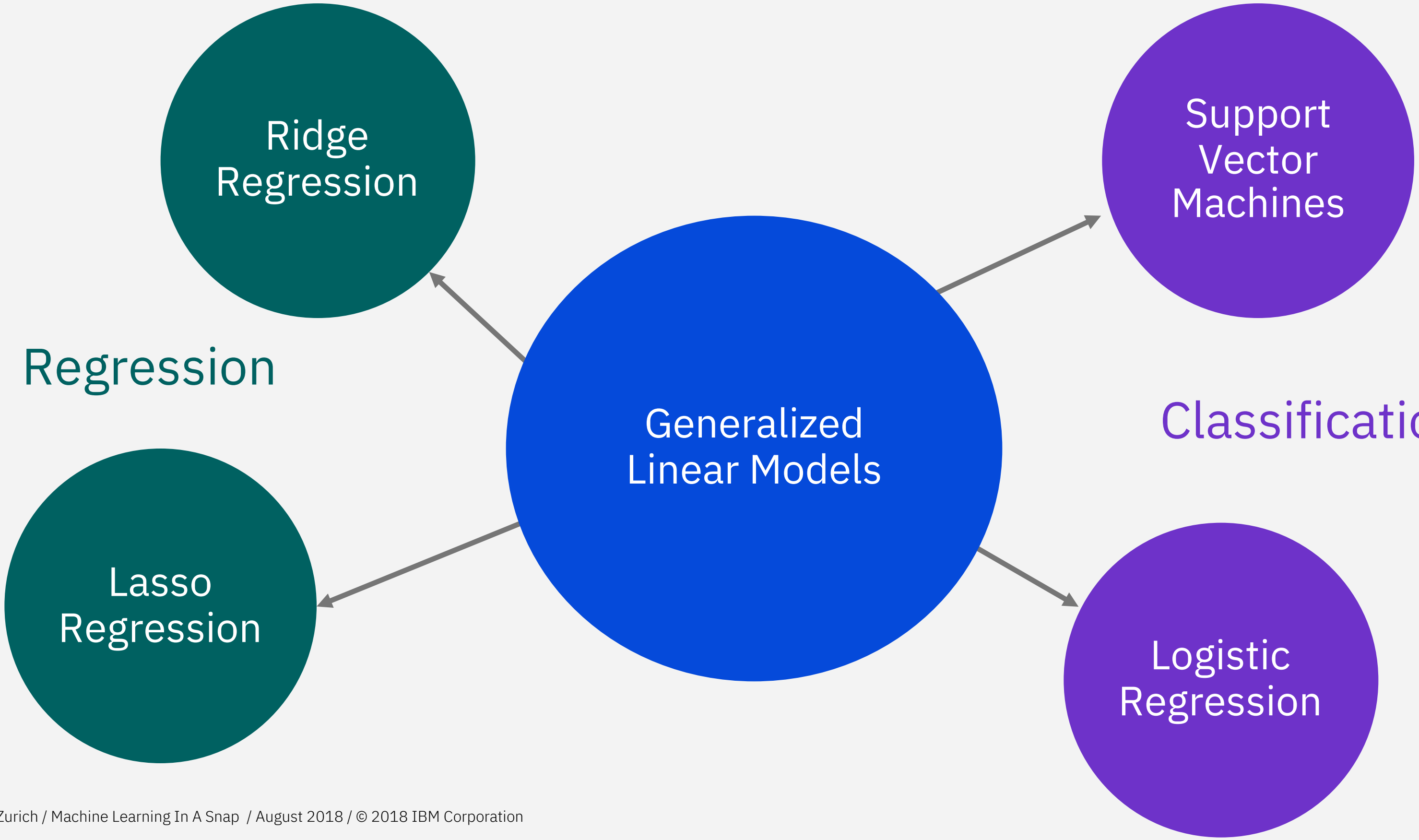
Machine Learning In A Snap

—

Thomas Parnell
Research Staff Member
IBM Research - Zurich

What are GLMs?



Why are GLMs useful?

Fast Training

Can scale to datasets with billions of examples and/or features.

Less tuning

State-of-the-art algorithms for training linear models do not involve a step-size parameter.

Interpretability

New data protection regulations in Europe (GDPR) give E.U. citizens the right to “obtain an explanation of a decision reached” by an algorithm.

