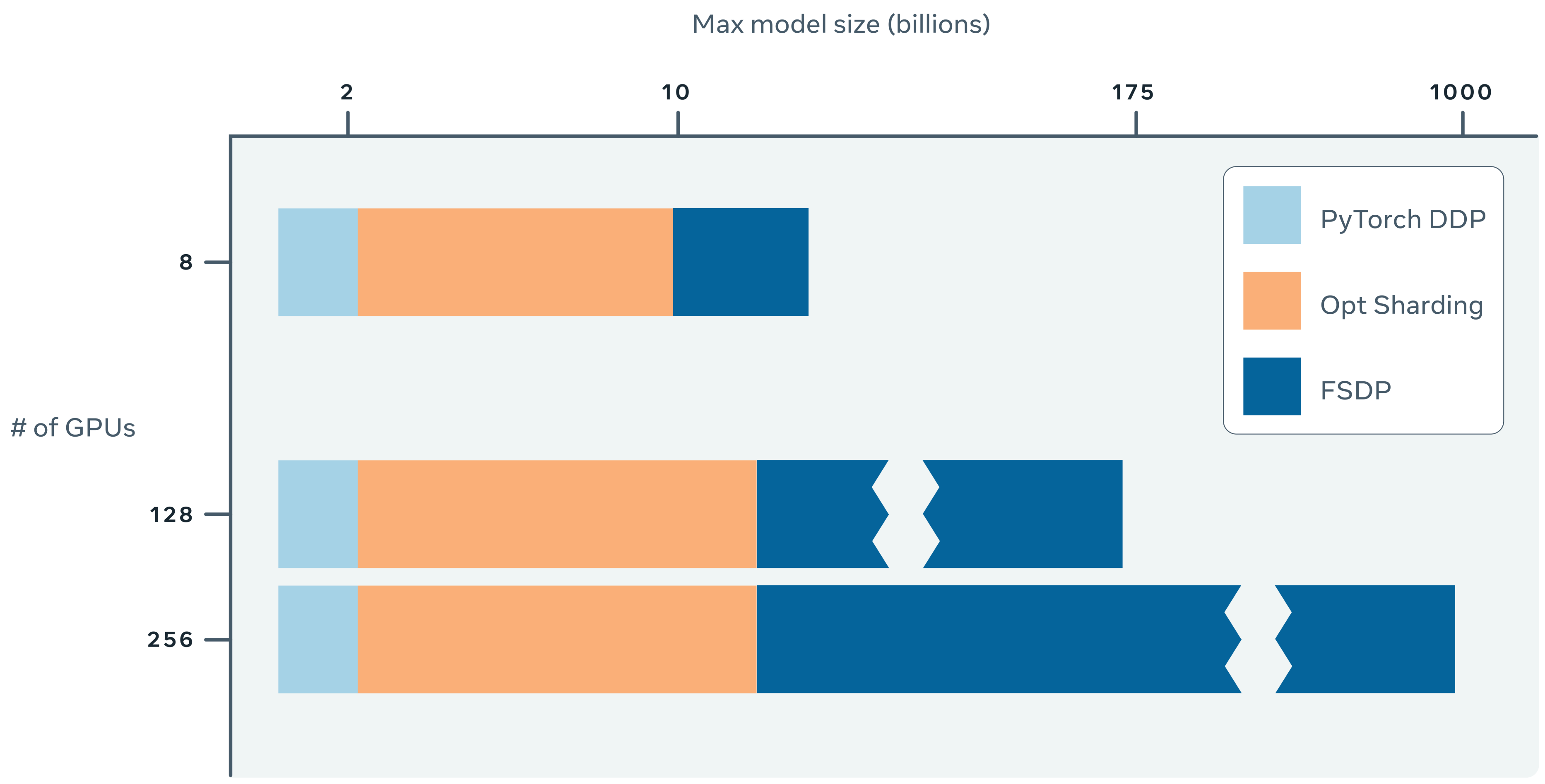
POSTED ON JULY 15, 2021 TO AI RESEARCH, DATA CENTER ENGINEERING, ML APPLICATIONS, OPEN SOURCE, PRODUCTION ENGINEERING

Fully Sharded Data Parallel: faster AI training with fewer GPUs



By Myle Ott, Sam Shleifer, Min Xu, Priya Goyal, Quentin Duval, Vittorio Caggiano

Training AI models at a large scale isn't easy. Aside from the need for large amounts of computing power and resources, there is also considerable engineering complexity behind training very large models. At Facebook AI Research (FAIR) Engineering, we have been working on building tools and infrastructure to make training large AI models easier. Our recent work in areas such as intra-layer model parallelism, pipeline model parallelism, optimizer state+gradient sharding, and mixture of experts is just part of our work to make training advanced AI models for any number of tasks more efficient.

