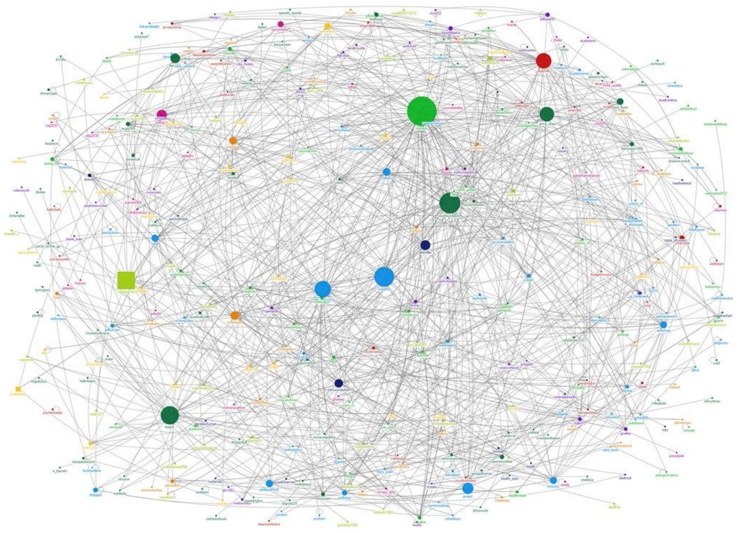
Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads

Presented by: Bradford Gill

Graph Neural Networks (GNNs)

- Analysis of graph structured data
- Applications include: drug discovery, chemistry, program analysis, recommender systems, and more
- GNN types include: GCN, GAT, GTN
- Popular topic of research in academics



Source: https://www.flickr.com/photos/notbrucelee/7219422352

Computing GNNs

- GPUs are currently the most common, however, are very expensive.
- GPUs also have limited scalability due to memory limitations
- Graph sampling improves scalability by not including all connects, however, reduces accuracy, has significant overhead, and does not guarantee convergence
- CPUs can be more cost-effective but suffer from slower speeds

Serverless computing

- Actually uses servers, but functions are not tied to one server
- Simple functions that are allocated on demand
- Very easy to scale
- Challenges include low compute power and high networking latency
- Very affordable