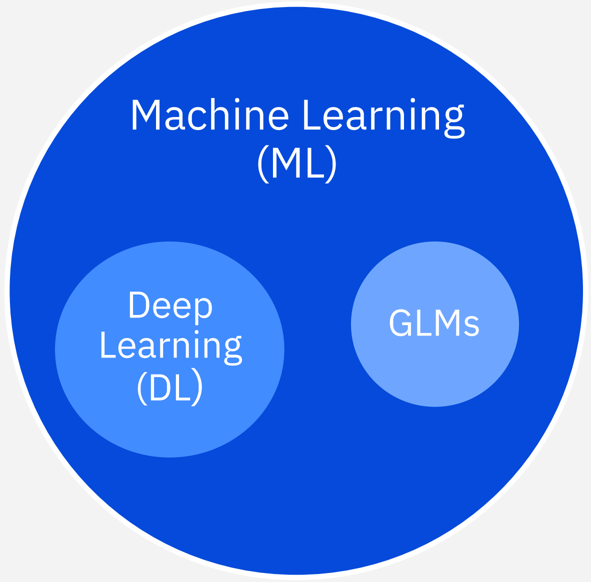
Widely used in industry

The Kaggle ”State of Data Science” survey asked 16,000 data scientists and ML practitioners what tools and algorithms they use on a daily basis.

37.6% of respondents use Neural Networks

63.5% of respondents use Logistic Regression

What is Snap Machine Learning?

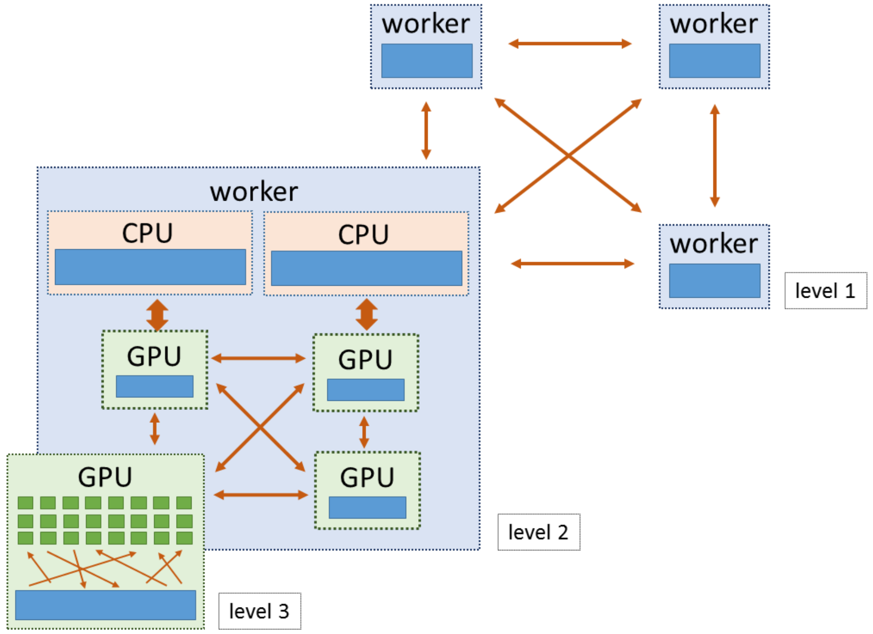


Snap ML: A new framework for fast training of GLMs

Framework	Models	GPU Acceleration	Distributed Training	Sparse Data Support
scikit-learn	ML/{DL}	No	No	Yes
Apache Spark* MLlib	ML/{DL}	No	Yes	Yes
TensorFlow**	ML	Yes	Yes	Limited
Snap ML	GLMs	Yes	Yes	Yes

* The Apache Software Foundation (ASF) owns all Apache-related trademarks, service marks, and graphic logos on behalf of our Apache project communities, and the names of all Apache projects are trademarks of the ASF.
**TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.

Multi-level Parallelism



Level 1

Parallelism across nodes connected via a network interface.

