PETUUM

Carnegie Mellon University

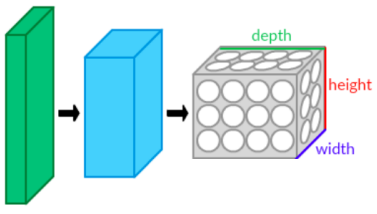
Litz: Elastic Framework for High-Performance Machine Learning

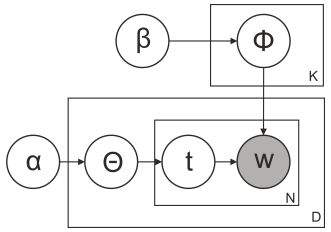
Aurick Qiao, Abutalib Aghayev, Weiren Yu, Haoyang Chen,
Qirong Ho, Garth A. Gibson, Eric P. Xing

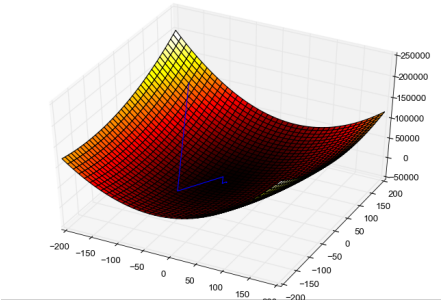
Petuum, Inc., Carnegie Mellon University,
Beihang University, Vector Institute

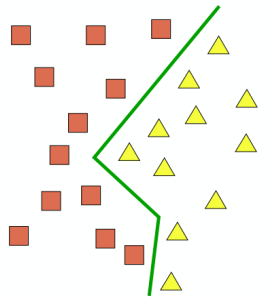
USENIX Annual Technical Conference (ATC) 2018

Machine Learning (ML) in Clouds & Data Centers













PETUUM

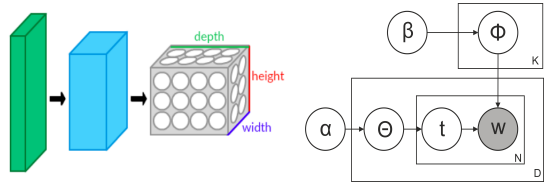
Machine Learning (ML) in Clouds & Data Centers

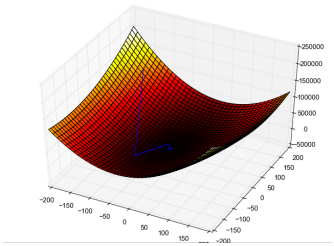
With advancements in distributed ML:

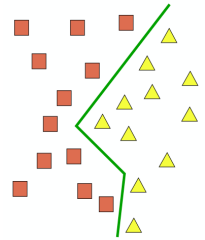
✔ Better scalability

✔ Higher throughput

✔ Faster convergence







But implementations are often:

✘ Inelastic

Jobs cannot elastically scale out or in.



PETUUM

Why Inelastic?

Efficient distributed ML training need flexible control over their execution.

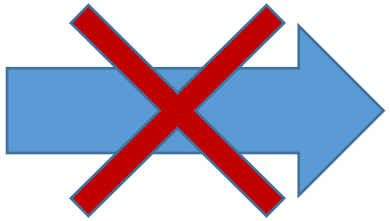
1. Controlling placement of mutable state **(Stateful Workers)**
2. Scheduling tasks at the application level **(Model Scheduling)**
3. Relaxing consistency of shared-memory **(Relaxed Consistency)**

PETUUM

Why Inelastic?

Efficient distributed ML training need flexible control over their execution.