Fully Sharded Data Parallel (FSDP) is the newest tool we're introducing. It shards an AI model's parameters across data parallel workers and can optionally offload part of the training computation to the CPUs. As its name suggests, FSDP is a type of data-parallel training algorithm. Although the parameters are sharded to different GPUs, the computation for each microbatch of data is still local to each GPU worker. This conceptual simplicity makes FSDP easier to understand and more applicable to a wide range of usage scenarios (compared with intra-layer parallelism and pipeline parallelism). Compared with optimizer state+gradient sharding data parallel methods, FSDP shards parameters more uniformly and is capable of better performance via communication and computation overlapping during training.

With FSDP, it is now possible to more efficiently train models that are orders of magnitude larger using fewer GPUs. FSDP has been implemented in the FairScale library and allows engineers and developers to scale and optimize the training of their models with simple APIs. At Facebook, FSDP has

The high computational cost of large-scale training

NLP research is one particular area where we can see the importance of efficiently leveraging compute for training AI. Last year, OpenAI announced that they had trained GPT-3, the largest-ever neural language model, with 175 billion parameters. It is estimated to have taken roughly 355 GPU years to train GPT-3, or the equivalent of 1,000 GPUs working continuously for more than four months.

Besides requiring a lot of compute and engineering resources, most approaches to scaling like this introduce additional communication costs and require engineers to carefully evaluate trade-offs between memory use and computational efficiency. For example, typical data parallel training requires maintaining redundant copies of the model on each GPU, and model parallel training introduces additional communication costs to move activations between workers (GPUs).

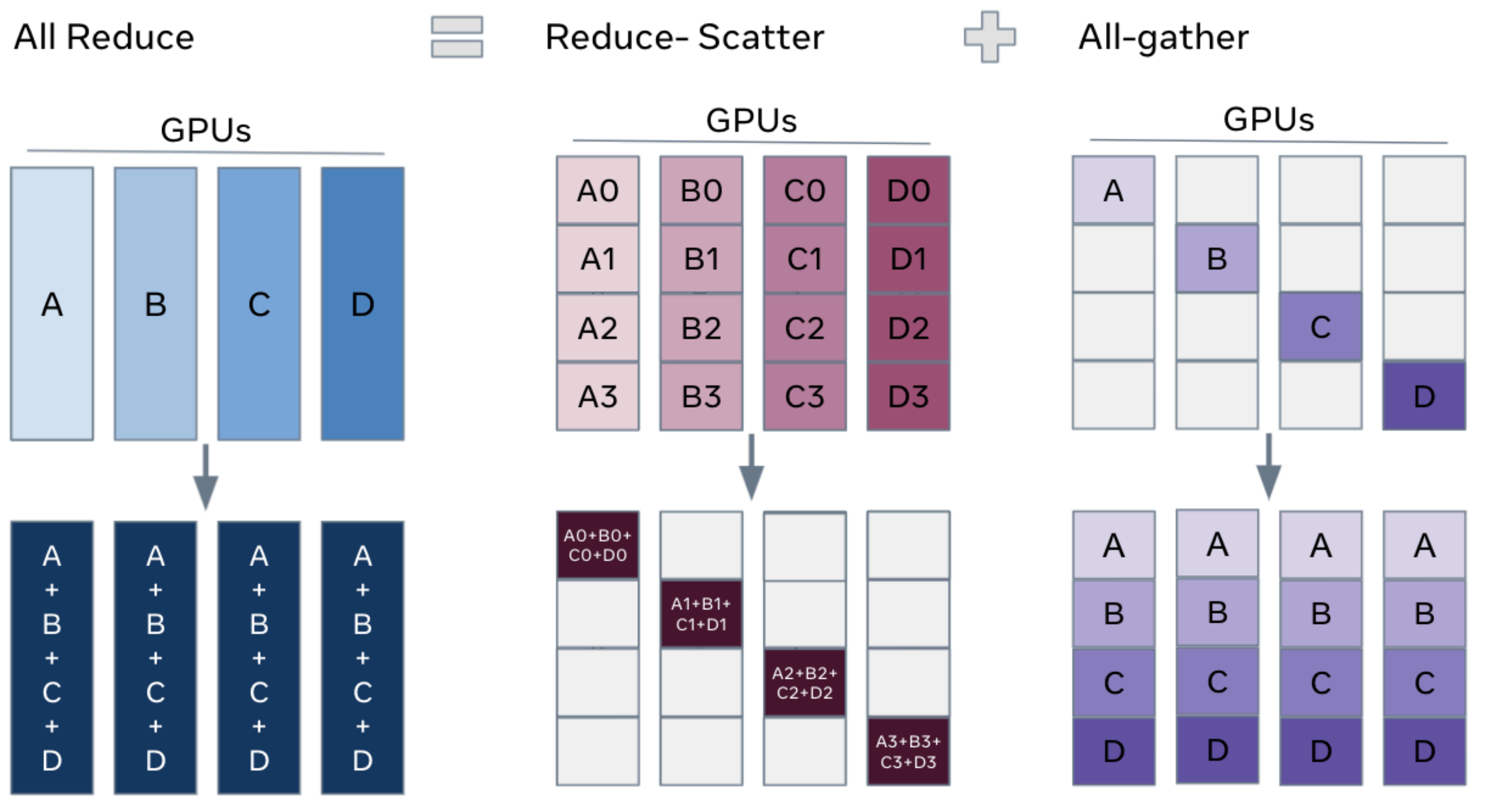
FSDP is relatively free of trade-offs in comparison. It improves memory efficiency by sharding model parameters, gradients, and optimizer states across GPUs, and improves computational efficiency by decomposing the communication and overlapping it with both the forward and backward passes. FSDP produces identical results as standard distributed data parallel (DDP) training and is available in an easy-to-use interface that’s a drop-in replacement for PyTorch’s DistributedDataParallel module. Our early testing has shown that FSDP can enable scaling to trillions of parameters.