Level 2

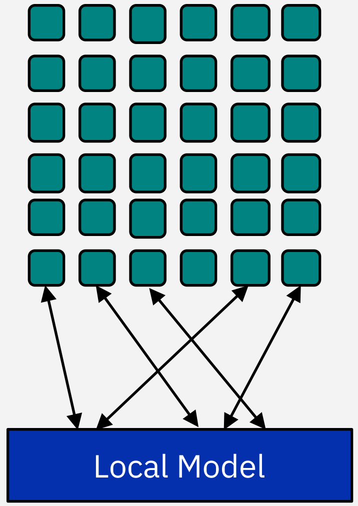
Parallelism across GPUs within the same node connected via an interconnect (e.g. NVLINK).

Level 3

Parallelism across the streaming multiprocessors of the GPU hardware.

GPU Acceleration

T. Parnell, C. Dünner, K. Atasu, M. Sifalakis and H. Pozidis,
”*Tera-scale coordinate descent on GPUs*”,
Future Generation Computer Systems, 2018



GLMs can be effectively trained using stochastic coordinate descent (SCD).

[Shalev-Schwartz 2013]

Recently, asynchronous variants of SCD have been proposed to run on multi-core CPUs.

[Liu 2013]

[Tran 2015]

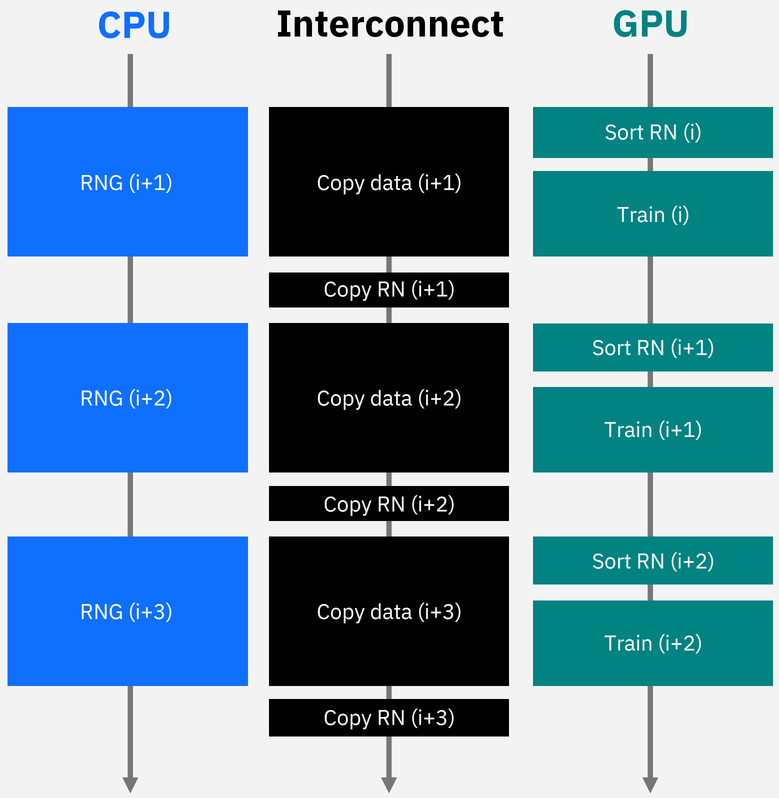
[Hsiesh 2015]

Twice parallel asynchronous SCD (TPA-SCD) is a another recent variant designed to run on GPUs.

It assigns each coordinate update to a different block of threads that executed asynchronously.

Within each thread block, the coordinate update is computed using many tightly coupled threads.

Streaming Pipeline



When the dataset is too big to fit into aggregate GPU memory, we need to copy the training data onto the GPU in batches.

The time required to copy the next batch of data on the GPU can become a bottleneck.

In Snap ML we use CUDA streams to implement a streaming pipeline.

We copy the data for the next batch while the GPU is training on the current batch.

With high-speed interconnects such as NVLINK, this allows us to completely hide the copy time behind the training time.

libglm	snap-ml-local	snap-ml-mpi	snap-ml-spark
Underlying C++/CUDA template library.	Small to medium-scale data.	Large-scale data.	Large-scale data
	Single node deployment.	Multi-node deployment in HPC environments.	Multi-node deployment in Apache Spark environments.
	Multi-GPU support.	Many-GPU support.	Many GPU support
	scikit-learn compatible Python API	Python API.	Python/Scala /Java* API.

*Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Example: snap-ml-local

```python
# Load data
from sklearn.datasets import load_from_svmlight_format
X, y_ = load_from_svmlight_format(filename_train)

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Create the logistic regression
if(use_snap_ml):
    from snap_ml import LogisticRegression
    lr = LogisticRegression(device_ids=[0,1])
else:
    from sklearn.linear_model import LogisticRegression
    lr = LogisticRegression()

# Training
lr.fit(X_train, y_train)

# Inference
proba_test = lr.predict_proba(X_test)

# Evaluate logarithmic loss on test set
from sklearn.metrics import log_loss
test_loss = log_loss(y_test, proba_test)
```

Acceleration existing scikit-learn applications by changing only 2 lines of code.

Example: snap-ml-mpi

Describe application using high-level Python code.

```python
# Load data
from snap_ml_mpi.Loaders import load_from_snap_format
train_data = load_from_libsvm_format(train_filename)
test_data  = load_from_libsvm_format(test_filename)

# Create the logistic regression
from snap_ml_mpi import LogisticRegression
lr = LogisticRegression(max_iter=200, dual=True, device_ids=[0,1,2,3], num_threads=128)

# Training
lr.fit(train_data)

# Inference
proba_test = lr.predict_proba(test_data)

# Evaluate logarithmic loss on test set
from snap_ml_mpi.Metrics import log_loss
test_loss = log_loss(y_test, proba_test)
```

Launch application on 4 nodes using mpirun (4 GPUs per node):

```
$ mpirun -n 4 -rf myrankfile python my_app.py
```

Click-through Rate (CTR) Prediction

Can use train ML models to predict whether or not a user will click on an advert?

Core business of many internet giants.

Labelled training examples are being generated in real-time.

Dataset: criteo-kaggle

Number of features: 1 million

Number of examples: 45 million

Size: 40 Gigabytes (SVM Light)

Dataset: criteo-tb

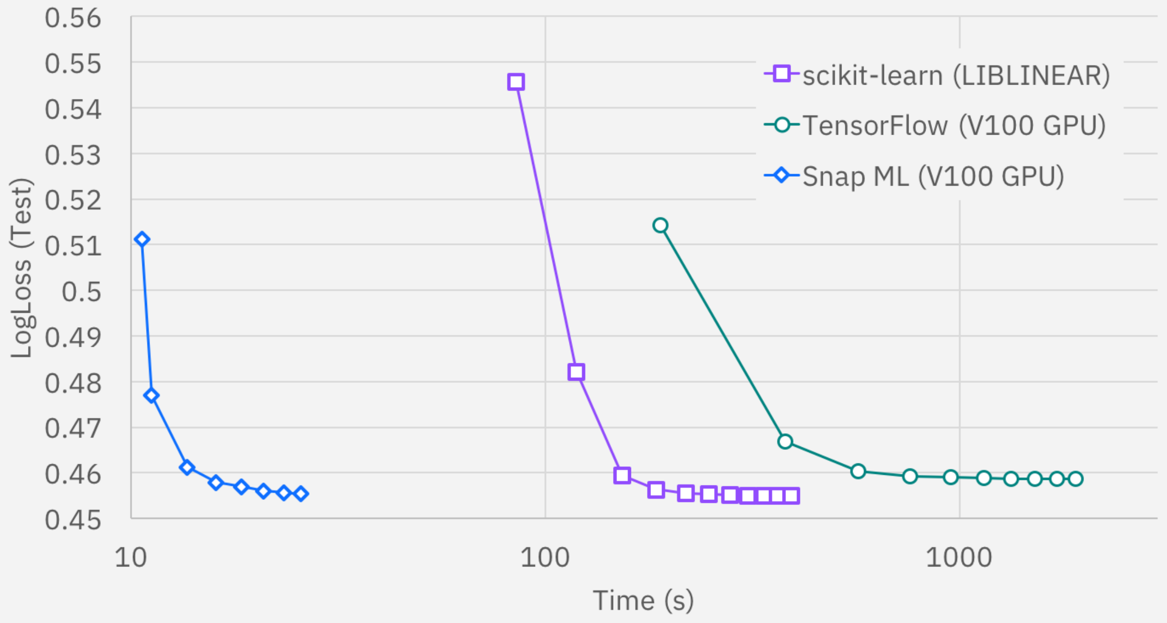
Number of features: 1 million

Number of examples: 4.2 billion

Size: 3 Terabytes (SVM Light)

