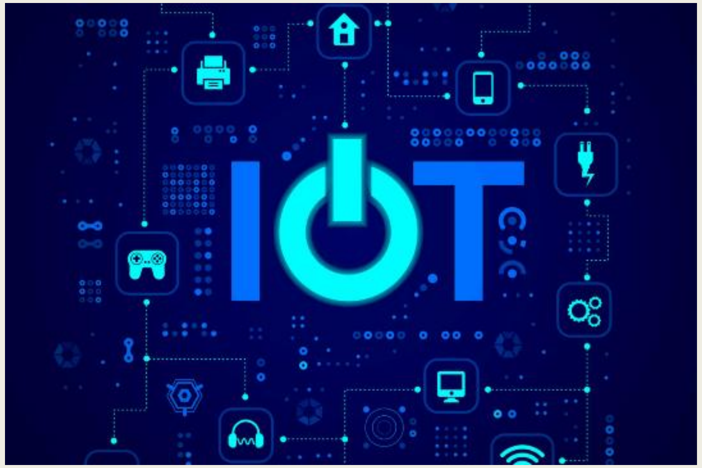
Scalable Distributed Machine Learning with Huge Data for IoT and Scientific Discovery: Opportunities and Challenges

Tony Luo Sajal Das
Department of Computer Science
Missouri University of Science and Technology
14 April 2020