1. **Stateful Workers**
2. **Model Scheduling**
3. **Relaxed Consistency**





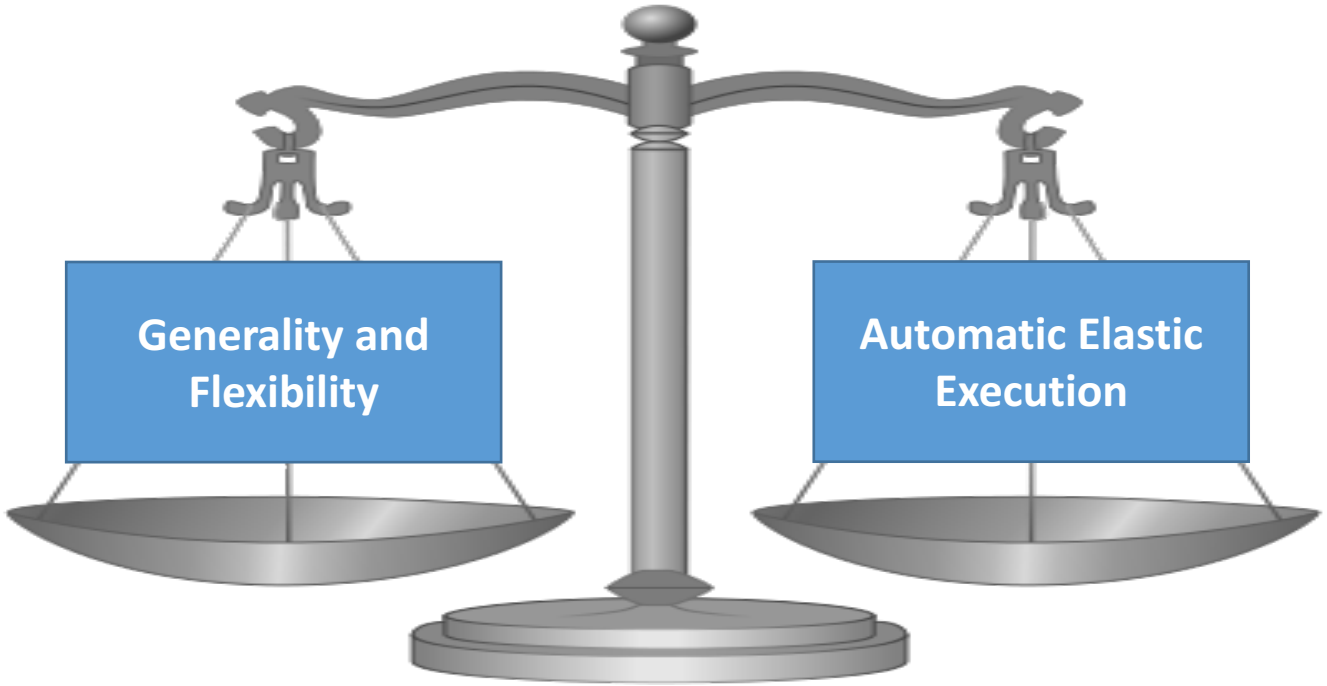


Difficult to implement using existing big data frameworks.

