Function	Idap	Date LGTM
Author(s)	**Xu Yan, Software Engineer, Baidu Research, yanxu05@baidu.com Helin Wang, Software Engineer, Baidu U.S.A., helinwang@baidu.com Yi Wu, Software Engineer, Baidu Research, wuyi05@baidu.com Xi Chen, Software Engineer, Baidu U.S.A., chenxi44@baidu.com Weibao Gong, Software Engineer, Baidu Research, gongweibao@baidu.com Xiang Li, Tech Lead Manager, CoreOS, xiangli.cs@gmail.com Yi Wang, Tech Lead of PaddlePaddle, Baidu U.S.A., yi.wang.2005@baidu.com**	
CNCF (mktg/pr)	**Kaitlyn/Sarah C.**	**SC reviewed 11/30**
K8s-docs Review	**Zach Corleissen**	**ZC edited 12/2**
PM Review	**PM-group**	
Review & Publish	**hrdinsky or sarahnovotny**	

Blog Title: PaddlePaddle Fluid: Elastic Deep Learning on Kubernetes

Authors: Xu Yan, Software Engineer, Baidu Research, yanxu05@baidu.com
Helin Wang, Software Engineer, Baidu U.S.A., helinwang@baidu.com
Yi Wu, Software Engineer, Baidu Research, wuyi05@baidu.com
Xi Chen, Software Engineer, Baidu U.S.A., chenxi44@baidu.com
Weibao Gong, Software Engineer, Baidu Research, gongweibao@baidu.com
Xiang Li, Tech Lead Manager, CoreOS, xiangli.cs@gmail.com
Yi Wang, Tech Lead of PaddlePaddle, Baidu U.S.A., yi.wang.2005@baidu.com

Date/Time to publish:

Staging:

Twitter: @Kubernetesio:

Editor's note: Today's post is a joint post from the deep learning team at Baidu and the etcd team at CoreOS.

PaddlePaddle Fluid: Elastic Deep Learning on Kubernetes

Two open source communities—PaddlePaddle, the deep learning framework originated in Baidu, and Kubernetes®, the most famous containerized application scheduler—are announcing the Elastic Deep Learning (EDL) feature in PaddlePaddle's new release codenamed Fluid.

Fluid EDL includes a [Kubernetes controller](https://github.com/kubernetes/community/blob/master/contributors/devel/controllers.md), [*PaddlePaddle auto-scaler*](https://github.com/PaddlePaddle/cloud/tree/develop/doc/autoscale), which changes the number of processes of distributed jobs according to the idle hardware resource in the cluster, and a new fault-tolerable architecture as described in the [PaddlePaddle design doc](https://github.com/PaddlePaddle/Paddle/blob/develop/doc/design/cluster_train/README.md).

Industrial deep learning requires significant computation power. Research labs and companies often build GPU clusters managed by SLURM, MPI, or SGE. These clusters either run a submitted job if it requires less than the idle resource, or pend the job for an unpredictably long time. This approach has its drawbacks: in an example with 99 available nodes and a submitted job that requires 100, the job has to wait without using any of the available nodes. Fluid works with Kubernetes to power elastic deep learning jobs, which often lack optimal resources, by helping to expose potential algorithmic problems as early as possible.

Another challenge is that industrial users tend to run deep learning jobs as a subset stage of the complete data pipeline, including the web server and log collector. Such general-purpose clusters require priority-based elastic scheduling. This makes it possible to run more processes in the web server job and less in deep learning during periods of high web traffic, then prioritize deep learning when web traffic is low. Fluid talks to Kubernetes' API server to understand the global picture and orchestrate the number of processes affiliated with various jobs.

In both scenarios, PaddlePaddle jobs are tolerant to a process spikes and decreases. We achieved this by implementing the new design, which introduces a master process in addition to the old PaddlePaddle architecture as described in a [previous blog post](http://blog.kubernetes.io/2017/02/run-deep-learning-with-paddlepaddle-on-kubernetes.html). In the new design, as long as there are three processes left in a job, it continues. In extreme cases where all processes are killed, the job can be restored and resume.

We tested Fluid EDL for two use cases: 1) the Kubernetes cluster runs only PaddlePaddle jobs; and 2) the cluster runs PaddlePaddle and Nginx jobs.

In the first test, we started up to 20 PaddlePaddle jobs one by one with a 10-second interval. Each job has 60 trainers and 10 parameter server processes, and will last for hours. We repeated the experiment 20 times: 10 with FluidEDL turned off and 10 with FluidEDL turned on. In Figure one, solid lines correspond to the first 10 experiments and dotted lines the rest. In the upper part of the figure, we see that the number of pending jobs increments monotonically without EDL. However, when EDL is turned on, resources are evenly distributed to all jobs. Fluid EDL kills some existing processes to make room for new jobs and jobs coming in at a later point in time. In both cases, the cluster is equally utilized (see lower part of figure).

![Figure 1](https://raw.githubusercontent.com/PaddlePaddle/cloud/develop/doc/autoscale/experiment/result/case1.png "Figure 1. Fluid EDL evenly distributes resource among jobs.")

In the second test, each experiment ran 400 Nginx pods, which has higher priority than the six PaddlePaddle jobs. Initially, each PaddlePaddle job had 15 trainers and 10 parameter servers. We killed 100 Nginx pods every 90 seconds until 100 left, and then we started to increase the number of Nginx jobs by 100 every 90 seconds. The upper part of Figure 2 shows this process. The middle of the diagram shows that Fluid EDL automatically started some PaddlePaddle processes by decreasing Nginx pods, and killed PaddlePaddle processes by increasing Nginx pods later on. As a result, the cluster maintains around 90% utilization as shown in the bottom of the figure. When Fluid EDL was turned off, there were no PaddlePaddle processes autoincrement, and the utilization fluctuated with the varying number of Nginx pods.

![Figure 2](https://raw.githubusercontent.com/PaddlePaddle/cloud/develop/doc/autoscale/experiment/result/case2.png "Figure 2. Fluid changes PaddlePaddle processes with the change of Nginx processes.")

We continue to work on FluidEDL and welcome comments and contributions. Visit the [PaddlePaddle repo](https://github.com/PaddlePaddle/cloud), where you can find the [design doc](https://github.com/PaddlePaddle/cloud/blob/develop/doc/autoscale/README.md), a [simple tutorial](https://github.com/PaddlePaddle/cloud/blob/develop/doc/autoscale/example/autoscale.md), and [experiment details](https://github.com/PaddlePaddle/cloud/tree/develop/doc/autoscale/experiment).

- Xu Yan (Baidu Research)
- Helin Wang (Baidu Research)
- Yi Wu (Baidu Research)
- Xi Chen (Baidu Research)
- Weibao Gong (Baidu Research)

- Xiang Li (CoreOS)
- Yi Wang (Baidu Research)