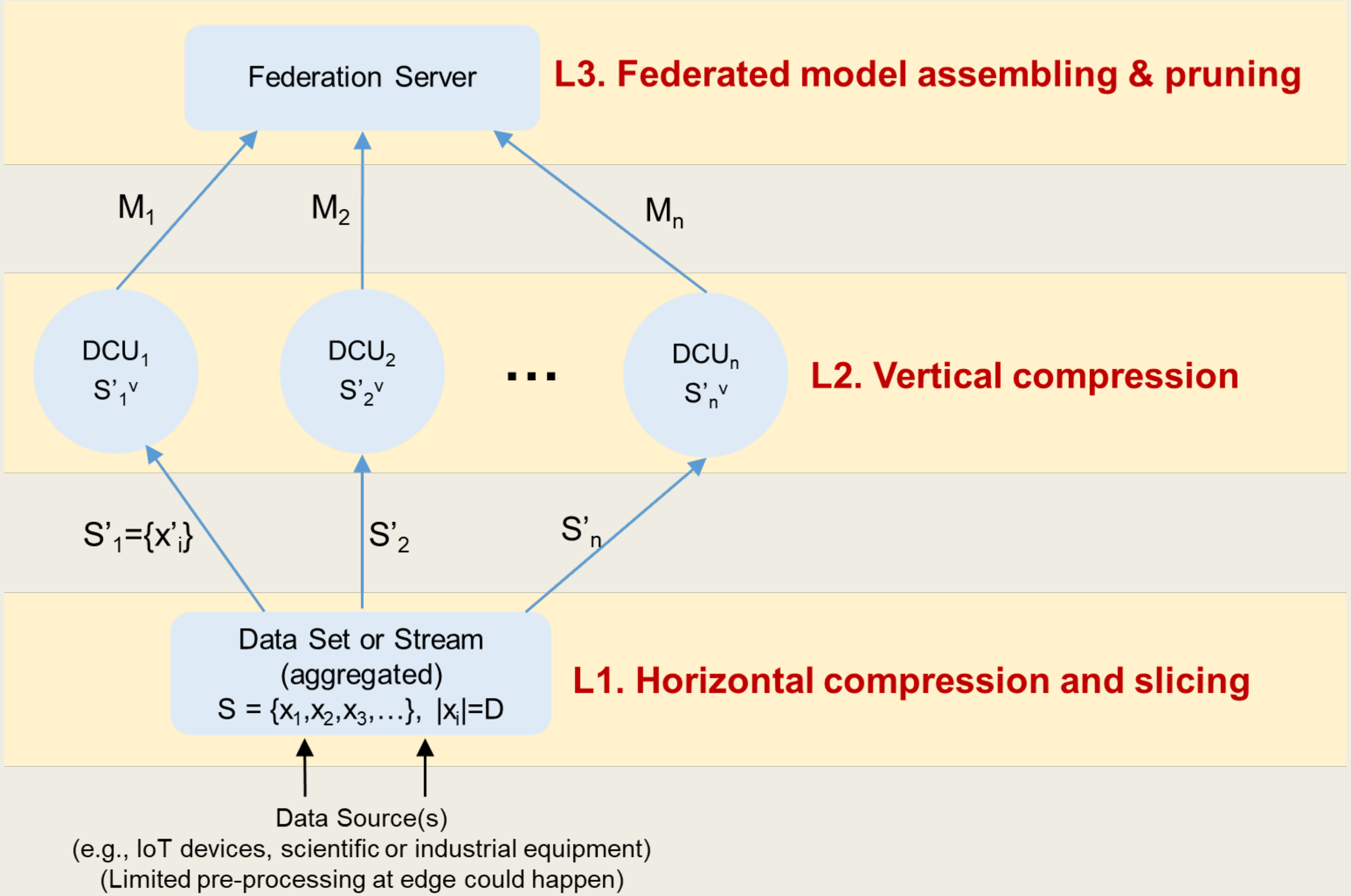
Motivation and Objective

- “**Huge Data**” problem: Data generation rates in petabyte, exabyte, and even zettabyte per hour are becoming increasingly common
  - Scientific fields such as astronomy, physics, and biological sciences
  - Internet of Things (IoT) with billions of sensors and devices interconnected



- Our goal: solve the “Huge Data” problem in the context of machine learning such that models can be efficiently trained on a huge amount of data and can make fast predictions on new data samples.

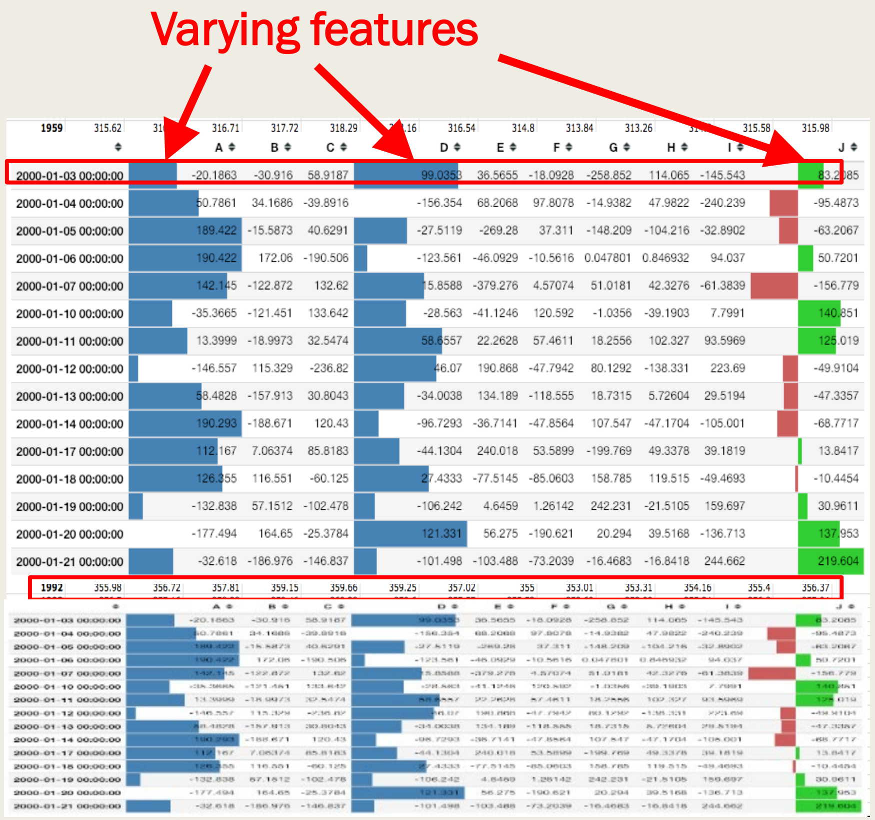
Scalable distributed ML framework



- Original data set or stream
  - S = {x1, x2, x3,…}
- Huge in two dimensions:
  - **Vertically huge:** a huge number of data samples (i.e., rows) or the influx streaming rate is extremely high
  - **Horizontally huge:** a huge number of features (i.e., columns), D

