简化数据获取！Uber开源深度学习分布训练库Petastorm

原创：AI前线小组 AI前线 9月24日

策划编辑 | Natalie
作者 | Uber ATG
翻译 | 无明
编辑 | Natalie

AI 前线导读： 近年来，深度学习在解决模式识别问题方面发挥了关键作用。Uber Advanced Technologies Group（ATG）使用深度学习来解决自动驾驶领域的各种问题，他们的很多模型需要来自传感器数十 TB 的训练数据。Uber ATG 的研究人员和工程师正在积极推动跨多个问题领域的自动驾驶技术，如感知、预测和规划。为了支持这些工作，他们致力于开发数据集存储解决方案，让研究人员更容易获得数据，从而可以专注于模型实验。本文将介绍 Petastorm，一个由 Uber ATG 开发的开源数据访问库。这个库可以直接基于数 TB Parquet 格式的数据集进行单机或分布式训练和深度学习模型评估。Petastorm 支持基于 Python 的机器学习框架，如 Tensorflow、Pytorch 和 PySpark，也可以直接用在 Python 代码中。

更多优质内容请关注微信公众号“AI 前线”（ID：ai-front）

深度学习集群的搭建

即使是在现代硬件上训练深度模型也很耗时，而且在很多情况下，很有必要在多台机器上分配训练负载。典型的深度学习集群需要执行以下几个步骤：

- 一台或多台机器读取集中式或本地数据集。
- 每台机器计算损失函数的值，并根据模型参数计算梯度。在这一步通常会使用 GPU。
- 通过组合估计的梯度（通常由多台机器以分布式的方式计算得出）来更新模型系数。

