ElasticDL：同时提升用户体验和集群利用率

弹性调度和刚性调度

- 刚性调度（gang scheduling）：**一个作业里的 n 个进程，要么都运行，要么都死掉。**
  - Google 开源的 Kubeflow 是一个 Kubernetes 扩展 operators，支持在 Kubernetes 上分布式地运行 TensorFlow 作业。因为 TensorFlow runtime 目前支持一定程度的容错，所以作业执行过程中，如果有一些 workers 挂了，剩下的可以继续。不过不支持因为日后资源富余，恢复 workers 数量。
  - Facebook 开源的 PyTorch Elastic 也是类似的扩展，支持分布式 PyTorch 作业。号称 Elastic，其实是 job 失败后从 checkpoint 重启。
  - XGBoost、MXNet 社区也习惯于复用 Kubeflow。用 MPI 写的程序也可以利用 Kubeflow 拉起。

  以上 Kubernetes operators 都可以在蚂蚁金服的 ASI（旧称 Sigma）上部署。

  Gang scheduling 的特点是：运行时如果有一个进程挂了（比如被更高优先级的作业抢占了资源），则整个作业挂掉。等资源足够再启动所有的 n 个进程了，则可以重启（或者从最近的 checkpoint 恢复）。

- 弹性调度（elastic scheduling）：**训练作业运行过程中，进程数量的变化不影响作业。**
  - 运行过程中，一个或者几个进程被高优先级的作业抢占，剩下的进程不受影响地继续进行。如果将来资源丰沛了，系统可以加几个进程，此时作业仍然不受影响地继续运行。
  - ElasticDL 可以启动和调度分布式 TensorFlow、PyTorch 作业。也很容易支持分布式 XGBoost 作业。

ElasticDL 和其他方案的对比

AI 作业的 elastic scheduling 需要了解 AI 作业的模型，所以更适合实现在 AI 训练作业专用的框架里，而不是通用的 Kubernetes operators 里。为此我们写了 ElasticDL 框架。目前公司内部 Kubemaker 项目部署的 Kubernetes operators 支持刚性调度的分布式 AI 作业。

	Gang scheduling	Elastic scheduling
TensorFlow	Kubeflow	ElasticDL
PyTorch	PyTorch Elastic	ElasticDL
XGBoost	Distributed XGBoost	ElasticDL

Elastic scheduling 的实现可以带来用户体验和集群利用率的双丰收，而一般的技术都是在两个优点之间做权衡（compromise）。我们做了两组实验来说明 ElasticDL 的优点。

实验一：多个 AI 训练作业

考虑两个 AI 训练作业需要的资源总和略超过集群的情况：

- 如果没有 elastic scheduling，则两个作业顺序执行。第二个作业的发起人需要等很久 —— 用户体验不好。并且任何时刻只有一个作业在运行 —— 集群资源用不满。
- 如果有 elastic scheduling，则两个作业并发执行，虽然后启动的作业拿不到期待的全部资源，但是也马上就开始执行了 —— 用户体验好。因为两个作业并发 —— 集群被用满。