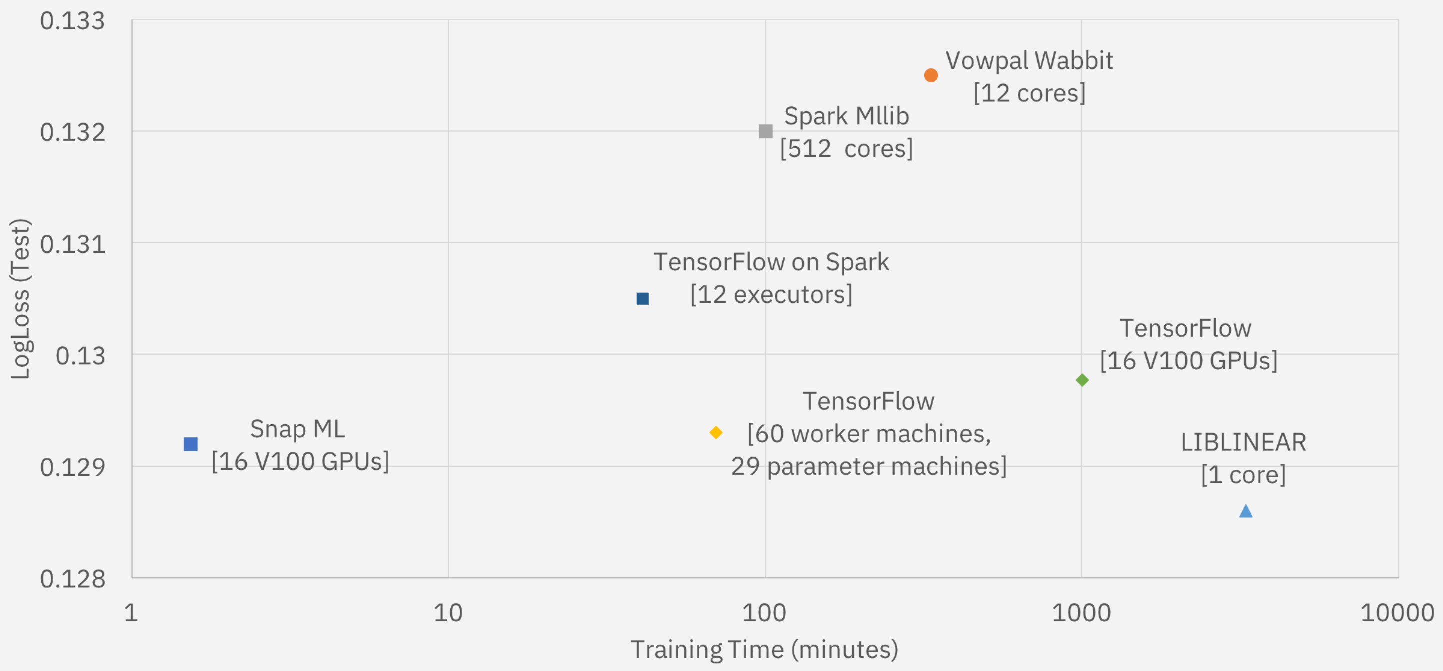
Single-node Performance

Dataset	Examples	Nodes	Total GPUs	Network	CPU-GPU interface
criteo-kaggle	45 million	1x Power AC922 server	1x V100	n/a	NVLINK 2.0



Terabyte-scale Benchmark

Dataset	Examples	Nodes	Total GPUs	Network	CPU-GPU interface
criteo-tb	4.2 billion	4x Power AC922 server	16x V100	InfiniBand	NVLINK 2.0



Conclusions

The Snap ML team:

Celestine Dünner

Dimitrios Sarigiannis

Andreea Anghel

Nikolas Ioannou

Haris Pozidis

Thomas Parnell

https://www.zurich.ibm.com/snapml/

Snap ML is a new framework for fast training of GLMs.

Snap ML benefits from GPU acceleration.

It can be deployed in single-node and multi-node environments.

The hierarchical structure of the framework makes it suitable for cloud-based deployments.

It can leverage fast interconnects like NVLINK to achieve streaming, out-of-core performance.

Snap ML significantly outperforms other software frameworks for training GLMS in both single-node and multi-node benchmarks.