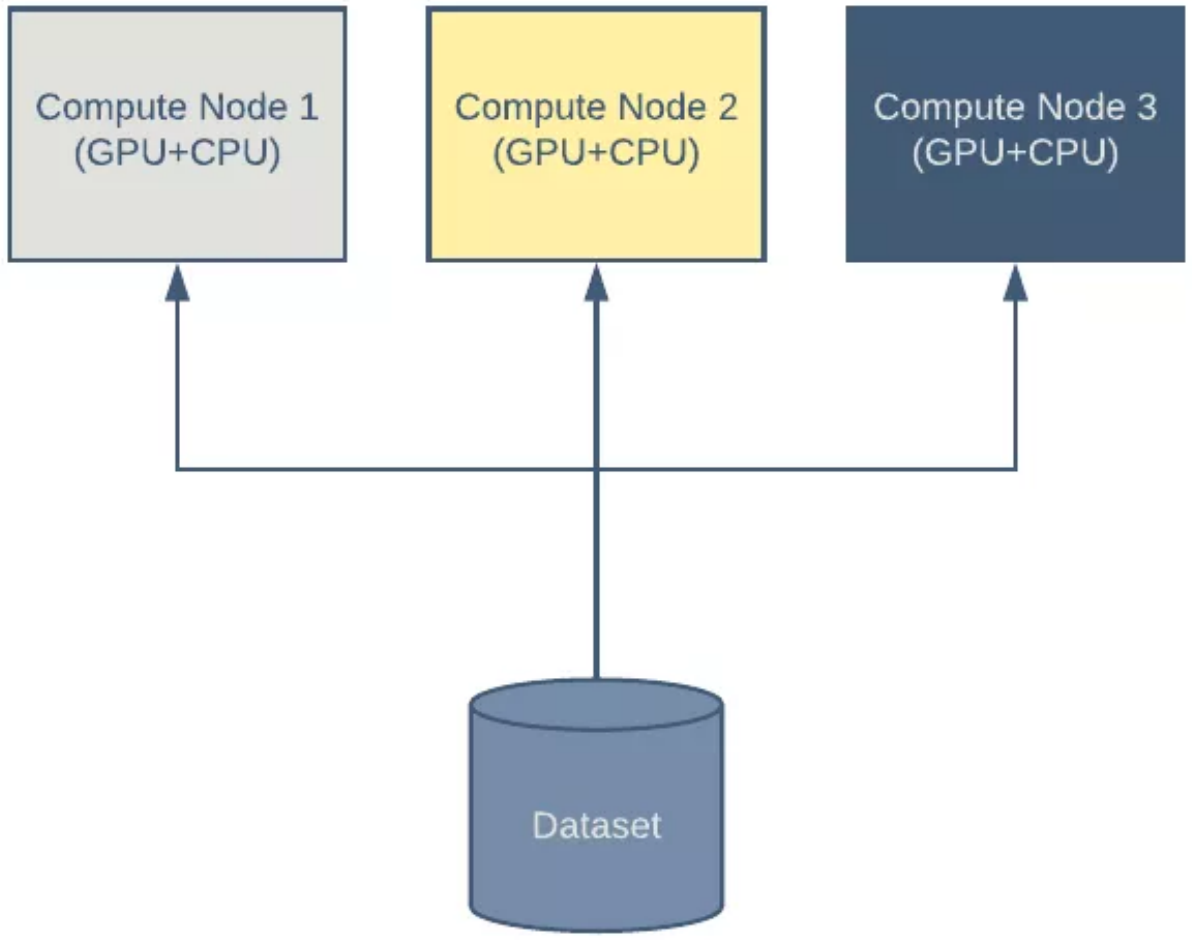

图 1：在这个深度学习集群架构中，有三个计算节点使用中央数据集。

考虑到 GPU 的成本，很有必要提高 GPU 集群的利用率。经过调优的数据访问层可以确保用于训练的数据对 GPU 总是可用的，这样 GPU 就不会处于空闲状态。

简化模型架构研究

准备数 TB 来自多个数据源的同步数据通常很容易出错。我们希望为研究人员提供单个数据集，让他们可以处理各种任务，无需为每种任务创建新的数据集。

为此，需要遵守以下原则：

- 数据集需要包含研究人员可能用到的数据的超集，这样他们就可以为特定实验选择列和行的子集。
- 对数据集中的传感器数据的预处理应该保持在最低限度。我们鼓励研究人员进行实时的预处理，并
- 将其作为训练 / 评估程序的一部分。在很多情况下，这可以通过其他未被充分利用的 CPU 来完成。

在业界，深度学习应用程序的数据集存储通常分为两类：多文件和记录流式数据集。

多文件数据集

在这种情况下，每个张量 / 图像 / 标签集被保存在单独的文件（例如，PNG、JPEG、NPZ 和 CSV）中。整个数据集被存储为一个或多个文件系统目录，每个目录包含大量的文件。文件数量可能达到数百万个（例如，ImageNet 有 120 万个文件）。如果以这种格式存储，Uber ATG 的数据集将超过 1 亿个文件。

这种方法让用户可以随机访问数据集中任何行的任何列。但是，多次往返文件系统的成本很高，所以很难大规模实现，特别是在使用现代分布式文件系统时，如 HDFS 和 S3（这些系统通常针对大块数据的快速读取进行了优化）。

记录流式数据集

另外一种方式是将数据行的集合组合在一起，保存成一个或多个文件。例如，Tensorflow 使用 protobuf 文件（TFRecord）。其他流行的格式还包括 HDF5 和 Python pickle 文件。

这种方法适用于 HDFS 和 S3 文件系统。但是，查询特定列需要通过网络传输所有字段，然后丢弃未使用的数据。如果要查询单行，还需要自定义索引。

在评估了多个方案后，我们决定使用 Apache Parquet 存储格式，它在一定程度规避了上述两种方法的一些缺点：

- 便于进行大量连续读取（对 HDFS/S3 友好）；
- 支持快速访问单个列；
- 在某些情况下允许更快的行查询；
- 与 Apache Spark 完美集成，可作为现成的查询 / 操作框架。

列式存储和 Apache Parquet

列式数据存储按照列（而不是行）的顺序来组织数据。例如，从自动驾驶车辆传感器记录的数据可能看起来像这样：

Row	camera #1	camera #2	Lidar	Labels
1	‹camera1-1›	‹camera2-1›	‹lidar 1›	‹labels 1›
2	‹camera1-2›	‹camera2-2›	‹lidar 2›	‹labels 2›
3	‹camera1-3›	‹camera2-3›	‹lidar 3›	‹labels 3›

行和列存储之间的差异如下所示：

	Row storage		Columnar storage
row 1	<camera1-1>		<camera1-1>
	<camera2-1>		<camera1-2>
	<lidar 1>		<camera1-3>
	<labels 1>		<camera2-1>
row 2	<camera1-2>		<camera2-2>
	<camera2-2>		<camera2-3>
	<lidar 2>		<lidar 1>
	<labels 2>		<lidar 2>
row 3	<camera1-3>		<lidar 3>
	<camera2-3>		<labels 1>
	<lidar 3>		<labels 2>
	<labels 3>		<labels 3>

以列式顺序存储数据允许用户只加载列的子集，从而减少通过网络传输的数据量。对于收集来自自动驾驶车辆传感器的数据来说，这种好处是显而易见的：试想一下，如果你的实验只需要来自某个摄像头的图像，那么就可以从同一行的 10 张高分辨率图像中加载其中的一张。

Apache Parquet 是一种列式存储格式，近年来越来越流行。它得到了 Apache Spark、Apache Arrow 和其他开源项目的支持，并且非常适合用于进行简化模型架构研究。

