Summary of Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds

Problem and Motivation

Machine learning is widely used across an extensive range of domains to learn and infer useful information from large amounts of data. Large companies such as Google, Microsoft, and Amazon deploy large-scale applications running in numerous data centers scattered across the globe, enabling them to host massive amounts of data while providing low latency service to their customers. Unfortunately, using this globally distributed data to train large ML models is infeasible because (1) moving such large amounts of data over WAN is extremely slow (53.7x slower) and (2) data in specific regions is often subject to privacy and sovereignty laws thus prohibiting its transfer over WAN. These constraints motivate the need for distributed ML systems that can operate across multiple data centers globally.

Hypothesis

In this paper, *Hsieh et al.* seek to implement a geo-distributed ML system that efficiently utilizes the limited WAN bandwidth available in data centers while still achieving the same accuracy and correctness of large-scale machine learning algorithms that operate over LAN. They are faced with two key challenges (1) *How to effectively communicate over WANs while retaining algorithm convergence and accuracy?* and (2) *How to make the system generic and work for ML algorithms without requiring modification?*

Hsieh et al. theorize that by decoupling the synchronization model within the respective data centers and maintaining approximately correct copies of the global model within each data center they can maximize utilization of LAN bandwidth within the data center as well as make more efficient use of the scarce and heterogeneous WAN bandwidth.

Solution Overview

Hsieh et al. introduce GAIA, a general ML system that can be effectively deployed on WANs to allow for the training of ML models using geo-distributed training data. As shown in the figure below, they achieve this by doing the following:

1. To make Gaia effective on WANs while fully utilizing the abundant LAN bandwidth, they design a new system architecture to decouple the model synchronization within a data center (LANs) from the model synchronization across different data centers (WANs). In Gaia, each data center has some worker machines and parameter servers. Each worker machine processes a shard of the input data stored in its data center to achieve data parallelism. The parameter servers in each data center collectively maintain a version of the global model copy, and each parameter server handles a shard of this global model

copy. A worker machine only READs and UPDATEs the global model copy in its data center.