How FSDP works

In standard DDP training, every worker processes a separate batch and the gradients are summed across workers using an all-reduce operation. While DDP has become very popular, it takes more GPU memory than it needs because the model weights and optimizer states are replicated across all DDP workers.

One method to reduce replications is to apply a process called full parameter sharding, where only a subset of the model parameters, gradients, and optimizers needed for a local computation is made available. An implementation of this method, ZeRO-3, has already been popularized by Microsoft.

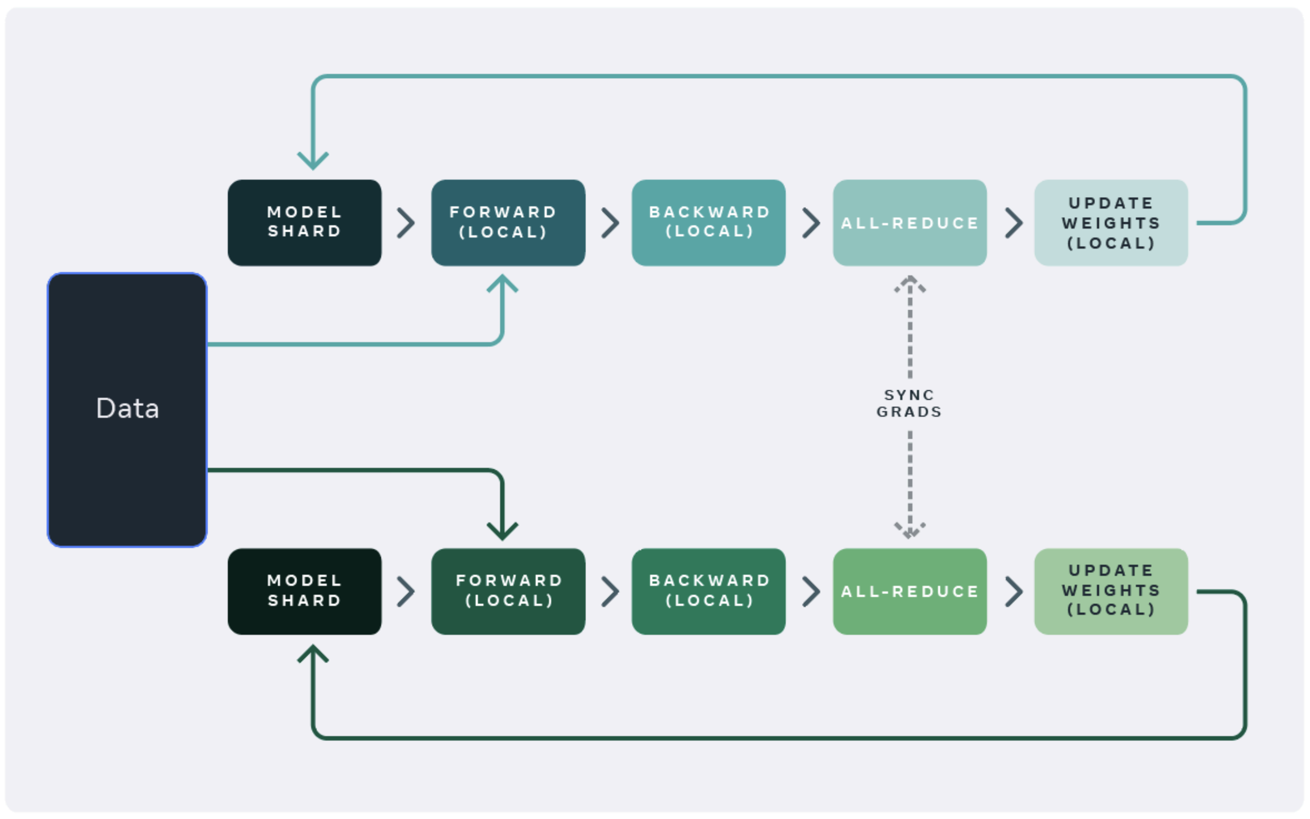
The key insight to unlock full parameter sharding is that we can decompose the all-reduce operations in DDP into separate reduce-scatter and all-gather operations:



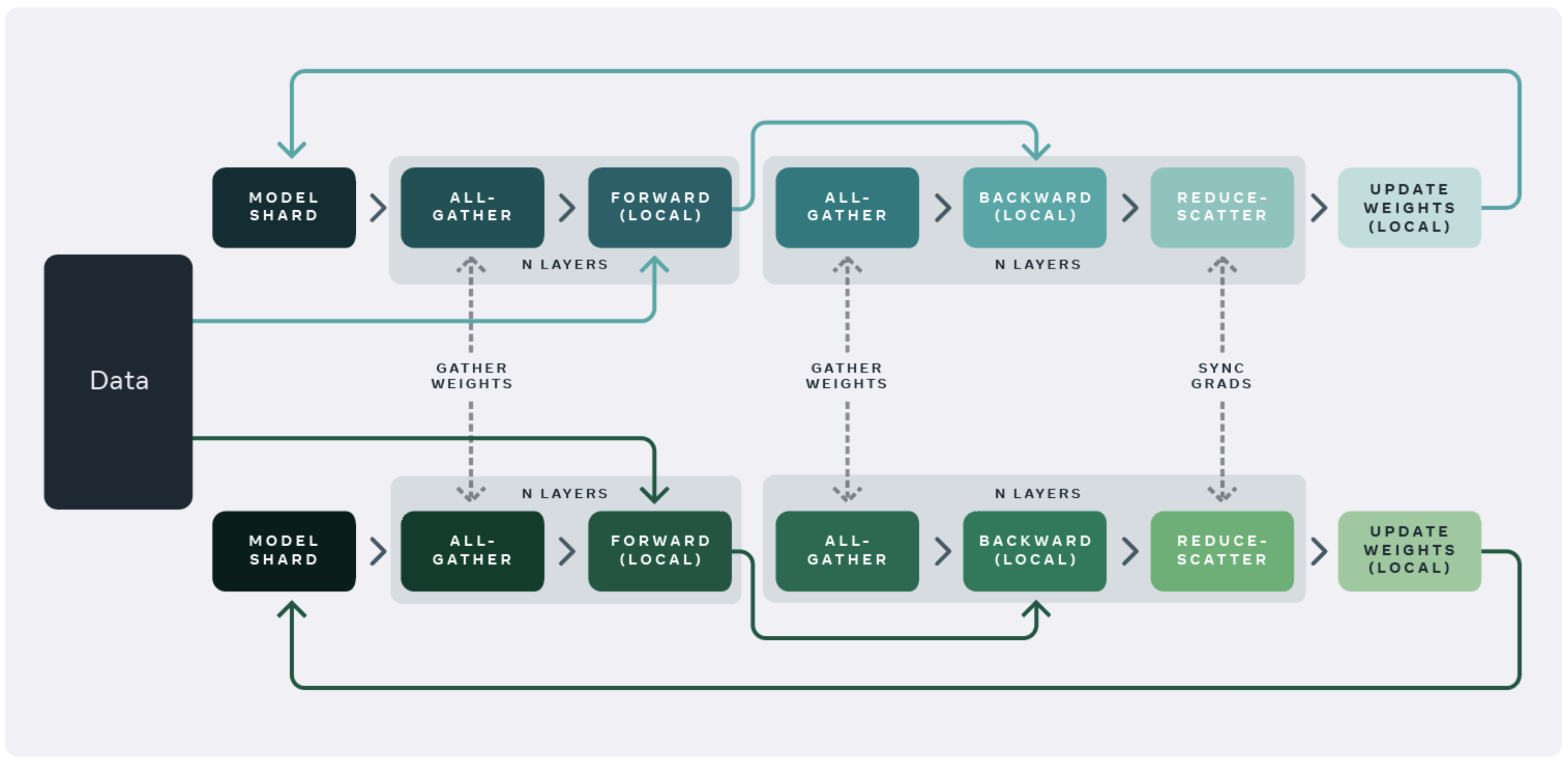
All-reduce as a combination of reduce-scatter and all-gather. The standard all-reduce operation to aggregate gradients can be decomposed into two separate phases: reduce-scatter and all-gather. During the reduce-scatter phase, the gradients are summed in equal blocks among ranks on each GPU based on their rank index. During the all-gather phase, the sharded portion of aggregated gradients available on each GPU are made available to all GPUs (see here for details on those operators).