Tensorflow 和 Pytorch 是深度学习社区常用的框架。这些框架本身并不支持 Parquet 存储访问，因此我们构建了 Petastorm 来填补这一空白。

Petastorm 简介

通常，一个数据集是通过连接多个数据源的记录而生成的。这个由 Apache Spark 的 Python 接口 PySpark 生成的数据集稍后将被用在机器学习训练中。Petastorm 提供了一个简单的功能，使用 Petastorm 特定的元数据对标准的 Parquet 进行了扩展，从而让它可以与 Petastorm 兼容。

有了 Petastorm，消费数据就像在 HDFS 或文件系统中创建和迭代读取对象一样简单。Petastorm 使用 PyArrow 来读取 Parquet 文件。

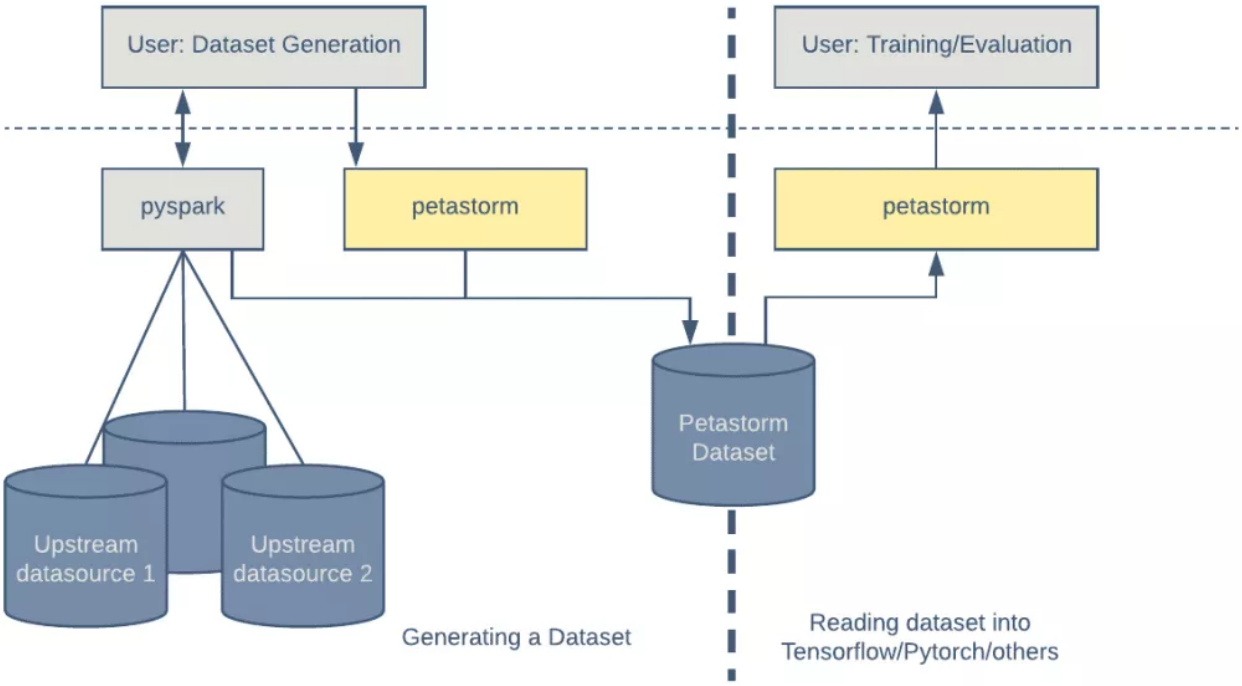

图 2：将多个数据源组合到单个表格结构中，从而生成数据集。可以多次使用相同的数据集进行模型训练和评估。

生成数据集

要使用 Petastorm 生成数据集，用户首先需要定义数据模式，也就是 Unischema。这是用户唯一需要定义模式的地方，Petastorm 会将它转换为其他框架所需的格式，例如 PySpark、Tensorflow 和 Python。

Unischema 的实例被序列化为 Parquet 存储元数据中的自定义字段，可以使用数据集的路径来读取它。

下面的示例演示了如何创建 Unischema 实例。必需的字段属性包括：字段名称、数据类型（使用 NumPy 数据类型来表示）、多维数组、用于数据编码 / 解码的 codec，以及一个表示字段是否可为空的布尔值。

```
HelloWorldSchema = Unischema('HelloWorldSchema', [
 UnischemaField('id', np.int32, (), ScalarCodec(IntegerType()), False),
 UnischemaField('image1', np.uint8, (128, 256, 3) CompressedImageCodec('png'), Fal
 UnischemaField('array_4d', np.uint8, (None, 128, 30, None), NdarrayCodec(), False
])
```