PETUUM

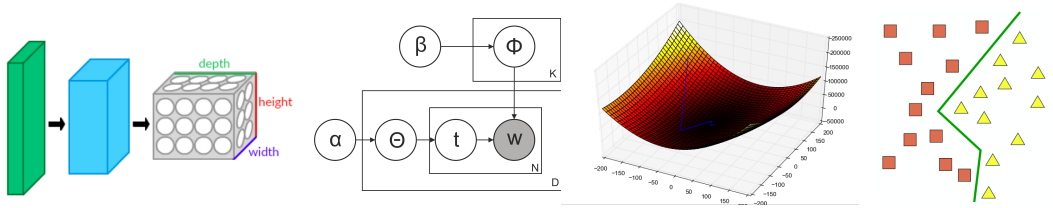
Litz Elastic ML Framework

Goal: design framework that balance…



PETUUM

Litz Elastic ML Framework



Event-driven Programming Model

Task-driven Consistency Model

Over-partitioned Application State

Elastic Execution System



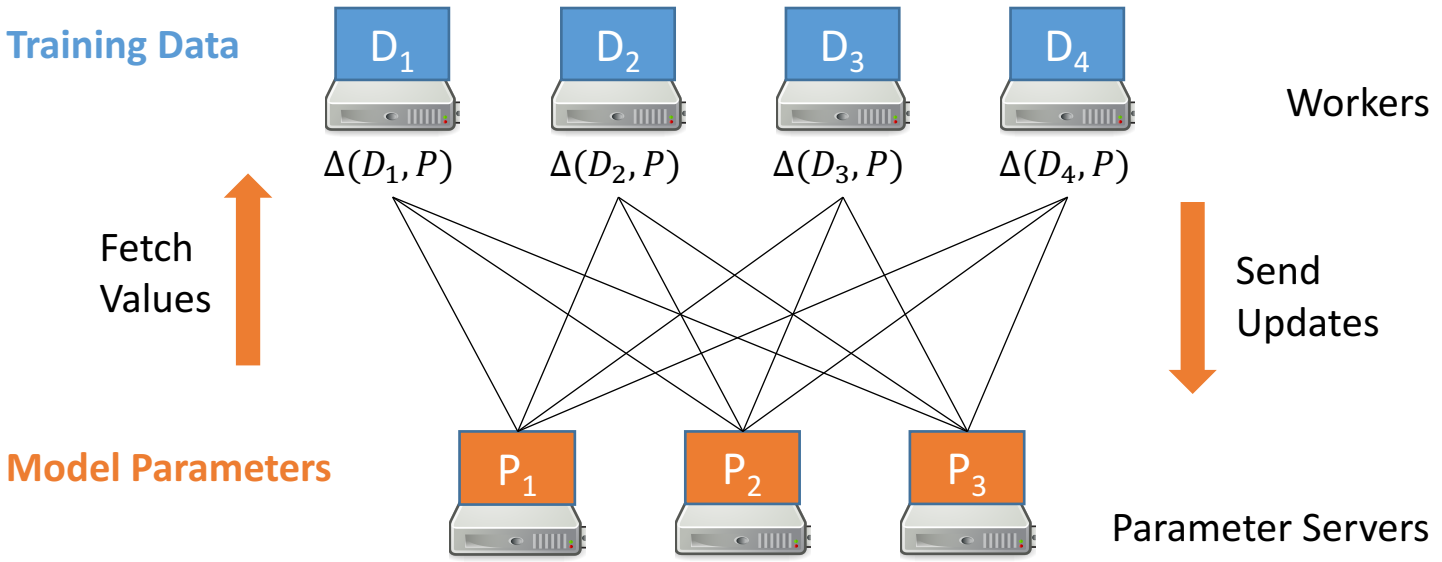
Applications on Litz can:

1. Complete faster by scaling out.
2. Quickly scale in when requested.
3. Have competitive performance with non-elastic systems.

PETUUM

Background: Parameter Server (PS)

Litz builds upon the popular Parameter Server **(PS)** architecture.