We can then rearrange the reduce-scatter and all-gather so that each DDP worker needs to store only a single shard of parameters and optimizer states. The figure below illustrates standard DDP training (top) and FSDP training (bottom):

Standard data parallel training



Fully sharded data parallel training



A comparison of standard data parallel training and fully sharded data parallel training. In standard data parallel training methods, a copy of the model is present on each GPU and a sequence of forward and backward passes are evaluated on only a shard of the data. After these local computations, the parameters and optimizers for each local process are shared with the other GPUs in order to calculate the global weight update. In FSDP, only a shard of the model is present on a GPU. Then, locally, all weights are gathered from the other GPUs — by means of an all-gather step — to calculate the forward pass. This gathering of weights is then performed again before the backward pass. After that backward pass, the local gradients are averaged and sharded across the GPUs by means of a reduce-scatter step, which allows each GPU to update its local weight shard.

To maximize memory efficiency, we can discard the full weights after each layer's forward pass, saving memory for subsequent layers. This can be implemented by applying the FSDP wrapper to every layer in the network (with `reshard_after_forward=True`).

```
FSDP forward pass:
    for layer_i in layers:
        all-gather full weights for layer_i
        forward pass for layer_i
        discard full weights for layer_i

FSDP backward pass:
    for layer_i in layers:
        all-gather full weights for layer_i
        backward pass for layer_i
        discard full weights for layer_i
        reduce-scatter gradients for layer_i
```

How to use FSDP

There are several ways to use FSDP in large-scale AI research. At this time, we offer four solutions to adapt to different needs.

1. Using FSDP in language models

For language models, FSDP is supported in the *fairseq* framework via the following new arguments:

- `–ddp-backend=fully_sharded`: enables full sharding via FSDP
- `–cpu-offload`: offloads the optimizer state and FP32 model copy to CPU (combine with`–optimizer=cpu_adam`)
- `–no-reshard-after-forward`: increases training speed for large models (1B+ params) and is similar to ZeRO stage 2
- Other popular options (`–fp16`, –update-freq, `–checkpoint-activations`, `–offload-activations`, etc.) continue to work as normal

See the fairseq tutorial for instructions on using FSDP to train a 13B-parameter model on eight GPUs or on a single GPU with FSDP + CPU offloading.

2. Using FSDP in computer vision models

For computer vision models, FSDP is supported in VISSL and tested on RegNets architectures. Layers like BatchNorm and ReLU are seamlessly handled and tested for convergence.

Use the following options to enable FSDP:

- `config.MODEL.FSDP_CONFIG.AUTO_SETUP_FSDP=True`
- `config.MODEL.SYNC_BN_CONFIG.SYNC_BN_TYPE=pytorch`
- `config.MODEL.AMP_PARAMS.AMP_TYPE=pytorch`

See this section of the yaml config for additional options to config FSDP within VISSL.

3. Using FSDP from PyTorch Lightning

For easier integration with more general use cases, FSDP is supported as a beta feature by PyTorch Lightning. This tutorial contains a detailed example on how to use the FSDP plugin with PyTorch Lightning. At a high level, adding `plugins='fsdp'` below can activate it.

```
model = MyModel()
trainer = Trainer(gpus=4, plugins='fsdp', precision=16)
```

4. Using the FSDP library directly from FairScale

The main library where FSDP has been developed, and where you can find the latest updates, is FairScale. You can directly use FSDP from FairScale with the below example by simply replacing the `DDP(my_module)`:

```
from fairscale.nn.data_parallel import FullyShardedDataParallel as FSDP
...
sharded_module = ~~DDP(my_module)~~FSDP(my_module)
optim = torch.optim.Adam(sharded_module.parameters(), lr=0.0001)
for sample, label in dataload.next_batch:
  out = sharded_module(x=sample, y=3, z=torch.Tensor([1]))
  loss = criterion(out, label)
  loss.backward()
  optim.step()
```

The FSDP library in FairScale exposes the low-level options for many important aspects of large-scale training. Here are some few important areas to consider when you apply FSDP with its full power.

1. **Model wrapping:** In order to minimize the transient GPU memory needs, users need to wrap a model in a nested fashion. This introduces additional complexity. The auto_wrap utility is useful in annotating existing PyTorch model code for nested wrapping purposes.
2. **Model initialization:** Unlike DDP, FSDP does **not** automatically synchronize model weights between GPU workers. This means model initialization must be done carefully so that all GPU workers have the identical initial weights.
3. **Optimizer settings:** Due to sharding and wrapping, only certain types of optimizer and optimizer settings are supported by FSDP. In particular, if a module is wrapped by FSDP and its parameters are flattened into a single tensor, users cannot use different hyperparameters for different parameter groups in such a module.
4. **Mixed precision:** FSDP supports advanced mixed precision training with FP16 master weights, as well as FP16 reduce and scatter on the gradients. Certain parts of a model may converge only if full precision is used. In those cases, additional wrapping is needed to selectively run parts of a model in full precision.
5. **State checkpointing and inference:** When the model scale is large, saving and loading the model state can become challenging. FSDP supports several ways to make that task possible, but it is by no means trivial.
6. Finally, FSDP is often used together with **activation checkpointing** functions like checkpoint_wrapper from FairScale. Users may need to carefully tune the activation checkpointing strategy to fit a large model within limited GPU memory space.