Snap ML can train a logistic regression classifier on the Criteo Terabyte Click Logs data in 1.5 minutes.

Snap ML is available as part of IBM PowerAI (v5.2 onwards)

Example: snap-ml-spark

Describe application using high-level Python code:

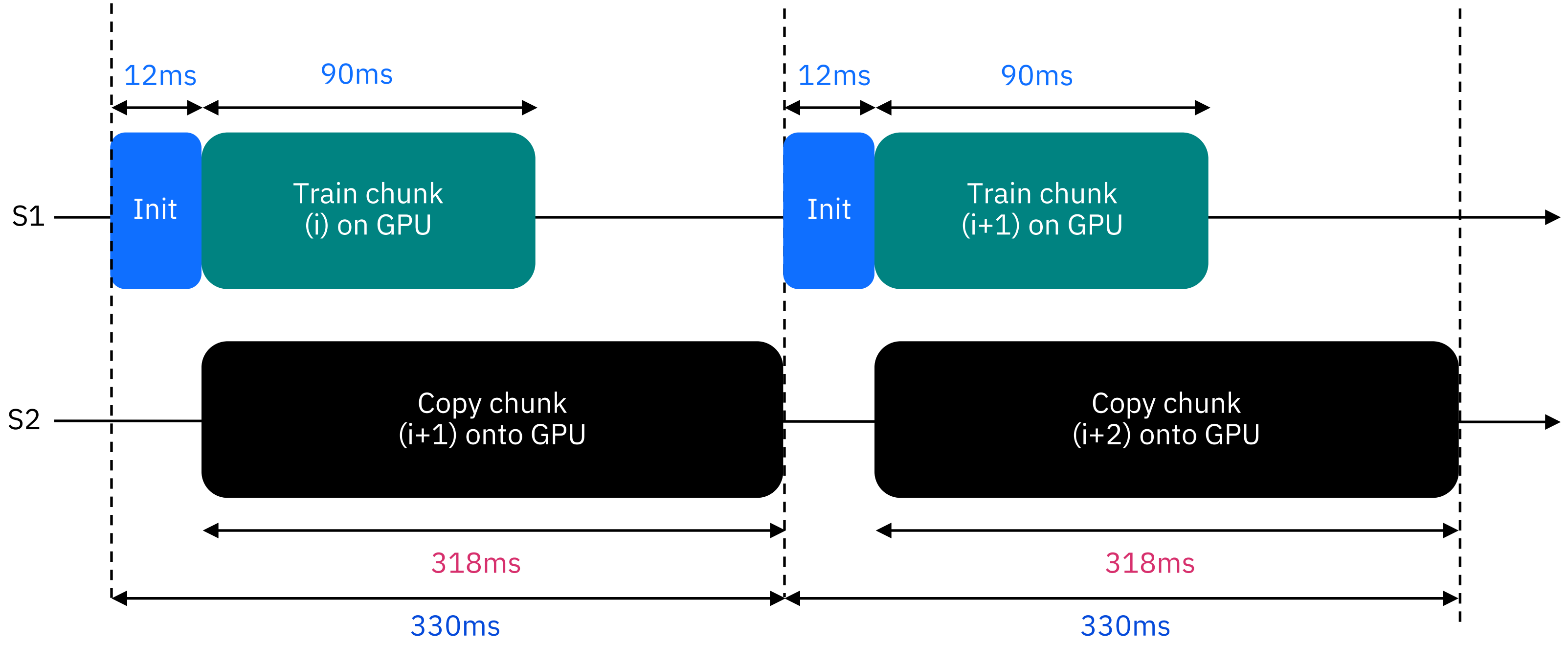
```python
# Load data
from snap_ml_spark import DatasetReader
train_data = DatasetReader().setFormat("libsvm").load(train_filename)
test_data = DatasetReader().setFormat("libsvm").load(test_filename)

# Create the logistic regression
from snap_ml_spark import LogisticRegression
lr = LogisticRegression(max_iter=40, regularizer=1.0, use_gpu=True)

# Training
lr.fit(train_data)

# Inference
test_data_with_pred = lr.predict_proba(test_data)

# Evaluate accuracy on the test set
from snap_ml_spark.Metrics import accuracy
accc = accuracy(test_data_with_pred)
```