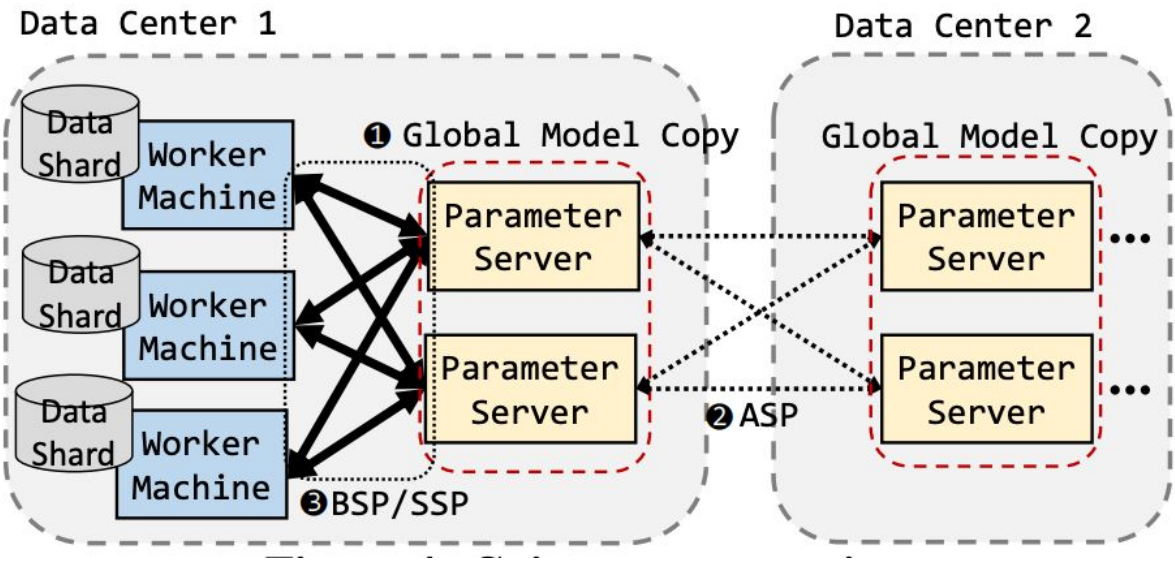

Figure 1: Gaia System Overview

2. They design a new ML synchronization model which they call ***Approximate Synchronous Parallel*** (ASP). ASP ensures that the global model copy in each data center is approximately correct. In this model, a parameter server shares only the significant updates with other data centers, and ASP ensures that these updates can be seen by all data centers in a timely fashion. ASP achieves this goal by using three techniques:
    a. **Significance filter** - Using a significance function and a significance threshold this filter indicates the significance of each update. With this, the parameter server aggregates updates from the local worker machines and shares the aggregated updates with other data centers only when the aggregated updates become significant.
    b. **ASP selective barrier** - When a parameter server receives significant updates at a rate that is higher than the WAN bandwidth can support, the parameter server first sends the indexes of these significant updates via an ASP selective barrier to the other data centers. The receiver of an ASP selective barrier blocks its local worker from reading the specified parameters until it receives the significant updates from the sender of the barrier.
    c. **ASP mirror clock** - When each parameter server receives all the updates from its local worker machines at the end of a clock cycle, it reports its clock to the servers that are in charge of the same parameters in the other data centers. When a server detects its clock is ahead of the slowest server that shares the same parameters by a predefined threshold DS (data center staleness), the

server blocks its local worker machines from reading its parameters until the slowest mirror server catches up.

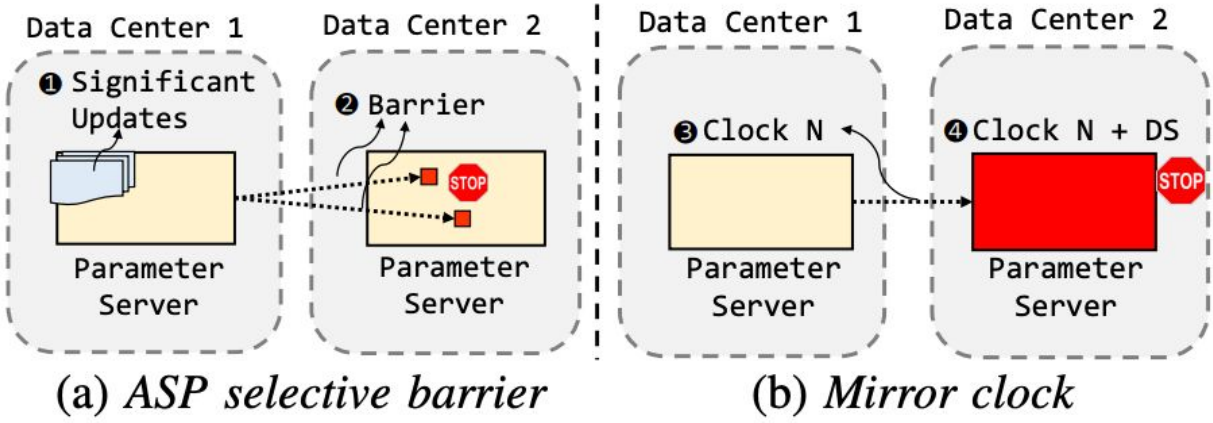

Figure 2: The Synchronization Mechanisms of ASP

Limitations and Possible Improvements

While showing promising results, this approach is heavily reliant on the implementation of a suitable significance function and threshold that allows for efficient performance and allows for the convergence of the ML algorithms. In addition, we believe that this paper can further be improved by expanding more on the issue of fault tolerance in their proposed solution. When dealing with such massive amounts of data distributed geographically across the world, the ability to recover is crucial.