Azure Functions

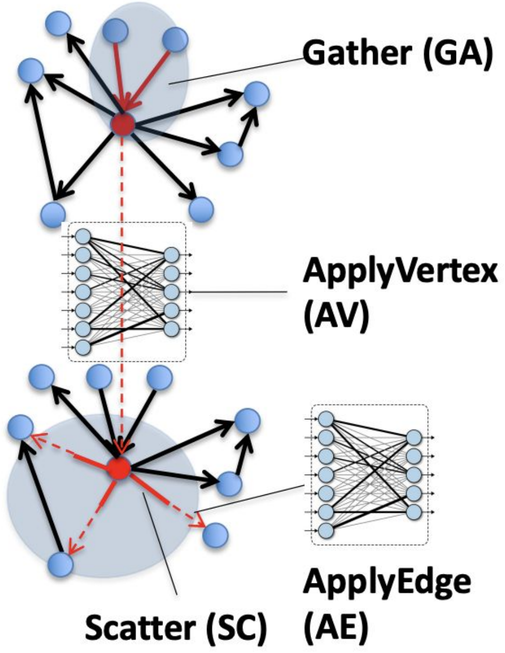
Google Cloud Functions

Dorylus

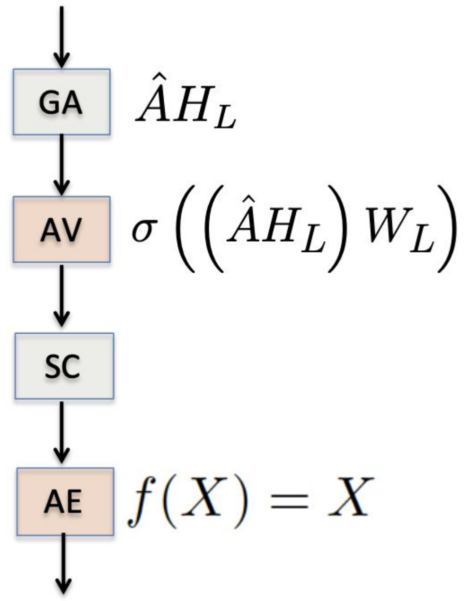
- Key takeaway: Lowers cost and enables larger graph computations without sampling
- Uses cheap CPU servers and AWS Lambda serverless threads to achieve inexpensive, efficient, and scalable GNN framework
- Two main challenges:
    - Overcomes low computation capabilities of Lambda
        - Split computations into two categories: Tensor computation and Graph computation
        - Graph computations take advantage of SIMD processes and are executed on CPU servers
        - Tensor computations are done on Lambda Threads and operate on node information. Fits well into Lambdas computational profile
    - Overcomes Lambdas High latency
        - Deploy bounded pipeline asynchronous computation
        - Hide latency behind other tasks
        - Stash weights on parameter server CPUs (PSs)

Overview of GNN process

- Gather: GA
- Apply vertex: AV
- Scatter: SC
- Apply Edge: AE
- Weight Update: WU (not shown)



(a) Computation model



(b) Annotated dataflow graph

Design overview

Three architecture components:

- EC2 graph server (GS)
  - Hosts graph partitions and execute GA and SC
  - Communicate with lambdas
- EC2 parameter server (PS)
  - Host weights for Lambdas
- Lambdas:
  - Tensor operations: basic linear algebra operations.
  - Execute AV and AE

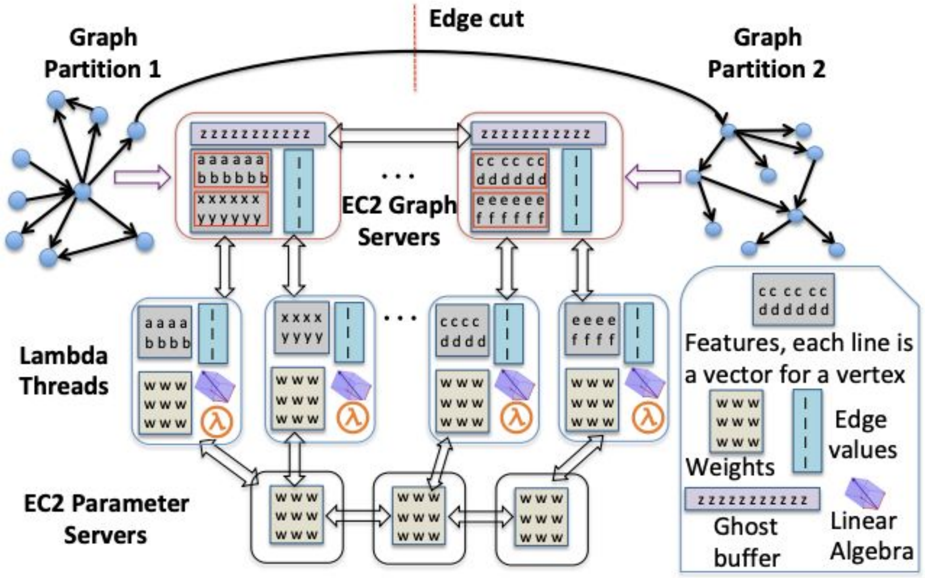
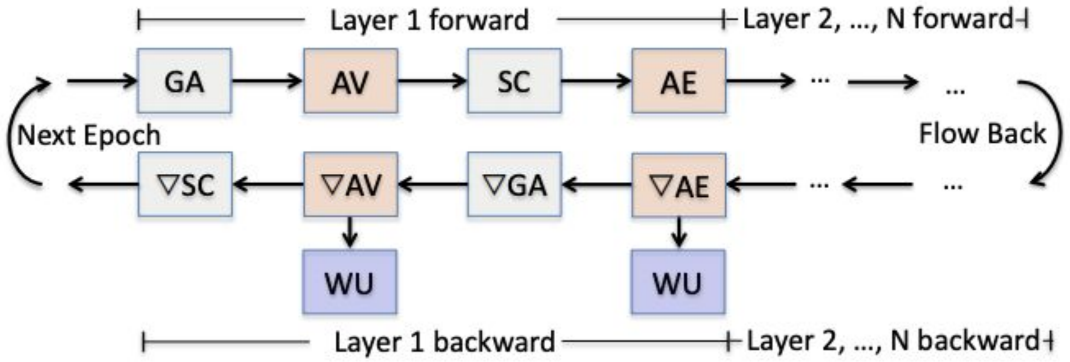
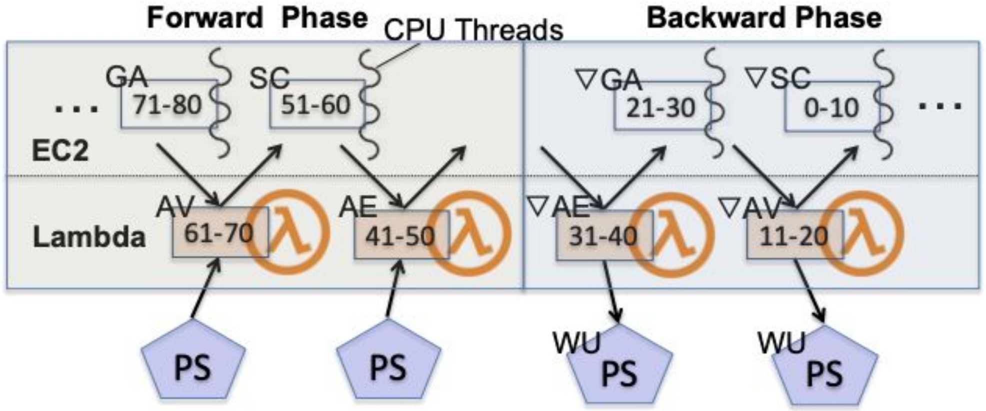


