DDSs
- Async Loop (SchMP) Engine
- Sync Loop (Data-Parallel) Engine
- Cache/Prefetch manager
- Scheduler
- Reconnaissance Iteration Executor

STRADS-AP runtime system

Major system design challenges:
(1) Long latency to access remote DDS elements
(2) Automatically finding data conflict-free parallel execution plans for Async loop operator
Major system design challenges:

(1) Long latency to access remote DDS elements
   Solution: caching/prefetching based on RE output

(2) automatically finding SchMP schedule plan for Async loop
   Solution: generate schedule plans based on RE output
STRADS-AP: Reconnaissance Execution

- RE is read-only iteration that does not change parameter values

Driver program
Parallel_For (1 to N) lambda{
  // loop body
  access DDS-A[]
  compute
  update DDS-B[]
}

worker1
loop 1 to N/2 {
  // loop body
  access DDS-A[]
  compute
  update DDS-B[]
}
Op1: rw access record on DDS
Op2: rw access record on DDS

worker2
loop N/2 to N {
  // loop body
  access DDS-A[]
  compute
  update DDS-B[]
}
OpN/2: rw access record on DDS
OpN/2+1: rw access record on DDS
OpN/2+2: rw access record on DDS
OpN: rw access record on DDS

- RE could be expensive, but it can be amortized.
- RE is executed one time, and RE outputs are reused repeatedly.
STRADS-AP Exploits Three ML Properties

For automatic parallelization, STRADS-AP exploits three characteristics of ML training routine (the loop bodies of the inner loop)

• **Serializability:** any serializable order of loop bodies is ML appropriate

• **Repetitiveness:** ML repeats the same parameter update operations over iterations

• **Steady State:** set of accessed parameters of an update operation depends on input data but does not depend on parameter values
STRADS-AP: RE Cost

- Reuse DDS access records and scheduling plan for the following iterations
  → RE and Schedule cost is well amortized over many iterations.

Application: SGD-MF, Data set: Netflix
Iteration count until convergence: 60
Overall execution time: 958 seconds
RE+Scheduling time: 114 seconds

12% overhead for Reconnaissance Execution+Scheduling
STRADS-AP: Evaluations

Benchmark Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Serial</th>
<th>OpenMP</th>
<th>MPI</th>
<th>Tartan</th>
<th>TF</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGDMF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MLR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Word2vec</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TransE</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Benchmark Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Workload</th>
<th>Feature</th>
<th>Size</th>
<th>Application</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td>100M ratings</td>
<td>489K users, 17K movies, rank=1000</td>
<td>2.2 GB</td>
<td>SGDMF</td>
<td>Recommendations</td>
</tr>
<tr>
<td>1Billion</td>
<td>1 billion words</td>
<td>Vocabulary size 308K, vector size=100</td>
<td>4.5 GB</td>
<td>Word2Vec</td>
<td>Word Embeddings</td>
</tr>
<tr>
<td>ImageNet</td>
<td>285K images</td>
<td>1K classes, 21,504 features, 24% sparsity</td>
<td>21 GB</td>
<td>MLR</td>
<td>Multi-Class Classification</td>
</tr>
<tr>
<td>FreeBase-15K</td>
<td>483K facts</td>
<td>14,951 entites, 1,345 relations, vector sz=100</td>
<td>36 MB</td>
<td>TransE</td>
<td>Graph Embeddings</td>
</tr>
</tbody>
</table>

Evaluation Methodology:
For productivity, compare line count and performs two user studies
For efficiency, measure training performance and prediction accuracy
STRADS-AP: Training Performance

Application: word2vector, dataset: 1billion dataset, 1b word tokens, vocab size: 307K, vector size=100
16 machines, each machine: 16 cores

Word2Vector Training time for 10 iterations

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Serial</th>
<th>OpenMP</th>
<th>STRADS-AP</th>
<th>MPI</th>
<th>TensorFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5400</td>
<td>6400</td>
<td>5900</td>
<td>6200</td>
<td>6000</td>
</tr>
<tr>
<td>16</td>
<td>3300</td>
<td>4000</td>
<td>3700</td>
<td>4100</td>
<td>3500</td>
</tr>
<tr>
<td>128</td>
<td>2500</td>
<td>3100</td>
<td>2900</td>
<td>3300</td>
<td>3000</td>
</tr>
<tr>
<td>256</td>
<td>2100</td>
<td>2800</td>
<td>2600</td>
<td>2900</td>
<td>2600</td>
</tr>
</tbody>
</table>

Max: 115,100s
Max: 22,160s

Less than 7% slower than MPI (hand-tuned) code

Development efforts for each

<table>
<thead>
<tr>
<th>Code</th>
<th>Effort</th>
<th>Lines in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRADS-AP</td>
<td>2 hours</td>
<td>402</td>
</tr>
<tr>
<td>MPI</td>
<td>2 weeks</td>
<td>559</td>
</tr>
<tr>
<td>TF</td>
<td>Complicated</td>
<td>364 Lines in C++, 282 in Python</td>
</tr>
</tbody>
</table>
STRADS-AP: Test Accuracy

Word2Vector similarity test accuracy & analogy test accuracy report after 10 iterations

<table>
<thead>
<tr>
<th>Cores</th>
<th>Similarity</th>
<th></th>
<th></th>
<th>Analogy</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STRADS-AP</td>
<td>MPI</td>
<td>TF</td>
<td>STRADS-AP</td>
<td>MPI</td>
<td>TF</td>
</tr>
<tr>
<td>128</td>
<td>0.601</td>
<td>0.601</td>
<td>0.602</td>
<td>0.566</td>
<td>0.564</td>
<td>0.568</td>
</tr>
<tr>
<td>256</td>
<td>0.603</td>
<td>0.597</td>
<td>0.601</td>
<td>0.562</td>
<td>0.557</td>
<td>0.561</td>
</tr>
<tr>
<td>Serial</td>
<td>0.610</td>
<td></td>
<td></td>
<td>ideal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP</td>
<td>0.608</td>
<td></td>
<td></td>
<td>ideal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

→ Run 10 iterations for all configurations
STRADS-AP’s test accuracy is comparable to ideal accuracy of serial code
STRADS-AP: Conclusion

• STRADS-AP simplifies distributed programming for general purpose ML algorithms while achieving performance comparable to hand-tuned MPI programs
Acknowledgement

Garth A. Gibson  Eric P. Xing

Abutalib Aghayev  Aaron Harlap  Cui Hengang  Xun Zheng  SeungH. Lee  Qirong Ho  Aurick Qiao  Jinlaing Wei
Thank you!