我们使用 PySpark 来写入 Petastorm 数据集。下面的示例演示了如何使用 Petastorm 创建 1000 行数据。

```
rows_count = 10
with materialize_dataset(spark, output_url, HelloWorldSchema, rowgroup_size_mb):

  rows_rdd = sc.parallelize(range(rows_count))\
      .map(row_generator)\
      .map(lambda x: dict_to_spark_row(HelloWorldSchema, x))

  spark.createDataFrame(rows_rdd, HelloWorldSchema.as_spark_schema())\
      .write \
      .parquet('file:///tmp/hello_world_dataset')
```

- materialize_dataset 上下文管理器在开始时执行必要的配置，并在最后写入 Petastorm 元数据。输出的 URL 可以指向 HDFS 或文件系统的位置。
- rowgroup_size_mb 定义了 Parquet 行组的大小（以兆字节为单位）。
- row_generator 是一个返回与 HelloWorldSchema 匹配的 Python 字典的函数。
- dict_to_spark_row 根据 HelloWorldSchema 来验证数据类型，并将字典转换为 pyspark.Row 对象。

读取数据集

接下来，我们将概述如何使用 Python 代码以及在两个常用的机器学习框架（Tensorflow 和 Pytorch）中读取数据集。

Python

在 Python 代码中，可以直接使用 Reader 实例访问 Petastorm 数据集。Reader 实现了迭代器接口，所以读取数据很简单：

```
with Reader('file:///tmp/hello_world_dataset') as reader:
  # Pure python
  for sample in reader:
      print(sample.id)
      plt.imshow(sample.image1)
```

Tensorflow

下面的示例显示了如何将数据集流式传输到 Tensorflow。examples 是一个元组，它的键来自 Unischema，而值为 tf.tensor 对象：

```
with Reader('file:///tmp/hello_world_dataset') as reader:
  tensor = tf_tensors(reader)
  with tf.Session() as sess:
      sample = sess.run(tensor)
      print(sample.id)
      plt.imshow(sample.image1)
```

在不久的将来，用户可以使用 tf.data.Dataset 接口来访问数据。

Pytorch

Petastorm 数据集可以通过适配器类 petastorm.pytorch.DataLoader 集成到 Pytorch 中，如下所示：

```
with DataLoader(Reader('file:///tmp/hello_world_dataset')) as train_loader:
  sample = next(iter(train_loader))
  print(sample['id'])
  plt.plot(sample['image1'])
```

使用 Spark 分析数据集

Spark 本身支持 Parquet 数据格式，因此可以使用各种 Spark 工具来分析和操作数据集。下面的示例演示了如何将 Petastorm 数据集读取为 Spark RDD 对象：

```
rdd = dataset_as_rdd('file:///tmp/hello_world_dataset', spark,
   [HelloWorldSchema.id, HelloWorldSchema.image1])
print(rdd.first().id)
```

标准的 PySpark 工具可用于处理 Petastorm 数据集。请注意，数据并不会被解码，而且只有在 Parquet 格式中具有相应原生表示的字段的值（例如标量）才有意义：

```
# Create a dataframe object from a parquet file
dataframe = spark.read.parquet(dataset_url)

# Show a schema
dataframe.printSchema()

# Count all
dataframe.count()

# Show a single column
dataframe.select('id').show()
```

可以使用 SQL 查询 Petastorm 数据集：

```
number_of_rows = spark.sql(
  'SELECT count(id) '
  'from parquet.`file:///tmp/hello_world_dataset`').collect()
```

Petastorm 的特性

Petastorm 提供了各种特性来支持自动驾驶算法的训练，包括行过滤、数据分片、shuffle、对字段子集的访问，以及对时间序列数据（n-gram）的支持。

典型数据集的结构包括：

- 在自动驾驶汽车测试运行期间收集的传感器数据的多个列，包括摄像头、激光定位器和雷达。
- 手动生成的标签，作为行的字段进行存储。

行数据按照时间顺序排序，并按照汽车的测试运行进行分组，行组大小通常在 30 到 100 范围内。

并行执行策略

Petastorm 提供了两种并行化数据加载和解码操作的策略：一种基于线程池，另一种基于进程池。策略的选择取决于所读取的数据类型。

通常，当行中包含编码的高分辨率图像时，应使用线程池策略。在这种情况下，大部分处理时间用于通过 C++ 代码来解码图像。这个时候不会持有 Python 全局解释器锁（GIL）。

当行很小时，使用进程池策略更合适。在这种情况下，大部分处理都通过 Python 代码来完成。这个时候必须并行运行多个进程，这样才能克服 GIL 导致的执行串行化。

n-gram

有些模型需要时间上下文，以便更好地解释环境或预测环境中参与者的未来行为。

如果底层的数据是按时间排列，Petastorm 就可以提供这样的时间上下文。如果向 Petastorm Reader 对象请求 n-gram，那么后续的行将被分组到单个训练样本中。