Figure 2: Dorylus's architecture.

Forward and backwards pass visualized

- Grey: executed on GSs
- Orange: executed on lambdas
- Purple: executed on PSs





Pipe lining

- Break up vertices each graph partition into mini batches called intervals
- Balance computation by assigning each interval the same number of vertexes and similar number of inter interval edges
  - Tensor computation size depends on number of vertexes and edges
  - Graph computation size depends on number of edges
- Each interval is processed by a task. GSs maintain a task Queue and enqueue tasks when the input is ready
- Lambda latency is hidden by overlapping graph and tensor computations

Bounded asyncrony

- Lambda execution time is not predictable, so the show must go on asynchronous
- asynchrony lowers performance
- Two points of pipeline are synchronous: weight updates and vertex activations at GA
- A balance between using outdated values and sacrificing speed must be struck.

Weight Stashing

- Similar to PipeDream
- Uses available weights at forward pass and caches those weights for the backward pass
- Weight stashing is distributed and duplicated across many PSs. This is only feasible because GNNs typically have few layers and weights

Bounded async for gather

- Allows tasks to proceed using stale values that are within S values of being current
- Updates happen across epochs rather than per epoch
- Bounded stale values by S guarantees convergence, this is not the case with unbounded staleness
- In essence: allows computation to continue unless values are overly stale and useless for computation

Lambda management

- Each GS runs and manages a lambda controller
- Each lambda
  - Uses openBLAS and AVX to computer tensor operations
  - Communicates with PSs and GSs through ZeroMQ
  - Runs on virtual private cloud to maximize bandwidth with EC2 instances
  - Faster when already running “Warm”

Lambda optimization

- The significant challenge is network bandwidth
- Three main optimizations:
  - Task fusion: the last layer of AV and ∇AV are directly connected so they are fused to avoid communication
  - Tensor rematerialization: sometimes it is more efficient to recalculate a value rather than cache and retrieve values. Cuts down on network communication.
  - Internal Lambda Streaming: Retrieve half the data, Simultaneously compute the first half and receive the second half. Hides some communication

Autotuning number of Lambdas

- Saturate lambdas appropriately
    - Too few lambdas and there is not enough parallelism to hide the latency.
    - Too many lambdas and there is pointless spending since latency is already hidden
- Optimal number of lambdas depends on network conditions and graph structure that is hard to predict before execution so autotuning is used

Algorithim:

# lambdas = min(# intervals, 100) initially

If the lambda Queue grows significantly, inputs are coming in too fast so decrease the number of lambdas

If the lambda Queue shrinks significantly, inputs are coming in too slow so decrease the number of lambdas

Asyncrony

Important context: S=0 and S=1 are faster per epoch than pipe.