Summary of Class Discussion

Q: Does anyone know whether Amazon, Google, etc are running something similar? Or they just train in one datacenter?

A: We haven't found any real-world applications.

Q: We don't have these bandwidth constraints in the datacenter but I'm not sure why this model wouldn't be even faster in this case? If we're adding computation and trying to overcome bandwidth limitations how could Gaia outperform a LAN? Would this only be the case when even the LAN is overwhelmed by bandwidth requirements?

A: This approach is not claiming to be faster than LAN, but rather approaching LAN speeds

Q: Is it possible that there is an ML training task so large that a single company might need multiple data centers worth of computing power and then use this method?

A: Yes

Q: 1) how to determine the global tendency 2) how to tell if local updates (from one datacenter) aligns with the global tendency, which seems to require either some centralized mechanism or more communication overhead

A: This question was based on a statement in slide 19 saying "unable to tell if local updates align with collaborative optimization". From reviewing the lecture video it seems that the presenters were actually making the point of comparing GAIA to CMFL's approach (Each client checks if its update aligns with the global tendency and is relevant enough to model improvement) for filtering local parameter updates. GAIA itself does not have any notion of a global tendency. Instead, it relies on the significance function and threshold to determine whether to share the aggregated updates with other data centers. However, since the significance function is determined by the ML programmer in GAIA, they implement a similar approach to CMFL to determine if local updates do align with the approximate global model stored in the data center.

Overall, the discussions on this paper were fruitful. Many discussions centered around determining the practicality of such an approach being deploying and working at scale. Some questioned the technique's reliance on the significance filter being precise, whether this approach was fault tolerant, also if the reported results were reliable given the variation in consistency models both in the LAN and WAN environments.

Summary of Towards Federated Learning At Scale: System Design

Problem and Motivation

Standard machine learning approaches require centralizing the training data on one machine or in a datacenter. When dealing with massive amounts of data such as the kind used by Google, having data locality on a single machine or datacenter is infeasible. Federated Learning focuses on the general approach of “bringing the code to the data, instead of the data to the code” and addresses the fundamental problems of privacy, ownership, and locality of data.

Federated Learning enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on the device, decoupling the ability to do machine learning from the need to store the data in the cloud. This goes beyond the use of local models that make predictions on mobile devices by bringing model training to the device as well.

Hypothesis

In this paper, Bonawitz et al. introduce their system design for allowing the distributed training of ML algorithms across mobile devices using the Android operating system. They attempt to sketch the major components of the system, describe the challenges and identify the open issues, in the hope that this will be useful to spark further systems research. Their work addresses a series of practical issues such as:

1. Device availability
2. Device connectivity and interrupted execution
3. Execution orchestration
4. Limited device storage and compute resources

Solution Overview

The goal is to build a system that can train ML algorithms on data stored on the phone which will never leave the device. The weights are combined in the cloud using an approach known as Federated Averaging, which constructs a global model that is then pushed back to the phone for inference.