下图显示了长度为 3 的 n-gram 的分组。AV Log#0 和 AV Log#1 表示两种不同的车载记录：

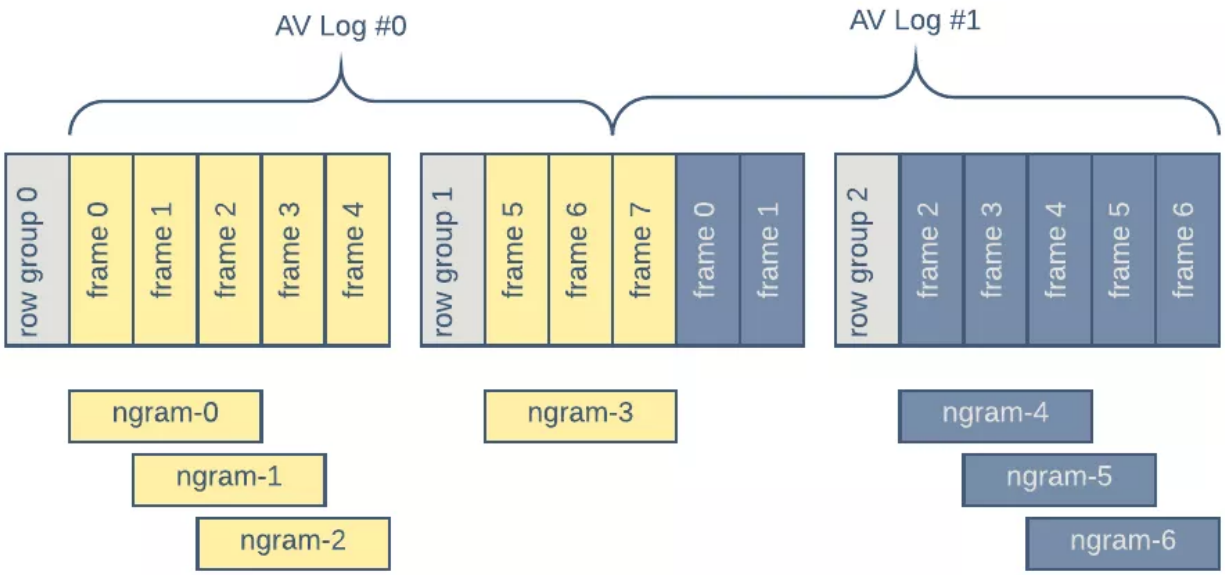

图 3：在读取数据集时构造 n-gram。n-gram 不能跨 Parquet 行组。

请注意，n-gram 分组不能跨 Parquet 行组。在图 3 中，row-group 0 生成了三个 n-gram，而 row-group 1 只生成一个，另外三个来自 row-group 2。n-gram 节省了 IO 和 CPU 带宽，因为不需要进行磁盘数据复制，也不需要进行重复加载和解码。

n-gram 按照它们在数据集中出现的顺序来生成，因此用户需要让数据集中的顺序与访问模式保持一致。

shuffle

如果数据集支持 n-gram 访问模式，那么它的行数据时按时间戳排序的。Parquet 支持加载行组中的全部行。因此，数据将被加载到高度相关的样本组中（例如，从一辆自动驾驶车辆的摄像头获取的连续两张图像将非常相似）。连续样本之间的高相关性不是我们所期望的，它们会降低训练算法的性能。为了减少相关性，Petastorm 提供了 shuffle 的功能。

