Device Placement Optimization with Reinforcement Learning

Azalia Mirhoseini et al. (Google, ICML '17)

Presented by: Stella Lau

21 November 2017

Motivation

Problem

Neural networks are large $\Rightarrow$ heterogeneous environment
Which operations go on which CPUs/GPUs?

Motivation

Problem

Neural networks are large $\Rightarrow$ heterogeneous environment
Which operations go on which CPUs/GPUs?

Solution

Expert manually specifies device placement?

- It's manual...

Use reinforcement learning

Contributions

A reinforcement learning approach for device placement optimization in TensorFlow graphs.

- Manually assigning variables and operations in a distributed TensorFlow environment is annoying
- `https://github.com/tensorflow/tensorflow/issues/2126`
- Reward signal: execution time

Auto device placement for distributed runtime #2126

Open **windreamer** opened this issue on Apr 27, 2016 · 27 comments

windreamer commented on Apr 27, 2016

In a distributed TF setting, we need to place variables and ops to different devices. This is annoying to manually assign each variable and op, especially when we have GPU resources in our environment.

TF offer a context named `tf.train.replica_device_setter` which place variables to ps devices in round-robin manner, and this is helpful but not enough. **@mrry** can you shed some light on the auto distributed devices placement problem?

windreamer commented on Apr 27, 2016

@bhack this is the best I can do for now, pls feel free to correct me and provide some more thoughts. thx.

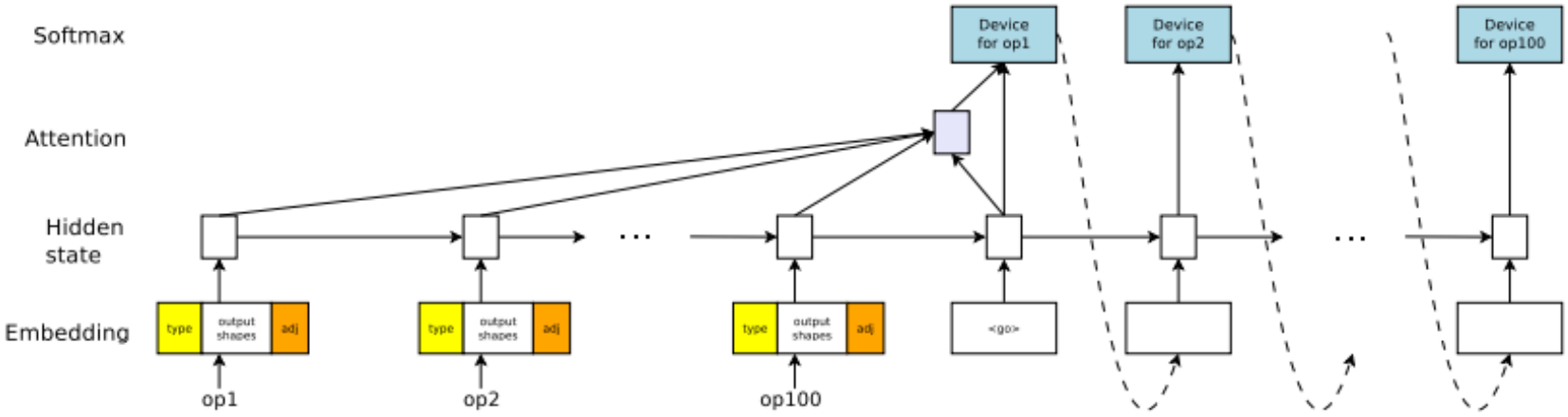
bhack commented on Apr 27, 2016 • edited

Thank you **@windreamer**. I also find this topic quite crucial for the "user experience" when we have distributed devices. Generally on cluster we could have really unbalanced GPU/CPU scalar resources ratio and there are different considerations to do for data parallel and model parallel approaches. So it is really interesting to have some feedbacks. /cc **@mtamburrano** **@lenlen**

Device placement optimization

- TensorFlow graph $\mathcal{G}$:
  M operations $\{o_1, \ldots, o_M\}$, list of D devices
- Placement $\mathcal{P}$: assign each operation o_i to a device $p_i \in D$
- $r(\mathcal{P})$: execution time of placement
- Device placement optimization: find $\mathcal{P}$ such that $r(\mathcal{P})$ is minimized

Architecture overview



Sequence-to-sequence model with LSTM and a content-based attention mechanism

1. Encoder RNN:
   - input = op_i embedded in (type, output shape, adj)
2. Decoder RNN: attentional LSTM with fixed number of time steps equal to number of operations
   - Decoder outputs device for operation at same encoder step
   - Each device has own tunable embedding, fed to next step

Challenges overview

1. Training with *noisy* policy gradients
2. Thousands of operations in TensorFlow graphs
3. Long training time

Challenge I: Training with noisy policy gradients

Problem

1. Noisy $r(\mathcal{P})$ especially at start (bad placements)
2. Placements converge $\Rightarrow$ indistinguishable training signals