L1. Horizontal compression and slicing

- **<u>Horizontal compression</u>**: reduce the number of rows
- Key observation: consecutive data samples often have little variation except for a few features, whose variation can be characterized by a well-defined function or probability distribution
- Compression method: only keep a "seed sample" + descriptive functions; all the subsequent samples are drastically condensed into a single number, until another "seed sample" appears, which indicates a change that the descriptive functions cannot describe.
  - Original S → Seed sample set S'={x'1, x'2, x'3,…} + metadata set Q = {{F1, k1}, {F2, k2}, {F3, k3},…}, k_j is the number of data samples described by tuple (x'_j, F_j)
  - Compression ratio: $r = 1 - m/\sum k_j$ where m=|S'| is the number of signal samples
  - *r* → 1 in many practical cases!

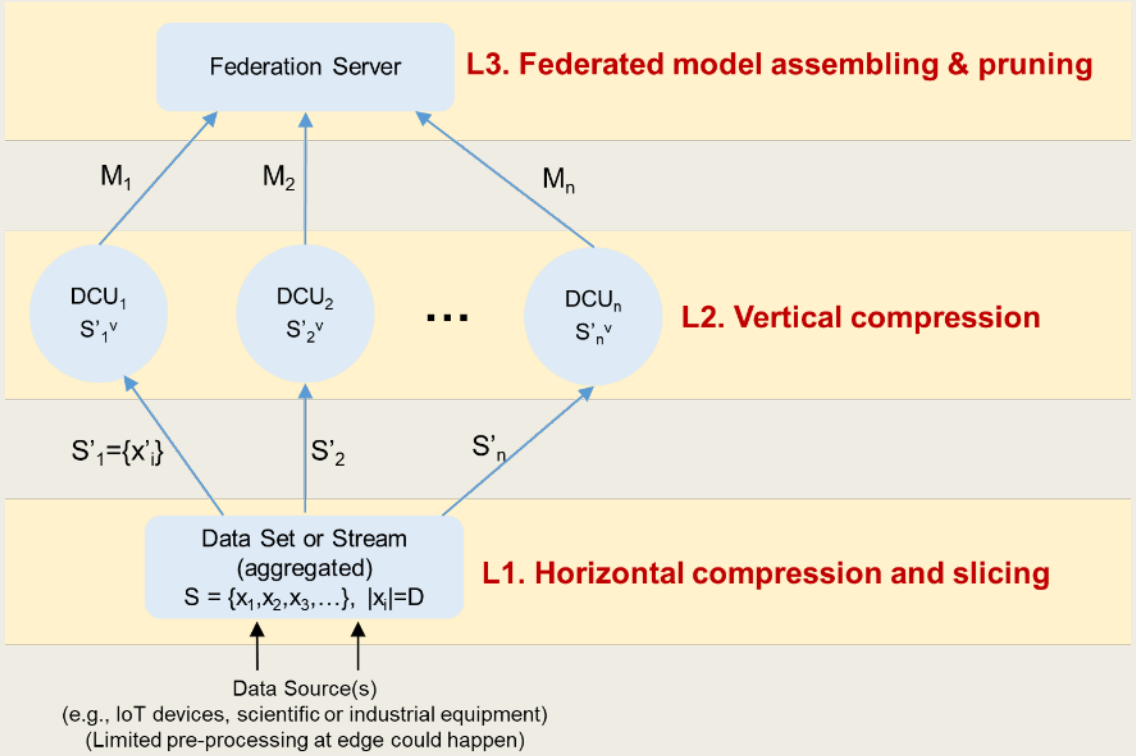


	A	B	C	D	E	F	G	H	I	J
2000-01-03 00:00:00	-20.1863	-30.916	58.9187	99.0353	36.5655	-18.0928	-258.852	114.065	-145.543	83.2085
2000-01-04 00:00:00	50.7861	34.1686	-39.8916	-156.354	68.2068	97.8078	-14.9382	47.9822	-240.239	-95.4873
2000-01-05 00:00:00	189.422	-15.5873	40.6291	-27.5119	-269.28	37.311	-148.209	-104.216	-32.8902	-63.2067
2000-01-06 00:00:00	190.422	172.06	-190.506	-123.561	-46.0929	-10.5616	0.047801	0.846932	94.037	50.7201
2000-01-07 00:00:00	142.145	-122.872	132.62	15.8588	-379.276	4.57074	51.0181	42.3276	-61.3839	-156.779
2000-01-10 00:00:00	-35.3665	-121.451	133.642	-28.563	-41.1246	120.592	-1.0356	-39.1903	7.7991	140.851