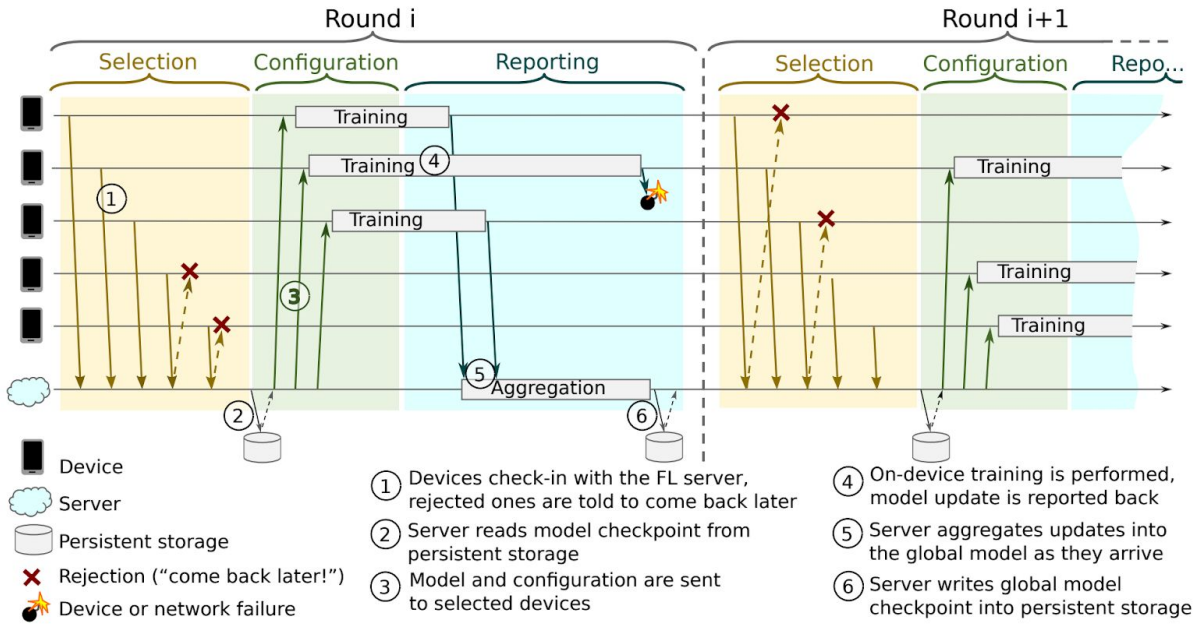

Figure 1: Federated Learning Protocol

The FL communication protocol enables devices to advance the global, singleton model of an FL population between rounds where each round consists of three phases. These phases are:

1. **Selection** - Periodically, devices that meet the eligibility criteria check in to the server. The server selects a subset of connected devices based on certain goals like the optimal number of participating devices.
2. **Configuration** - The server is configured based on the aggregation mechanism selected for the selected devices. The server sends the FL plan and an FL checkpoint with the global model to each of the devices.
3. **Reporting** - The server waits for the participating devices to report updates. As updates are received, the server aggregates them using Federated Averaging and instructs the reporting devices when to reconnect. If enough devices report in time, the round will be successfully completed and the server will update its global model, otherwise, the round is abandoned.

Device Architecture

The device's first responsibility in on-device learning is to maintain a repository of locally collected data for model training and evaluation. Applications are responsible for making their data available to the FL runtime as an example store by implementing an API provided by Google. When a task arrives at the device, the FL runtime will access the appropriate example store to compute model updates.