Launch application on Spark cluster (1 GPU per Spark executor):

```
$ spark-submit --jars snap-ml-spark-v1-ppc64le.jar my_app.py
```

Out-of-core performance (PCIe Gen3)

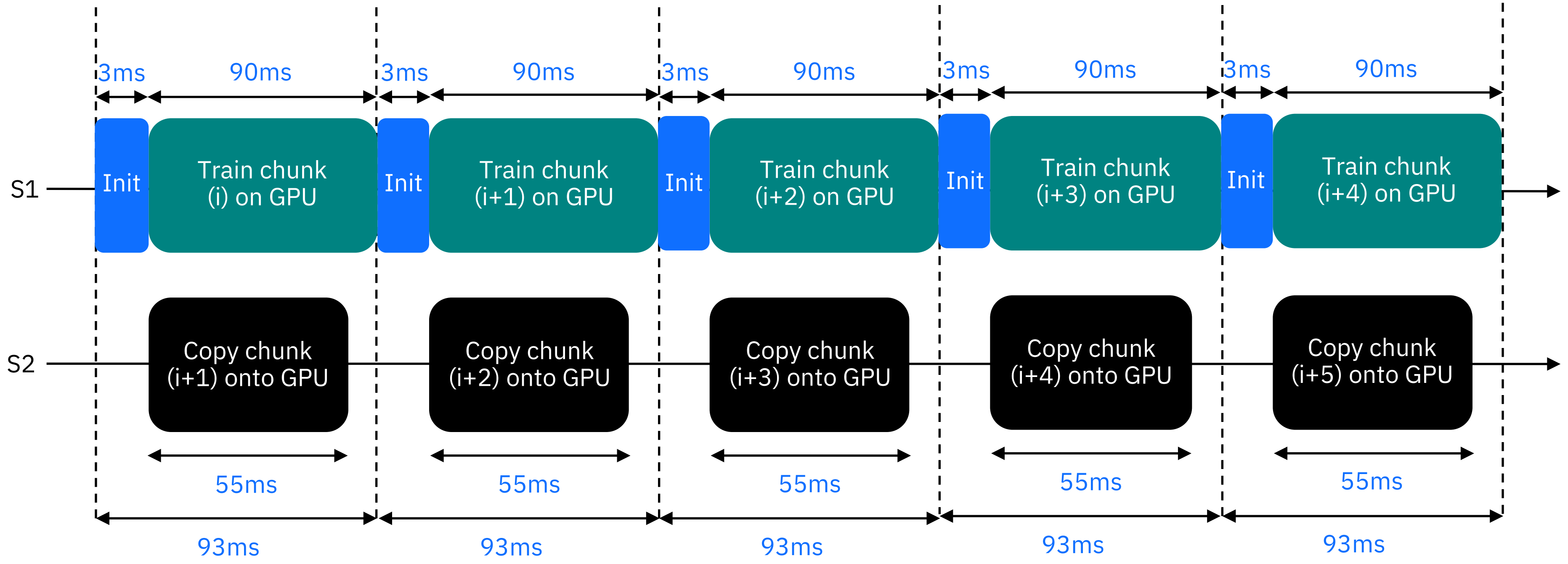
Dataset	Examples	Nodes	Total GPUs	Network	CPU-GPU interface
criteo-tb	200 million	1x Intel Xeon* Gold 6150	1x V100	n/a	PCI Gen3



*Intel Xeon is a trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Out-of-core performance (NVLINK 2.0)

Dataset	Examples	Nodes	Total GPUs	Network	CPU-GPU interface
criteo-tb	200 million	1x Power AC922 server	1x V100	n/a	NVLINK 2.0


3ms
90ms
3ms
90ms
3ms
90ms
3ms
90ms
3ms
90ms
S1
Init
Train chunk (i) on GPU
Init
Train chunk (i+1) on GPU
Init
Train chunk (i+2) on GPU
Init
Train chunk (i+3) on GPU
Init
Train chunk (i+4) on GPU
S2
Copy chunk (i+1) onto GPU
Copy chunk (i+2) onto GPU
Copy chunk (i+3) onto GPU
Copy chunk (i+4) onto GPU
Copy chunk (i+5) onto GPU
55ms
55ms
55ms
55ms
55ms
93ms
93ms
93ms
93ms
93ms