2000-01-11 00:00:00	13.3999	-18.9973	32.5474	58.6557	22.2628	57.4611	18.2556	102.327	93.5969	125.019
2000-01-12 00:00:00	-146.557	115.329	-236.82	46.07	190.868	-47.7942	80.1292	-138.331	223.69	-49.9104
2000-01-13 00:00:00	58.4828	-157.913	30.8043	-34.0038	134.189	-118.555	18.7315	5.72604	29.5194	-47.3357
2000-01-14 00:00:00	190.293	-188.671	120.43	-96.7293	-36.7141	-47.8564	107.547	-47.1704	-105.001	-68.7717
2000-01-17 00:00:00	112.167	7.06374	85.8183	-44.1304	240.018	53.5899	-199.769	49.3378	39.1819	13.8417
2000-01-18 00:00:00	126.355	116.551	-60.125	27.4333	-77.5145	-85.0603	158.785	119.515	-49.4693	-10.4454
2000-01-19 00:00:00	-132.838	57.1512	-102.478	-106.242	4.6459	1.26142	242.231	-21.5105	159.697	30.9611
2000-01-20 00:00:00	-177.494	164.65	-25.3784	121.331	56.275	-190.621	20.294	39.5168	-136.713	137.953
2000-01-21 00:00:00	-32.618	-186.976	-146.837	-101.498	-103.488	-73.2039	-16.4683	-16.8418	244.662	219.604

- <u>Horizontal slicing</u>: slice the compressed data S' horizontally, into S'_1, S'_2,…, S'n while keeping intact the width of each data sample D. The subsets S'_1, S'_2,…, S'n are then sent to *n* distributed computing units (DCUs)
- When S is a data stream rather than a static dataset, the slicing procedure is substituted by a dispatching mechanism

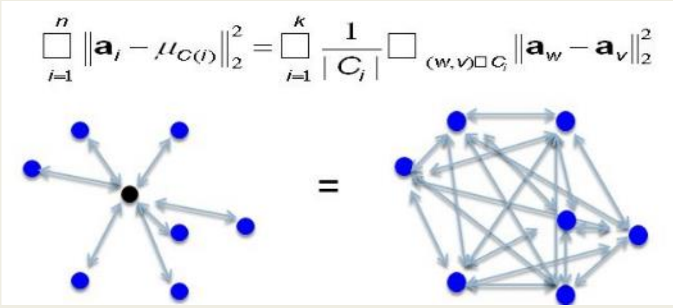
Does not wait for each subset S'i to be complete but immediately sends it to a DCU once the size of S'i reaches a certain value (can be as small as one), and this repeats.

Choice of DCU: round-robin or adaptively based on bandwith and workload.