我们做了一个实验来验证上述好处。这个实验可以在 ASI 集群和开源 Kubernetes 集群上复现。实验结果如下图。

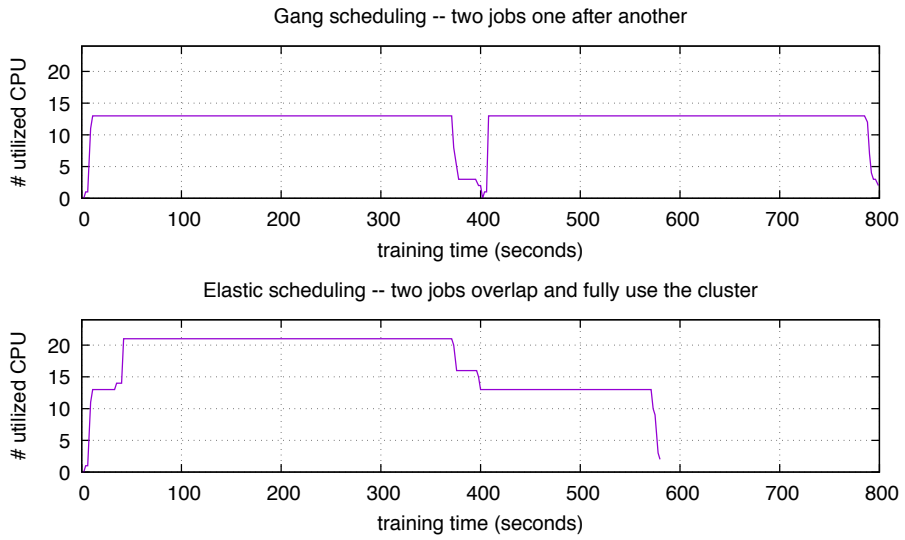

Figure 1: overlap jobs

上图对应的实验里，我们用 gang scheduling 的方式提交了两个训练作业，每个作业都需要 13 个 CPU。而 Google Cloud 上租用的实验集群总 CPU 数是 24，不足同时运行两个作业，所以依次运行它们。可以看到第一个作业在 395 秒时结束。随后集群花了一点时间调度，然后开始运行第二个作业，直到 795 秒时结束。

下图对应的实验里，我们用 ElasticDL 来执行同样的两个训练作业。第一个作业提交之后的 30 秒，我们提交了第二个作业。第二个作业马上就开始运行，用满了集群剩下的资源，而不需要等到第一个作业结束。在 395 秒时，第一个作业结束。随后，在 580 秒时，第二个作业也结束了。因为弹性调度，使得两个作业尽量同时运行，所以总结束时间比也上图要早。

总结：

- 用户等待作业启动时间几乎是 0。这对于 AI 很重要，因为用户最关注的是第一个迭代尽快开始 —— 如果第一个迭代 fail 了，很可能是用户程序的 bug。
- 集群利用率高。第二个实验（elastic scheduling）执行期间，有一段时间集群利用率是 100%；其他时间也不低于第一个实验（gang scheduling）。
- 作业完成更快。第二个试验里，两个作业用了约 580 秒；第一个实验里需要约 795 秒。

实验二：AI 作业和在线服务混布

运行各种在线服务的生产集群，通常需要留出余量资源，以应付突然增长的用户请求量。我们希望利用这些“余量”来做 AI 训练，从而提升集群利用率。下面实验验证：通过用较低优先级运行 ElasticDL 训练作业，在用户请求增加的时候，Kubernetes 自动扩容在线服务（NGINX）；此时 ElasticDL 作业自动释放资源，配合在线服务的扩容。当流量高峰过去之后，Kubernetes 自动缩容 NGINX 服务，此时，ElasticDL 自动利用释放的资源。

图中紫色曲线是 NGINX 服务使用的 CPU 数量，随用户请求数量变化。绿色曲线是 ElasticDL 训练作业使用的 CPU 数量，随 NGINX 的资源需求自动变化。蓝色曲线是机群的总体资源利用率 —— 保持在 90% 以上。

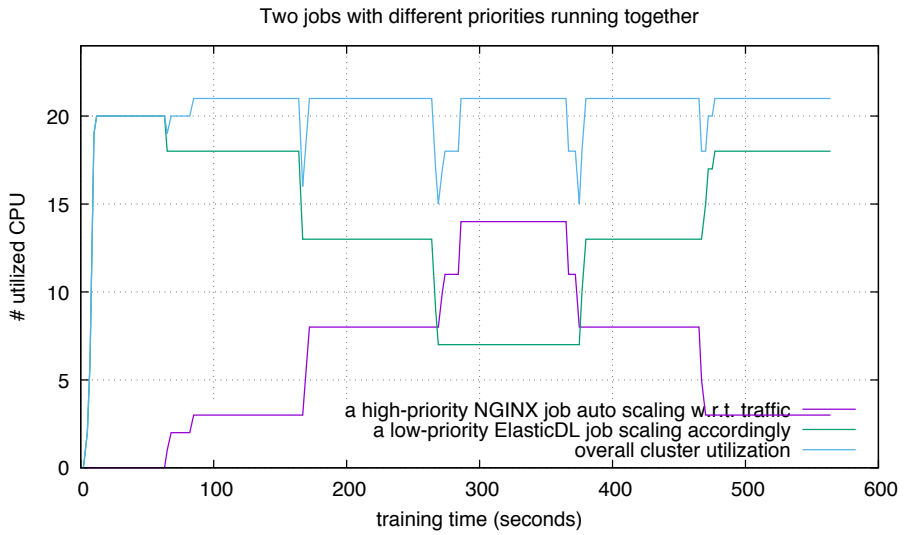

Figure 2: auto react

实验三：训练时更改 worker 数量不影响收敛性

有用户担心训练过程中 worker 的数量发生变化，会导致不收敛。实际情况下从未发生这类问题。用 ElasticDL 和用 gang scheduling 分别训练 Wide & Deep model 和 xDeepFM model，收敛曲线如下：