Next steps

FSDP is open source, and early users have tried it and contributed to it. We think it can benefit the entire research community, and we look forward to working with everyone in making it better. In particular, these are some of the important areas.

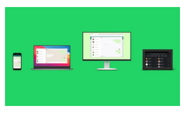
1. **Making FSDP more general.** So far, FSDP has been used on both NLP and vision models with SGD and Adam optimizers. As newer models and optimizers emerge, FSDP needs to continue supporting them. Being a purely data-parallel training scheme, FSDP has the greatest potential to be general in supporting a wide range of AI algorithms.
2. **Making FSDP auto-tune.** There are many knobs that users can tune today with FSDP for both scaling and performance. We look forward to developing algorithms for auto-tuning both GPU memory usage and training performance.

Try it out and contribute!

FSDP is currently available directly from the FairScale library.

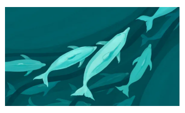
Thanks for sticking with us thus far. Please try FSDP in your research or production work. We would love to hear your feedback, and, as always, pull requests are welcome!

Like Share 223 people like this. Sign Up to see what your friends like.



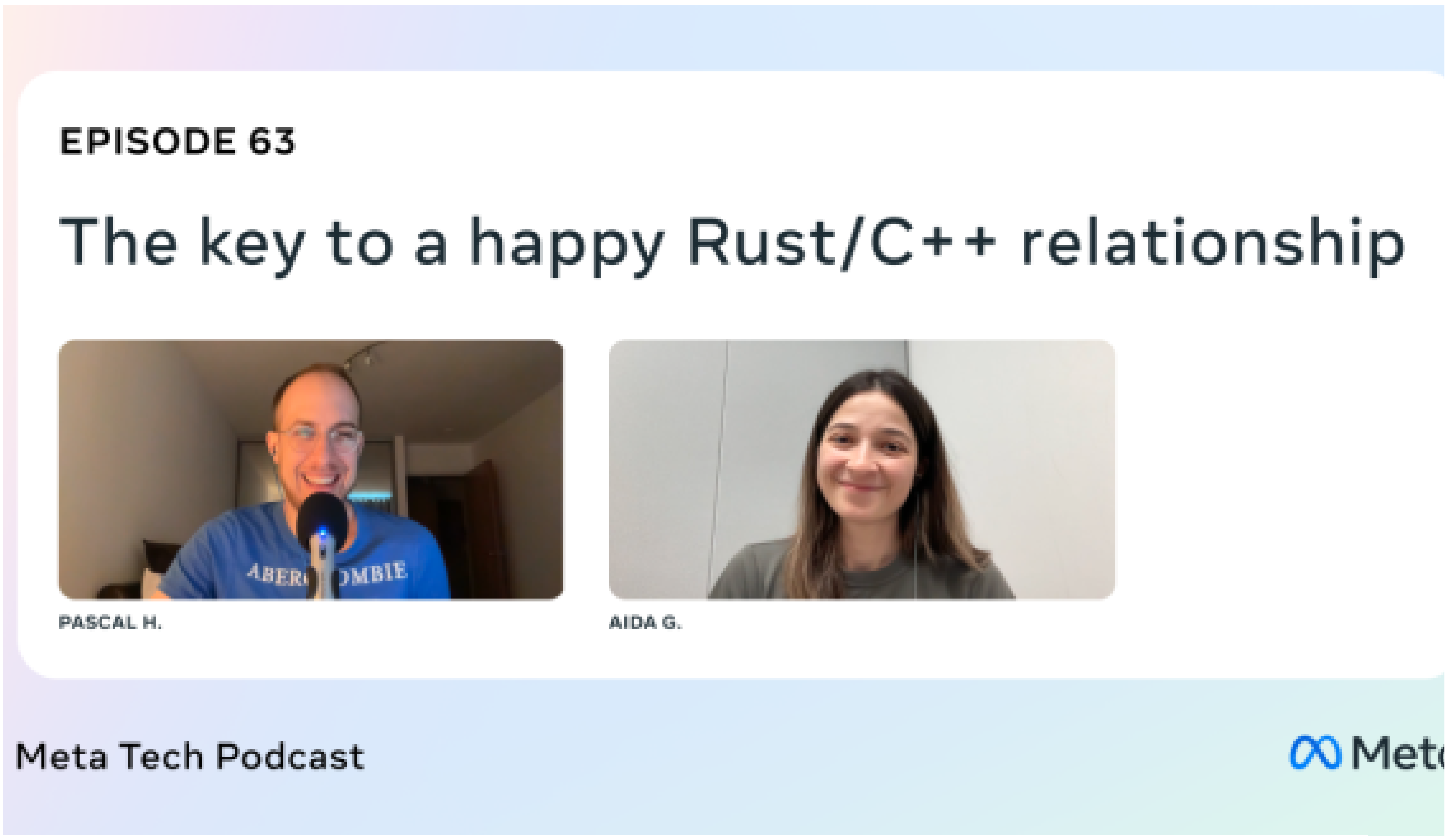
◂ **Prev**
How WhatsApp enables multi-device capability