Overall S0 is faster to convergence by 1.234 and 1.233 over pipe and S=0 respectively

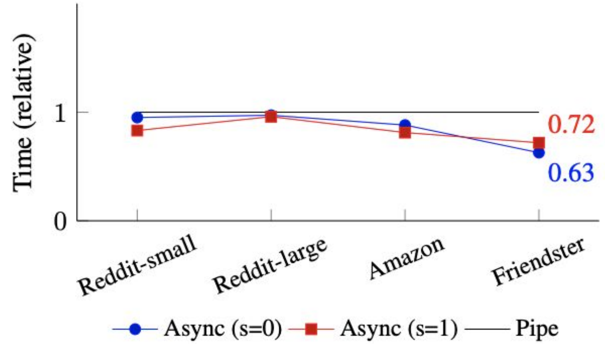
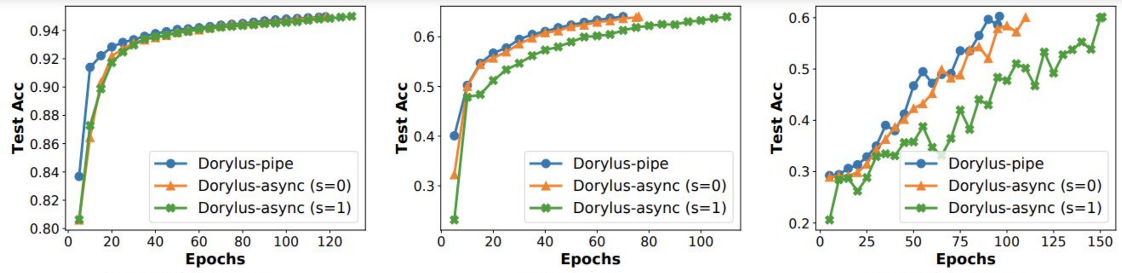


Figure 6: Per-epoch GCN time for async (s=0) and async (s=1) normalized to that of pipe.



(a) Reddit-small
R[s=0]:1.00, R[s=1]:1.07, Accuracy:94.96%

(b) Amazon
R[s=0]:1.09, R[s=1]:1.57, Accuracy:64.08%

(c) Reddit-large
R[s=0]:1.14, R[s=1]:1.58, Accuracy:60.07%

Figure 5: Asynchronous progress for GCN: All three versions of Dorylus achieve the final accuracy *i.e.*, **94.96%**, **64.08%**, **60.07%** for the three graphs). `Friendster` is not included because it does not come with meaningful features and labels.

