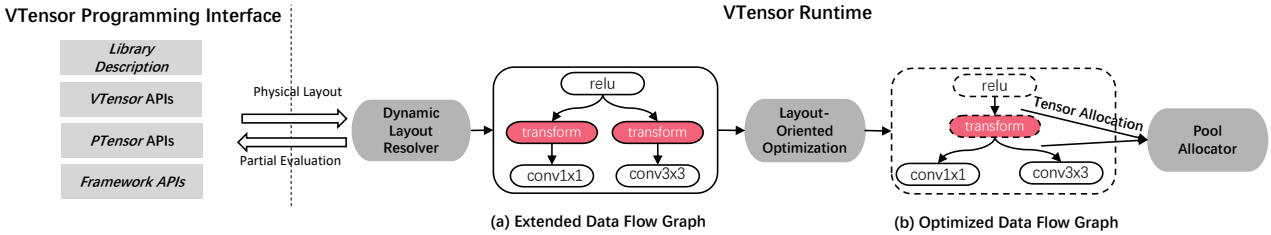

Figure 2: The VTensor framework.

four categories of programming interfaces to define an operation, describe a library, and illustrate how to invoke a library routine.

- *VTensor API* enables developers to implement an operator by writing the "Compute" function in order to access the virtual tensors.
- *PTensor API* allows developers to declare the corresponding physical tensors for virtual tensors.
- *Library Description* is required for each library to describe the layout mapping, the layout transformation handler, and some guidelines for selecting a layout.
- *Framework APIs* are provided for developers to register some handlers with the VTensor framework.

To resolve the layout for virtual tensors, we propose a "Dynamic Layout Resolver", which partially evaluates the dataflow graph to determine the physical layout for each node in the graph. Then it determines the locations required for layout transformations, and inserts the corresponding transformation operations as individual nodes into the dataflow graph, called the *extended dataflow graph*. Afterwards, we apply a pattern-based graph optimization to the extended dataflow graph to globally minimize the number of layout transformation nodes. Meanwhile, for efficient execution of the extended dataflow graph, VTensor uses offline profiling and a straggler-aware scheduling algorithm to globally allocate hardware resources to the computation nodes.

Finally, the VTensor resolver makes it possible to determine the number of temporary tensors in advance, so that we can use a centralized allocation scheme for all input/output tensors of each operation in order to avoid frequent fragmented memory allocation.

3 EVALUATION

To evaluate VTensor, we have implemented it in TensorFlow 1.14 and measured the inference latency time on the Intel Xeon E7-4820. For performance evaluation, we follow [3] to set TensorFlow's environment variables and parameters.

Compared with TensorFlow, as shown in Figure 3, VTensor has achieved a significant reduction in code sizes, from 9.95% to 70.97%, with an average of 46.6%. Since the layout-dependent codes in TensorFlow are extremely ad-hoc, developers have to explicitly write different library wrappers for layout selection and transformation. Furthermore, developers are required to write the *Compute* function for each library. In comparison, VTensor automatically inserts appropriate layout transformations when necessary, and shares the same *Compute* function for all library.

In comparison, VTensor achieves a performance improvement ranging from 10.82% to 47.96%, with an average of 27.5% compared

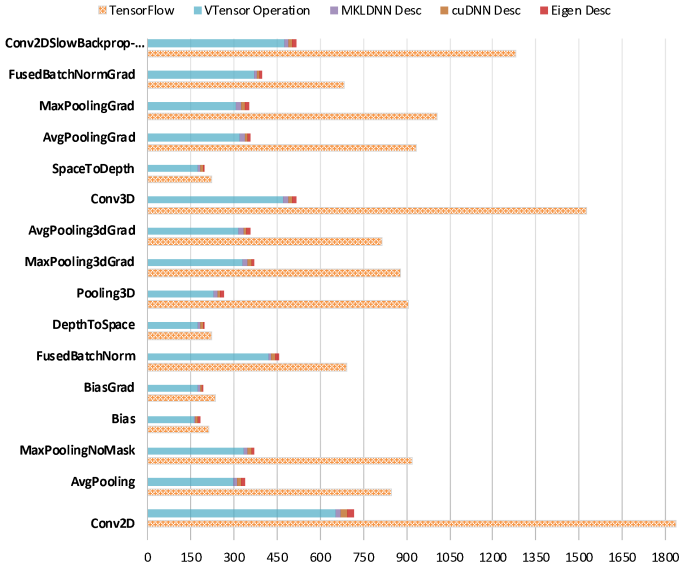

Figure 3: Comparison of LOC when writing an operator using VTensor/TensorFlow framework.

with TensorFlow. For networks with a large number of branches, VTensor can leverage layout-oriented optimizations to reduce the number of layout transformations and the straggler-aware algorithm can find more opportunities to exploit inter-op parallelism.

ACKNOWLEDGMENTS

This work is supported in part by the National Key R&D Program of China (2017YFB0202901), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030101), the National Natural Science Foundation of China (61802368, 61521092, 61432016, 61432018,61332009, 61702485, and 61872043), CCF-Tencent Open Research Fund, and an Australian Research Council grants (DP170103956 and DP180104069).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Andrew Anderson and David Gregg. 2018. Optimal DNN primitive selection with partitioned boolean quadratic programming. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. 340–351.

[3] Niranjan Hasabnis. 2018. Auto-tuning tensorflow threading model for CPU backend. In *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*. IEEE, 14–25.