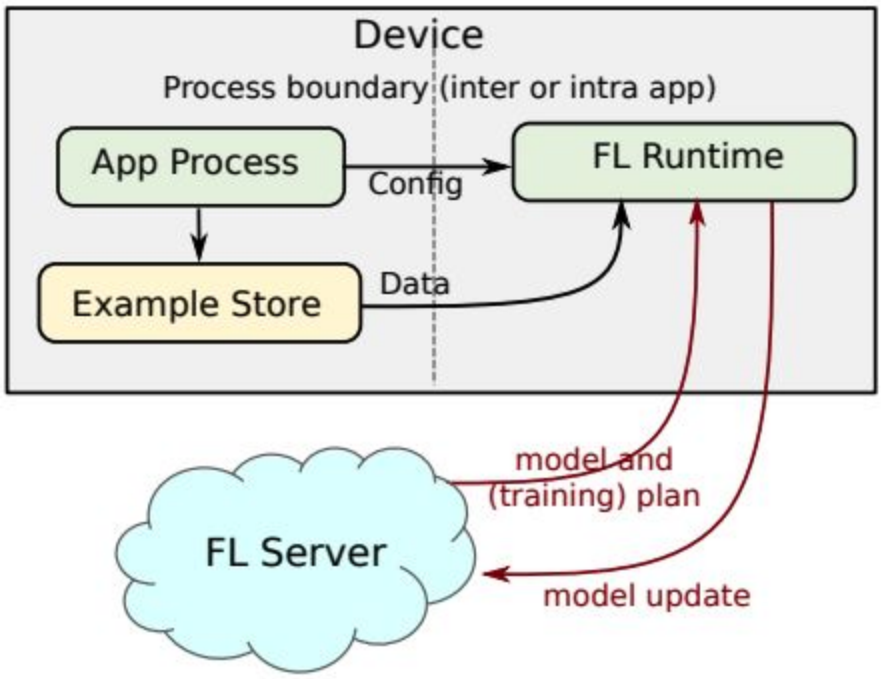

Figure 2: Device Architecture

Server

The design of the FL server is driven by the necessity to operate over many orders of magnitude of population sizes and other dimensions. The FL server is designed around the Actor Programming Model. Actors are universal primitives of concurrent computation that use message passing as the sole communication mechanism. The main actors in the system include:

1. **Coordinators -** are Top-level actors that enable global synchronization and advancing rounds in lockstep. A Coordinator registers its address and the FL population it manages in a shared locking service, so there is always a single owner for every FL population which is reachable by other actors in the system. The Coordinator receives information about how many devices are connected to each Selector and instructs them how many devices to accept for participation, based on which FL tasks are scheduled
2. **Selectors -** are responsible for accepting and forwarding device connections. After the Master Aggregator and set of Aggregators are spawned, the Coordinator instructs the Selectors to forward a subset of its connected devices to the Aggregators, allowing the Coordinator to efficiently allocate devices to FL tasks regardless of how many devices are available.
3. **Master Aggregators -** manage the rounds of each FL task. In order to scale with the number of devices and update size, they make dynamic decisions to spawn one or more Aggregators to which work is delegated.

Federated Averaging

FedAvg is a variation of the Stochastic Gradient Descent (SGD) algorithm, which combines local SGD on each client with a server that performs model averaging.

At the beginning of each round, a random fraction C of clients is selected, and the server sends the current global algorithm state to each of these clients (e.g., the current model parameters). They only select a fraction of clients for efficiency, as the experiments show diminishing returns for adding more clients beyond a certain point. Each selected client then performs local computation based on the global state and its local dataset and sends an update to the server. The server then applies these updates to its global state, and the process repeats.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Figure 3: Federated Learning Algorithm

Limitations and Possible Improvements

Current limitations of this implementation include:

- **Bias** - The FL protocol assumes that all devices are equally likely to participate and complete each round. In practice, the system potentially introduces bias by the fact that

devices only train when they are on an unmetered network and charging. In some countries, the majority of people rarely have access to an unmetered network.
- **Convergence Time** - FI has a slower convergence time when compared to ML on centralized data where training is backed by the power of a data center.
- **Better device sampling** - FL currently randomly selects devices. This opens the possibility for improved performance by discovering better strategies for selecting participating devices e.g data quality, device resources.

Summary of Class Discussion

Q: How do they do backward propagation? In the case of DNNs, do they do the forward propagation on the device or both?

A: Both are done on the device

Q: I assume binaural oscillations refer to the fact that most phones will not be active while their owner is asleep with their phone charged in (at night)? If I understand their constraints correctly, it seems like this is the time when most phones would be used for training to avoid impacting user experience/battery life.

A: Yes

Q: I'm surprised they have such a low drop out rate of 6 - 8%

A: They addressed this by oversampling at 130%

Q: So this is an optimization in model accuracy, not in system performance correct?

A: This is a reference to the device selection process where we were discussing that we can optimize by filtering out devices that do not consist of any new data to add to the model. This can count as both accuracy and performance optimization.

Discussions on this paper were interesting. We discussed at length the topic of model manipulation from device farms that contain uncompromised phones. This is a very difficult problem to solve that may influence models in the future.