Time and Cost comparisons

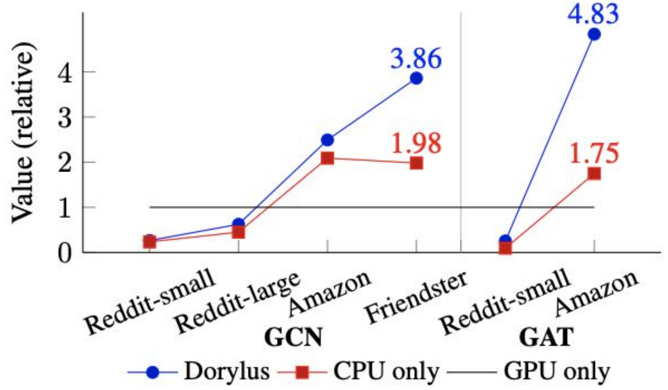
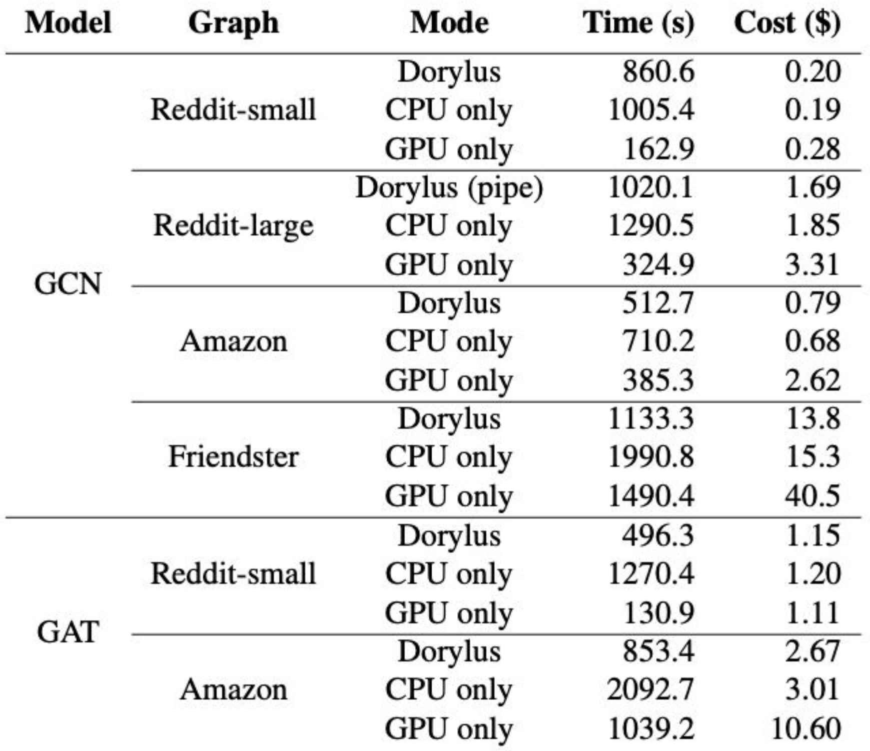


Figure 7: Dorylus, with Lambdas, provides up to 2.75× performance-per-dollar than using the CPU-only variant.



Model	Graph	Mode	Time (s)	Cost ($)
GCN	Reddit-small	Dorylus	860.6	0.20
		CPU only	1005.4	0.19
		GPU only	162.9	0.28
	Reddit-large	Dorylus (pipe)	1020.1	1.69
		CPU only	1290.5	1.85
		GPU only	324.9	3.31
	Amazon	Dorylus	512.7	0.79
		CPU only	710.2	0.68
		GPU only	385.3	2.62
	Friendster	Dorylus	1133.3	13.8
		CPU only	1990.8	15.3
		GPU only	1490.4	40.5
GAT	Reddit-small	Dorylus	496.3	1.15
		CPU only	1270.4	1.20
		GPU only	130.9	1.11
	Amazon	Dorylus	853.4	2.67
		CPU only	2092.7	3.01
		GPU only	1039.2	10.60

Table 4: We ran Dorylus in 3 different modes: “Dorylus”, our best Lambda variant using async(s=0) (except in one case), the “CPU only” variant, and the “GPU only” variant. For each mode we used multiple combinations of models and graphs. For each run we report the total end-to-end running time and the total cost.