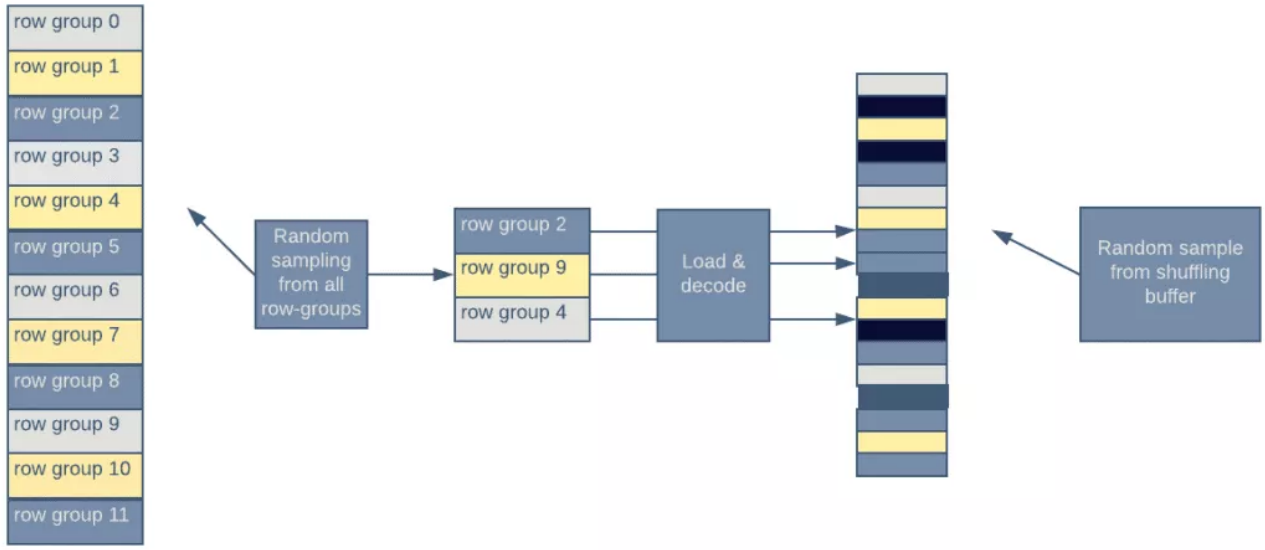

图 4：通过随机选择要加载的行组，然后将个体样本放入内存 shuffle 缓冲区来实现 shuffle。

Petastorm 从数据集中的随机选择一组行组。解码过的行被放入行 shuffle 缓冲区，然后从缓冲区中选择一个随机行返回给用户。

行谓词（过滤器）

对于在多个实验室中重用的数据集实例，能够有效地选择行子集是非常重要的。Petastorm 支持行谓词。Petastorm 行谓词利用了 Parquet 存储分区，只加载符合条件的列。

在未来，我们计划将 Parquet 的谓词下推（pushdown）功能集成到 Petastorm 中，以进一步加快查询。

行组索引

Petastorm 支持存储一个键与一组行组的映射。这种映射有助于快速查找符合特定条件的行组。在使用“行谓词”的地方，需要进行额外的过滤。

为分布式训练进行分片

在分布式训练环境中，每个进程通常负责训练数据的一个子集。一个进程的数据子集与其他进程的数据子集正交。Petastorm 支持将数据集的读时分片转换为正交的样本集。