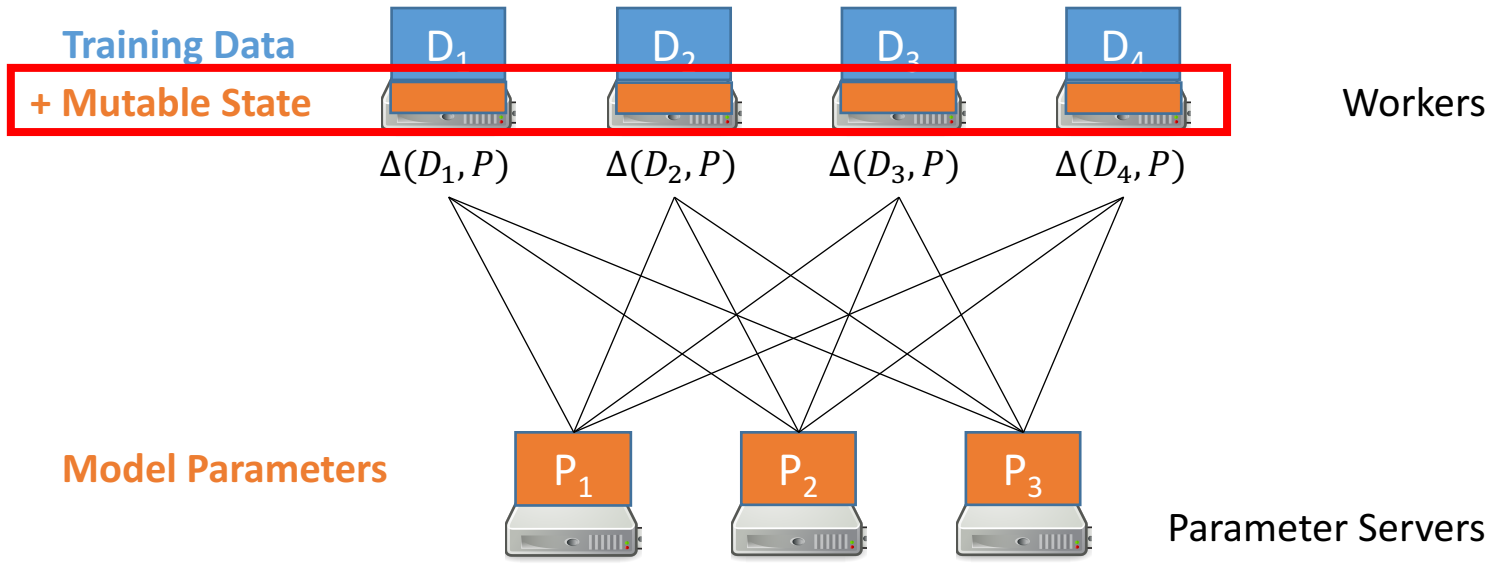


NIPS ’12, NIPS ’13, OSDI ’14, SoCC ’15

PETUUM

Background: Stateful Workers

ML applications may co-locate mutable state with training data to improve locality of access (**Stateful Workers**).

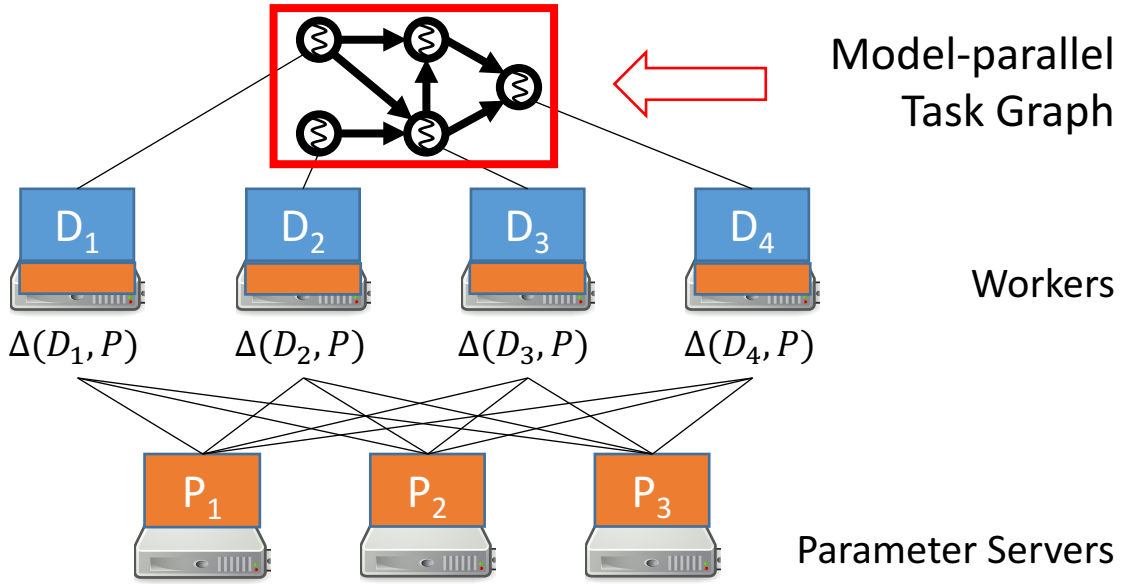


Cluster assignments, parameters in non-parametric models, etc.

PETUUM

Background: Model Scheduling

ML applications may schedule update computations according to dependency structures in the model (**model scheduling**).