可以看到，采用 gang scheduling 持续用 4 个或者 8 个 workers，和用 ElasticDL 并且 worker 数量在 4 到 8 之间变化，得到的收敛曲线很难分辨。差别在自然误差范围之内。

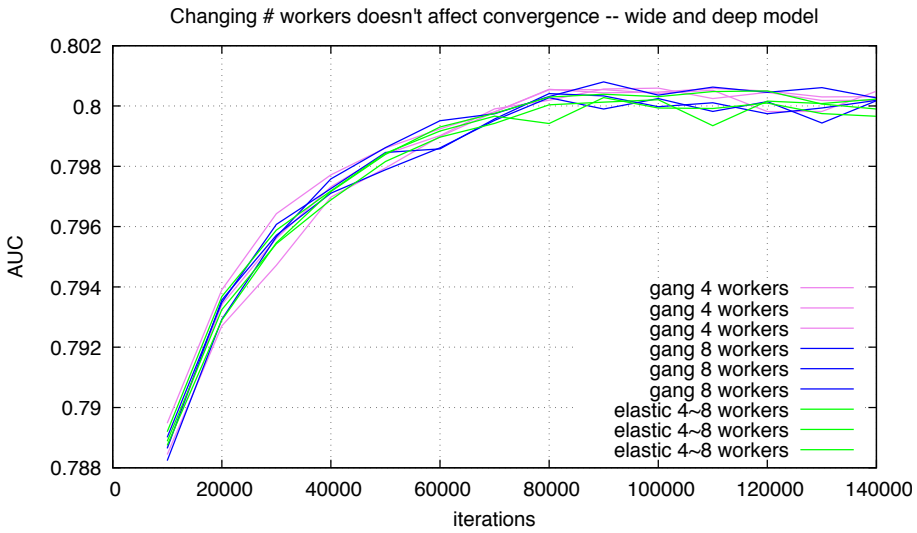

Figure 3: wide-n-deep training coverges

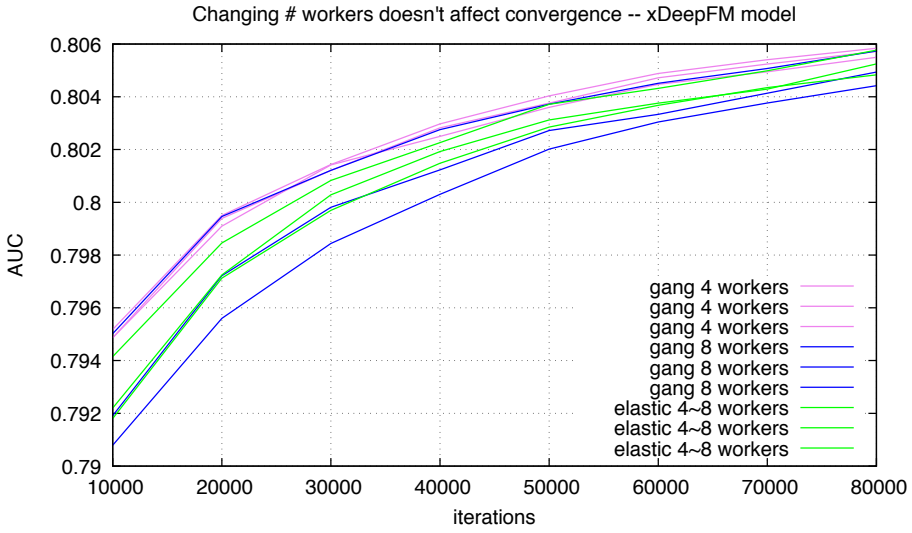

Figure 4: xdeepfm training converges