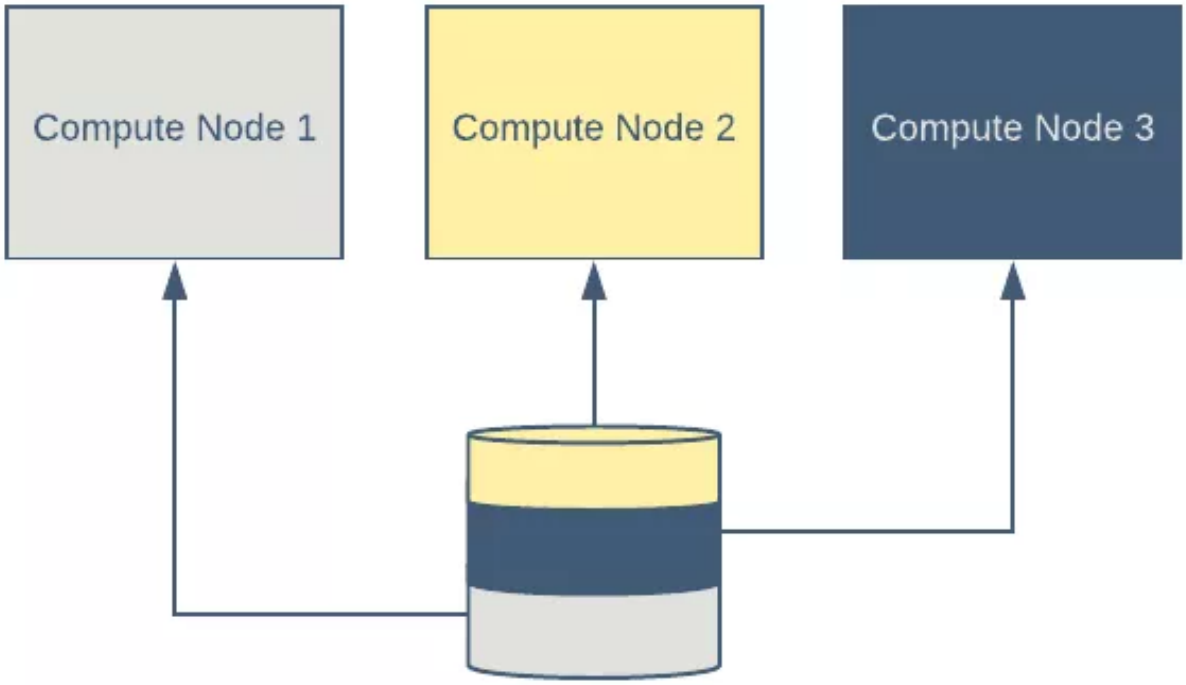

图 5：Petastorm 将数据集的非重叠子集提供给参与分布式训练的不同机器。

本地缓存

Petastorm 支持在本地存储中缓存数据。当网络连接速度较慢或带宽很昂贵时，这会派上用场。

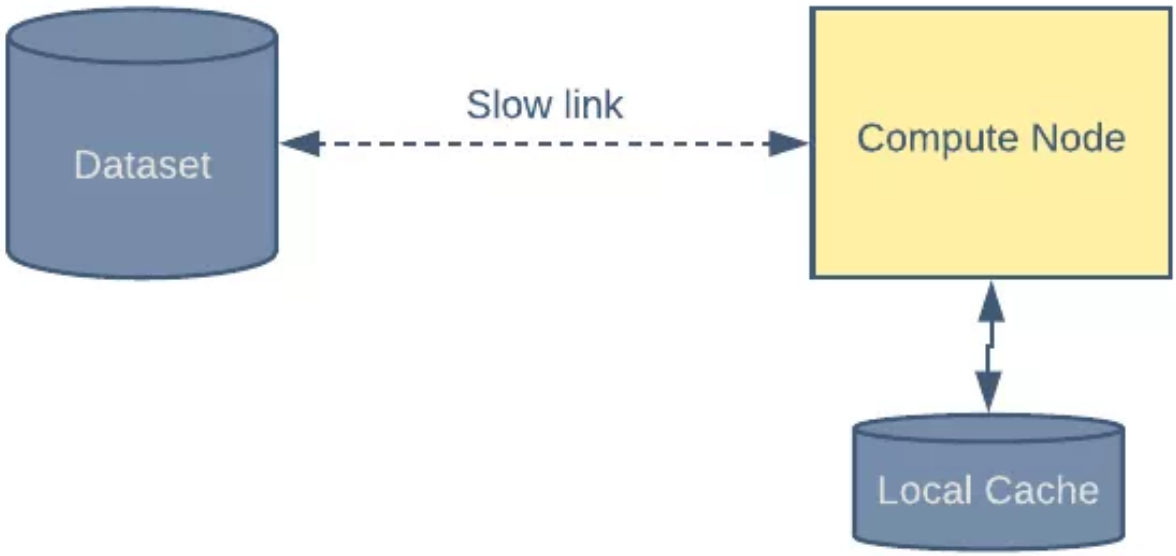

图 6：如果启用了本地缓存，每个会话仅下载一次数据。

在第一个时间段，从远程存储读取一组样本，并保存到本地缓存中。在随后的时间段，将从本地缓存中读取所有数据。

Petastorm 架构

Petastorm 的设计目标包括：

- 通过单数据模式定义进行数据的编码和解码。
- 为 ML 框架和纯 Python 代码提供可用的高数据加载带宽。
- 将 Apache Spark 作为分布式集群计算框架来生成数据集。
- 与 Python、ML 平台无关的 Petastorm 核心组件的实现。
- 呈现给 Tensorflow 和 PyTorch 框架的原生接口。
- etl 包实现了生成数据集的功能。
- Reader 是训练和计算代码使用的主要数据加载引擎。Reader 使用 Python 实现，不依赖任何 ML 框架（Tensorflow、Pytorch），并且可以通过 Python 代来实例化和使用。
- 为 Tensorflow 和 PyTorch 提供适配器。
- Unischema 可以被数据集生成和数据加载代码引用。