Challenge I: Training with noisy policy gradients

Problem

1. Noisy $r(\mathcal{P})$ especially at start (bad placements)
2. Placements converge $\Rightarrow$ indistinguishable training signals

Solution

- Empirical finding: use $R(\mathcal{P}) = \sqrt{r(\mathcal{P}}$
- Stochastic policy $\pi(\mathcal{P}|\mathcal{G};\theta)$: minimize $J(\theta) = \mathbf{E}_{P\sim\pi(\mathcal{P}|\mathcal{G};\theta)}[R(\mathcal{P})|\mathcal{G}]$
- Train with policy gradients: reduce variance with baseline
- $\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}(R(\mathcal{P}_i) - B)\cdot\nabla_\theta \log p(\mathcal{P}_i|\mathcal{G};\theta)$
- Some placements fail to execute $\Rightarrow$ specify failing signal
- Some placements randomly fail: bad at end $\Rightarrow$ after 5000 steps, update parameters only if placement executes

Challenge II: Thousands of operations in TensorFlow graphs

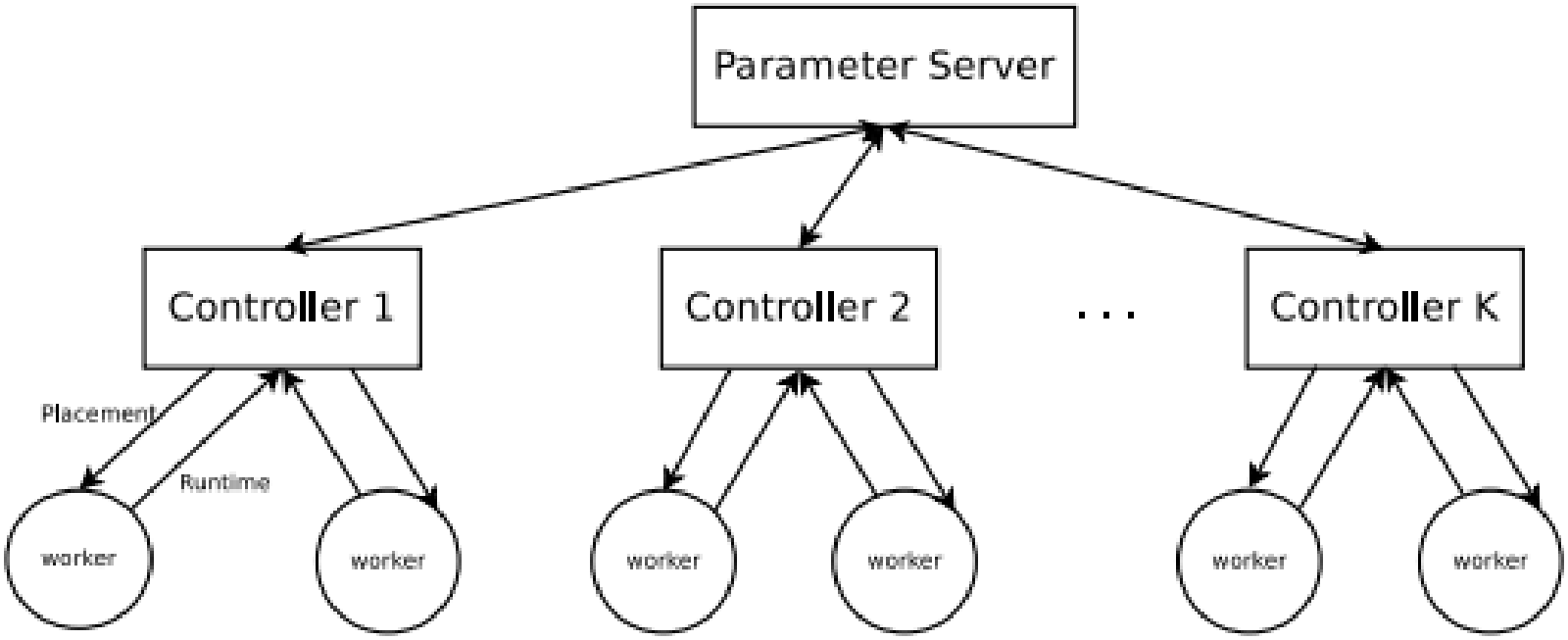
Model	#operations	#groups
RNNLM	8943	188
NMT	22097	280
Inception-V3	31180	83

Co-location groups: manually force several operations to be on the same device
Heuristics:

1. Default TensorFlow co-location groups: co-locate each operation's outputs with its gradients
2. If output of op_X is consumed only by op_Y, co-locate X and Y (recursive procedure, especially useful for initialization)
3. Model-specific rules: e.g. with RNN models, treat each LSTM cell as a group

Challenge III: Long training time

Use asynchronous distributed training to speed up training



Challenge III: Long training time

Use asynchronous distributed training to speed up training

- K workers per controller, K is number of placement samples
- Phase I: workers receive signal to wait for placements, controller receives signal to sample K placements
- Phase II: Worker executes placement, measures run time. Executed for 10 steps, average run time except first
- 20 controller, with 4-8 workers $\Rightarrow$ 12-27 hours training
- More workers = more accurate estimates, more idle workers
- Each controller has own baseline