L2. Vertical compression

- At each DCU: reduce the dimension of each sample from D to d (d<<D)
- Classic dimensionality reduction technique: principal component analysis (PCA)
  - Deterministic: always result in the same subset of reduced features
  - Not desired by some ML tasks such as ensemble learning, which needs model diversity to boost performance
- We propose random Johnson-Lindenstrauss projection as an alternative
  - Introduces randomness while preserving pairwise distances



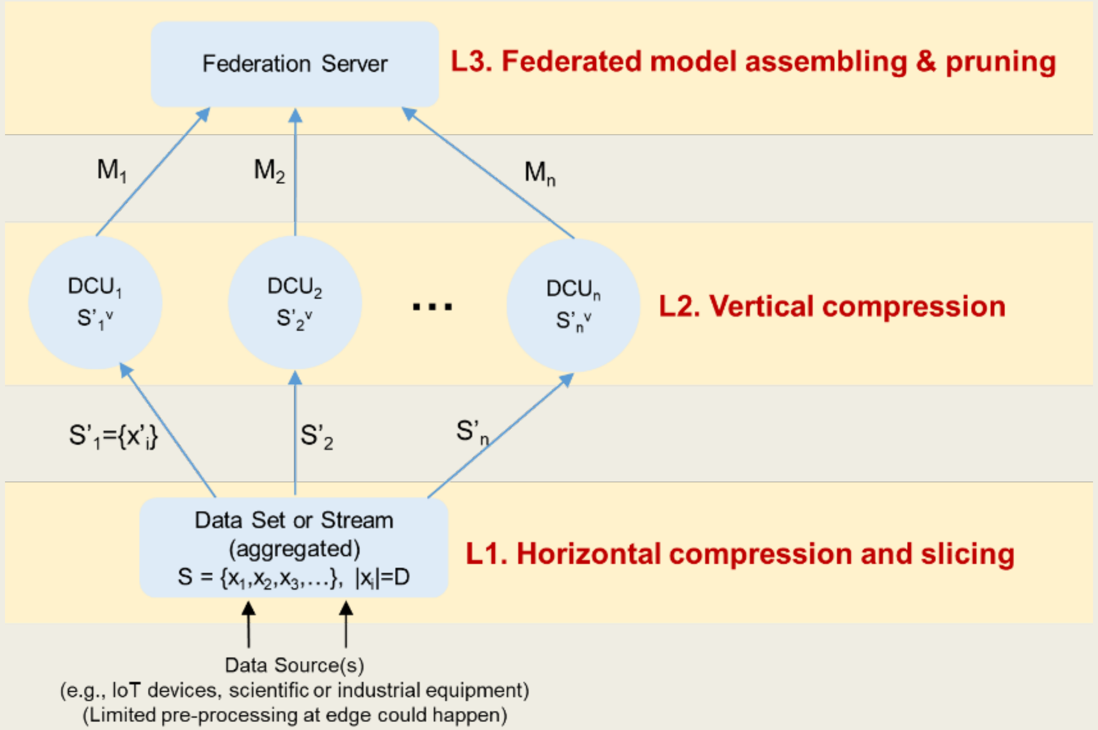
- After vertically compressing S'_i into S'^{v}_i, a sub-model M_i is trained over S'^{v}_i on the same DCU
  - Training process needs to be tailored to accommodate the metadata introduced by L1, for which we anticipate a “virtual expansion” procedure

L3. Federated model assembling & pruning

- **Assembling**: aggregate the n sub-models M_i into one global model M
- **Challenge**: sub-models are heterogeneous
  - Horizontal slicing (L1) may result in non-i.i.d. datasets
  - Vertical compression (L2) may result in different features
  - Conventional distributed machine learning not applicable
  - Federated averaging can handle non-iid data but not feature heterogeneity
  - A novel method is needed: Open challenge
- **Pruning**: global model M may still contain a large number of parameters due to the huge data setting, and thus may need a pruning process before deployment in order to achieve fast runtime predictions
  - Starting point: exist pruning and quantization techniques for deep neural networks (DNN)

Conclusion

- Proposed a framework and preliminary ideas to advance the state-of-the-art in scalable distributed machine learning for huge data analytics
- Our approach could alleviate the hurdles of storing, transferring, and processing huge data such that scientific research and IoT systems at huge-data scale could substantially benefit from AI



Tony Luo:
tluo@mst.edu
https://tluocs.github.io