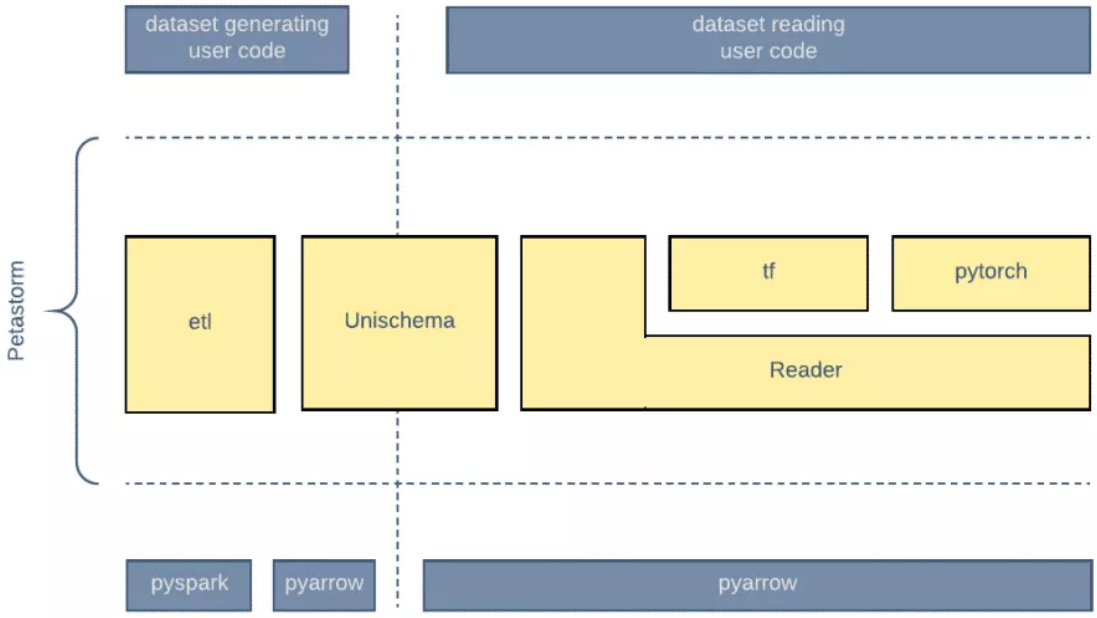

图 7：Petastorm 提供了支持数据集生成和读取的组件。Unischema 定义了可供两者使用的公共数据模式。

对 Parquet 的修复

当我们开始使用 Spark 写入 Parquet 数据集时遇到了一些麻烦。原因主要是数据的行大小，它包含了几个数兆字节的字段。我们的第一个问题是数据集中的行组比预期的要大得多，导致内存不足等问题。深入研究代码后发现，parquet-mr 在检查行组是否达到用户设置的目标大小之前，强制限制行组最少为 100 行。针对这个问题，Parquet 已经有一个相关的拉取请求，于是我们 fork 了这个代码库，并做了一些修改，让行组的大小符合我们的要求。

在解决了行组大小的问题后，我们发现，当我们尝试生成较小的行组或使用更大的字段时，Spark 作业会耗尽内存。通过深入研究生成的数据集，我们发现，在添加新字段或减少行组大小时，存储文件元数据的 Parquet 页脚大小显著增加了。

原来 Parquet 会为代表图像的巨大二进制字段或其他多维数组生成统计数据。由于 Parquet 在会页脚中保存这些字段的最小值和最大值，因此，如果行组大小足够小，那么页脚就会变得很大，直至无法全部放到内存中。这个问题在 parquet-mr 代码库中已经得到了解决，但是我们使用的是 Spark 2.1.0（依赖了 Parquet 1.8.1）。为了解决这个问题，我们升级了 Spark（Parquet 1.8.3 中已经修复了这个问题）。

下一步

下面我们重点介绍一下计划在不久的将来推出的一些改进：

减少 shuffle 的内存占用

大行组有助于提高 IO 利用率和数据加载速度。不过，它们也会增加连续样本之间的相关性。我们正在积极改进 shuffle 机制。

谓词下推支持

Pyarrow 将很快提供谓词下推支持。我们希望用它来实现更快的行过滤。

改进与 Spark 的集成

在 Spark 中访问 Petastorm 数据集时，某些操作似乎比预期花费更多的时间或内存。我们需要进一步调查 Parquet 库代码，以了解有效处理大型字段的其他细微差别。

额外的存储格式

Petastorm 抽象了底层存储格式。我们可以将 Parquet 以外的存储格式集成到 Petastorm 中，从而为实验和数据加载性能调整提供更大的自由。

GitHub 开源项目传送门：

Petastorm：

http://www.github.com/uber/petastorm

parquet-mr：

https://github.com/apache/parquet-mr

英文原文：

https://eng.uber.com/petastorm/

今日荐文

点击下方图片即可阅读

比 Hive 快 800 倍！大数据实时分析领域黑马开源 ClickHouse

推　荐

从市面上热门的 Caffe、TensorFlow、CNTK、Keras 等到腾讯最新发布的 PocketFlow，AI 框架多种多样，更有向移动端倾斜的趋势。面对企业中的不同业务线，我们该如何选择适合自己的框架？如何规避不同框架的缺点将优势最大化？

AICon 全球人工智能与机器学习技术大会上，老师木将作为出品人，与大家一起探讨 AI 工具和框架选型，以及 AI 平台搭建，企业如何基于已有大数据基础设施引入人工智能平台，有哪些难点...

大会 6 折报名倒计时 10 天，感兴趣的小伙伴识别下图二维码或点击“阅读原文”即可报名，团购更优惠，详情咨询：18514549229（同微信）。

如果你喜欢这篇文章，或希望看到更多类似优质报道，记得给我留言和点赞哦！

Read more