Benchmarks: three models

1. RNNLM: Recurrent Neural Network Language Model
   - grid structure; very parallelisable
2. NMT: Neural Machine Translation
   - LSTM layer, softmax layer, attention layer
3. Inception-V3: imagine recognition and visual feature extraction;
   - multiple blocks; branches of convolutional and pooling layers; more restricted parallelisation

Pre-processed with co-location groups

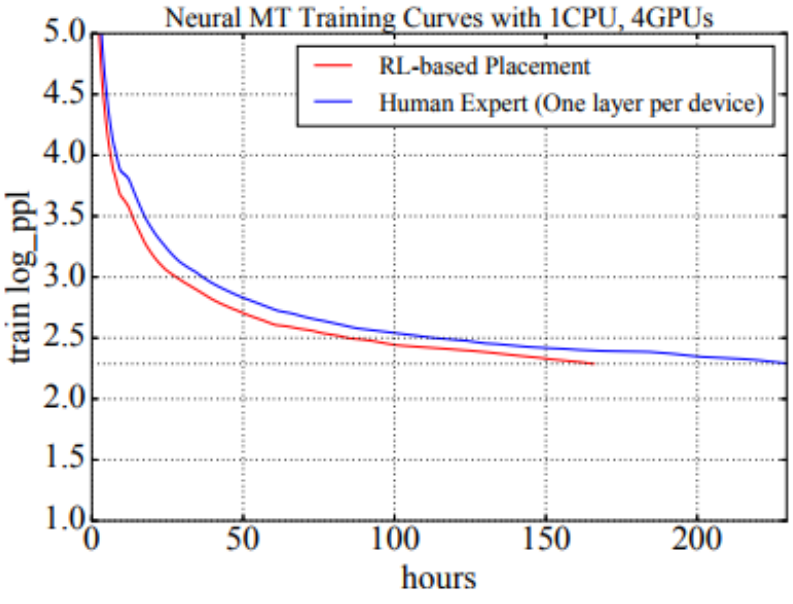
Single step run times

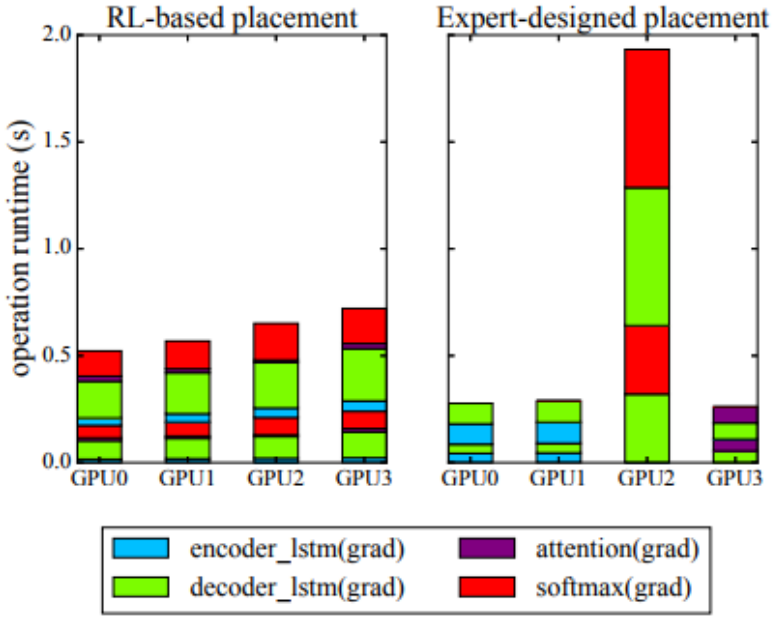
Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**	2	25.24	22.88	11.22	**4.60**	0.0%
			4	23.41	24.52	10.65	**3.85**	19.0%

- RNNLM: fit entire graph into one GPU to reduce inter-device communication latencies
- NMT: non-trivial placement. Use 4 GPUs, put less computationally expensive operations on CPU
- Inception-V3: use 4 GPUs; baselines assign all operations to a single GPU

Other contributions

- Reduced training time to reach the same level of accuracy
- Analysis of reinforcement learning based placements versus expert placements
  - NMT: RL approach balances workload better
  - Inception-V3: less balanced because less room for parallelism





Related work

- Neural networks and reinforcement learning for combinatorial optimization
  - Novelty: large-scale applications with noisy rewards
- Reinforcement learning to optimize system performance
- Graph partitioning
  - Graph partitioning algorithms are only heuristics: cost models need to be constructed (hard to estimate, not accurate)
  - Scotch optimizer: balance tasks among set of connected nodes, reducing communication costs

Summary and comments

A reinforcement learning approach to device placement optimization in TensorFlow

Questions?

- Only execution time is used as a metric. What about memory?
- Device placement optimization is still time consuming (20 hours with 80 GPUs?)
- Limited detail on training procedure and architecture
- Limited discussion on directions for future work