Next ▸
Migrating Facebook to MySQL 8.0



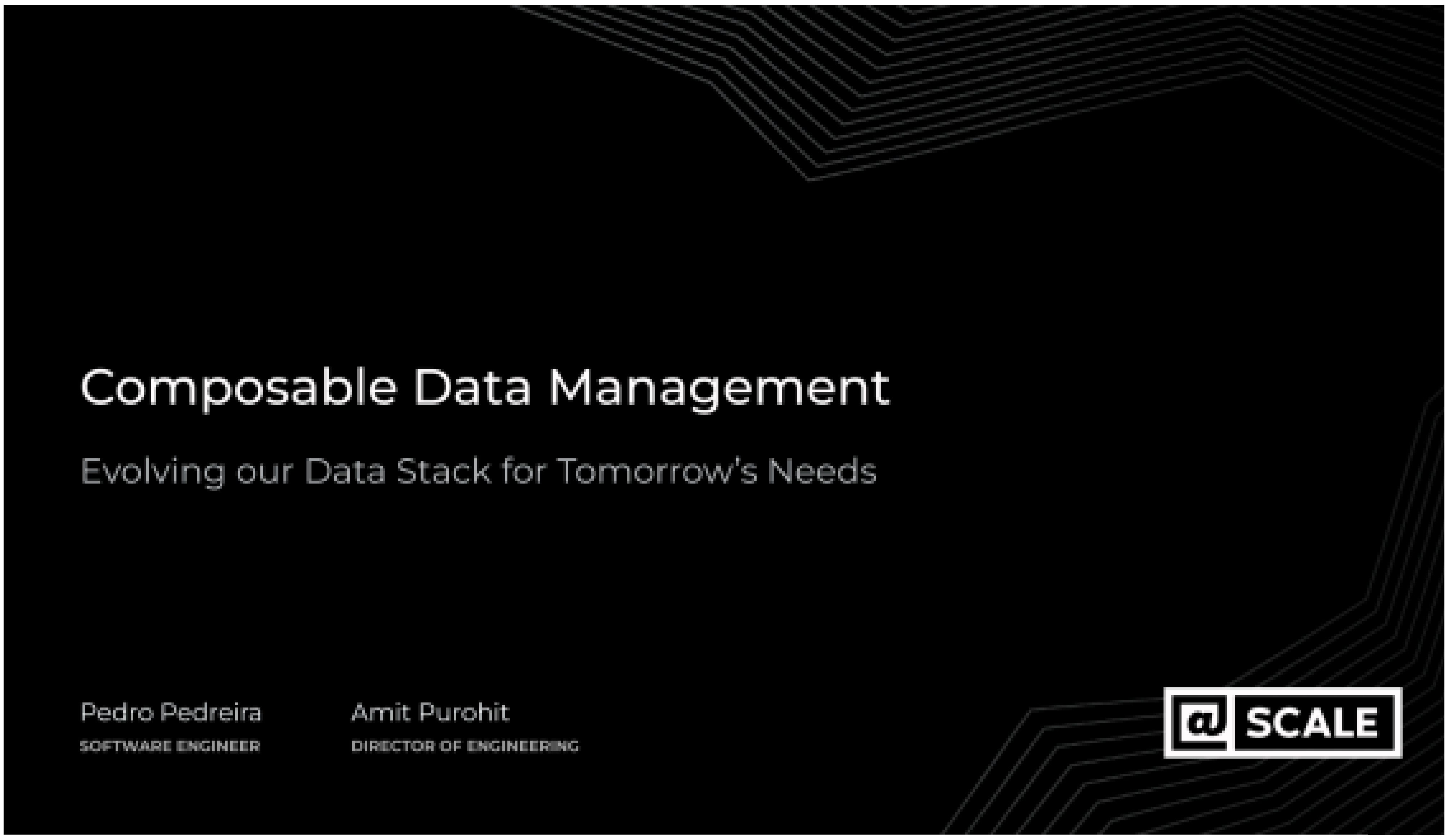
Read More in Open Source

View Al



JUN 25, 2024

The key to a happy Rust/C++ relationship



Composable Data Management

Evolving our Data Stack for Tomorrow's Needs

Pedro Pedreira
SOFTWARE ENGINEER

Amit Purohit
DIRECTOR OF ENGINEERING

@ SCALE

MAY 22, 2024

Composable data management at Meta

FEB 26, 2024

How DotSlash makes executable deployment simpler

FEB 20, 2024

Aligning Velox and Apache Arrow: Towards composable data management

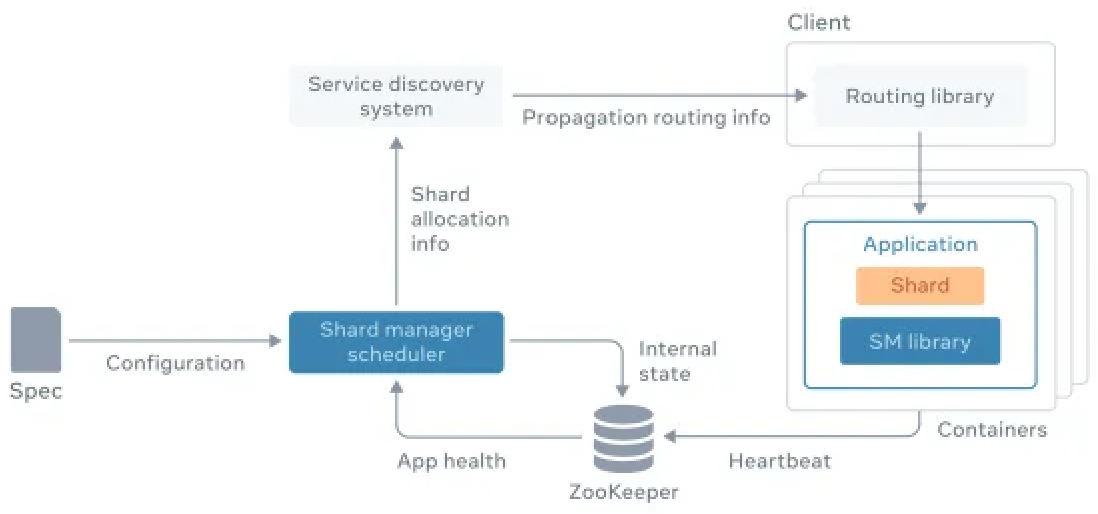
FEB 6, 2024

DotSlash: Simplified executable deployment

OCT 23, 2023

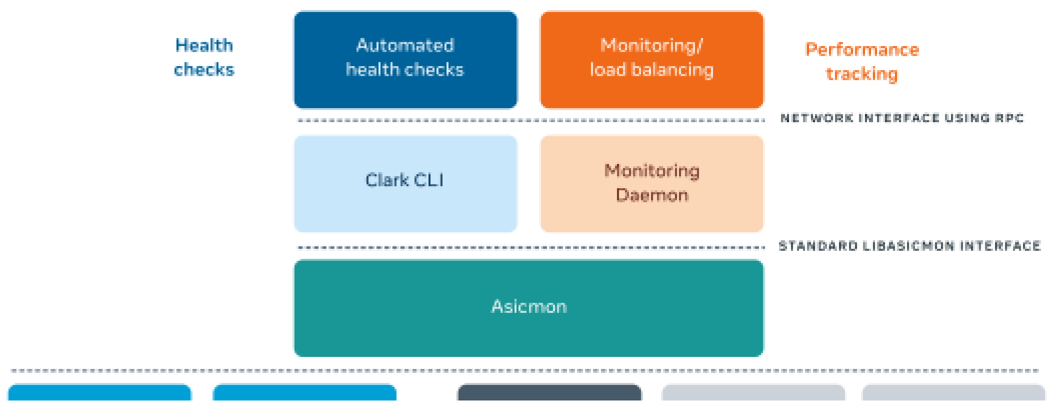
5 Things you didn't know about Buck2

Related Posts



Aug 24, 2020

Scaling services with Shard Manager



Oct 22, 2018

Significantly faster generation and training for AI-based audio systems

Related Positions

Research Engineer - Reality Labs
MENLO PARK, US

Research Engineer - Reality Labs
BURLINGAME, US

Research Engineer - Reality Labs
NEW YORK, US

Applied Research Scientist, Speech
MENLO PARK, US

Applied Research Scientist, Speech
SEATTLE, US

See All Jobs

Available Positions

Research Engineer - Reality Labs
MENLO PARK, US
Research Engineer - Reality Labs
BURLINGAME, US
Research Engineer - Reality Labs
NEW YORK, US
Applied Research Scientist, Speech
MENLO PARK, US
Applied Research Scientist, Speech
SEATTLE, US

See All Jobs

Technology at Meta

Engineering at Meta - X
Follow

AI at Meta
Read

Open Source

Meta believes in building community through open source technology. Explore our latest projects in Artificial Intelligence, Data Infrastructure, Development Tools, Front End, Languages, Platforms, Security, Virtual Reality, and more.

ANDROID

Meta for Developers

Read

Meta Bug Bounty

Learn more

RSS

Subscribe

BACKEND

HARDWARE

Learn More

Meta

Engineering at Meta is a technical news resource for engineers interested in how we solve large-scale technical challenges at Meta.

Home

Company Info

Careers