Comparison over multiple servers

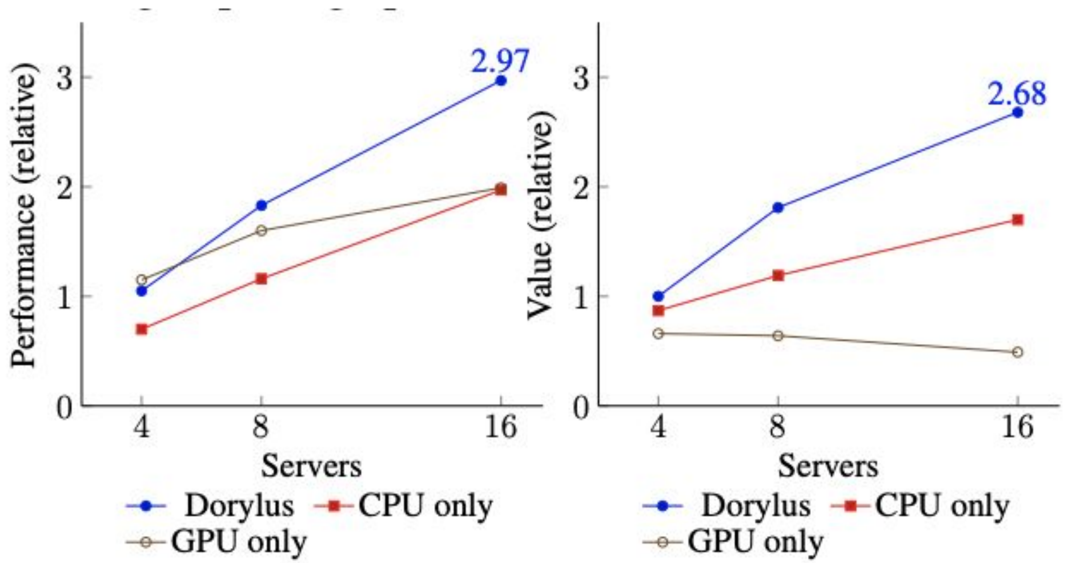
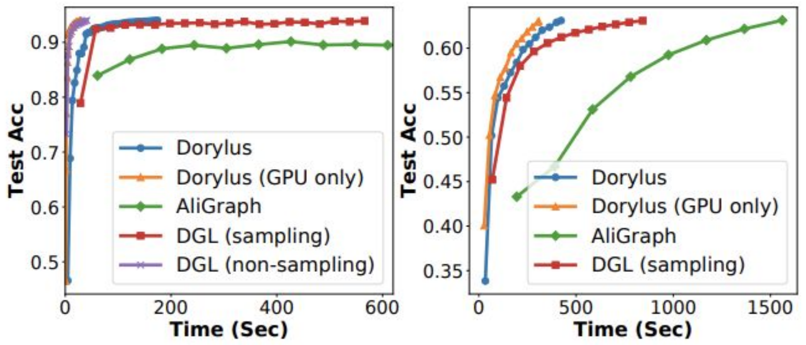


Figure 8: Normalized GCN training performance and value over `Amazon` with varying numbers of graph servers.

Comparison with other solutions



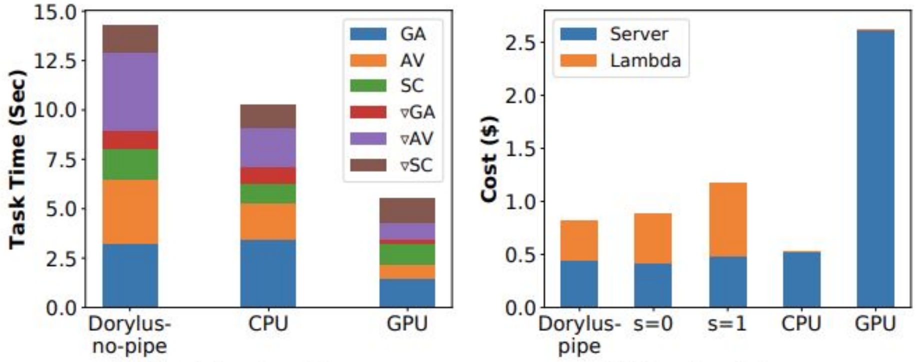
(a) `Reddit-small` (b) `Amazon`

Figure 9: Accuracy comparisons between Dorylus, Dorylus (GPU only), AliGraph, DGL (sampling), and DGL (non-sampling). DGL (non-sampling) uses a single V100 GPU and could not scale to `Amazon`. Each dot indicates five epochs for Dorylus and DGL (non-sampling), and one epoch for DGL (sampling) and AliGraph.

Cost/time comparisons

Graph	System	Time (s)	Cost ($)
Reddit-small	Dorylus	165.77	0.045
	Dorylus (GPU only)	28.06	0.052
	DGL (sampling)	566.33	0.480
	DGL (non-sampling)	33.64	0.028
	AliGraph	–	–
Amazon	Dorylus	415.23	0.654
	Dorylus (GPU only)	308.27	2.096
	DGL (sampling)	842.49	5.728
	DGL (non-sampling)	–	–
	AliGraph	1560.66	1.498

Table 5: Evaluation of end-to-end performance and costs of Dorylus and other GNN training systems. Each time reported is the time to reach the target accuracy.



(a) Task time breakdown (b) Cost breakdown

Figure 10: Time and cost breakdown for the `Amazon` graph.