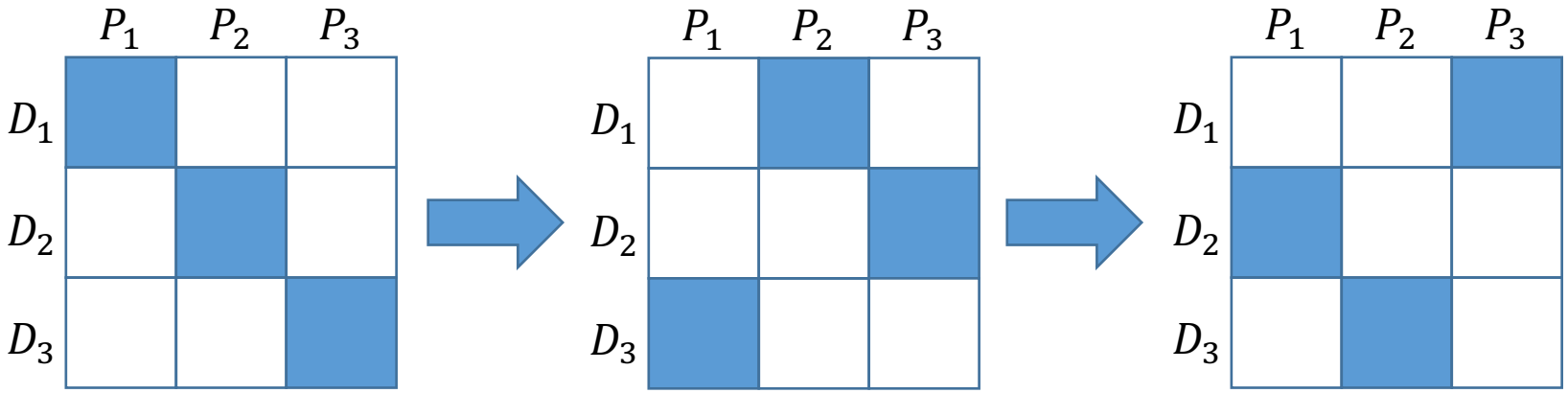


Can greatly improve convergence rate of models (EuroSys '16)

PETUUM

Background: Model Scheduling

Example: DSGD (KDD '11)



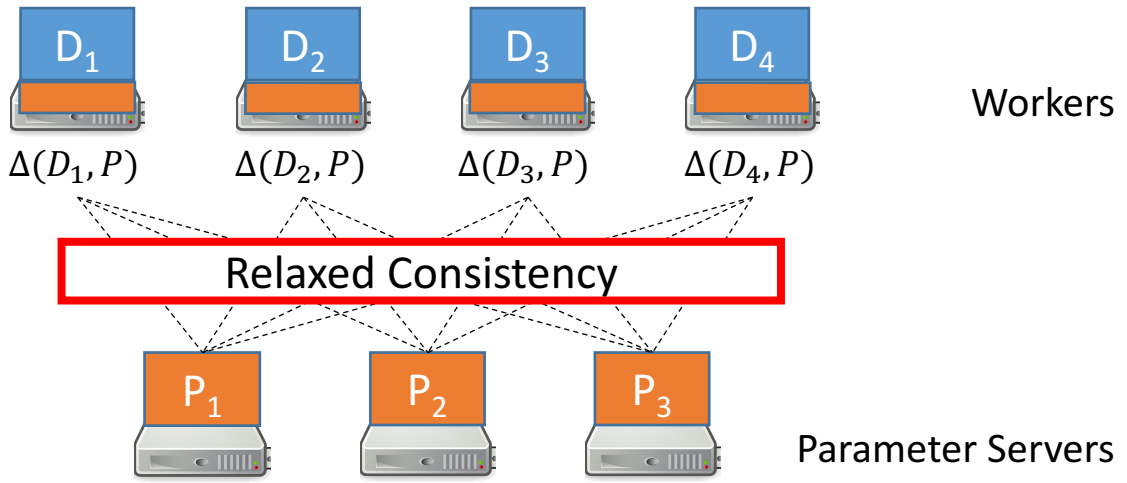
Only schedules non-overlapping parameter updates in parallel.

Also: dynamic scheduling, eg. parallel coordinate descent

PETUUM

Background: Relaxed Consistency

Workers may access values in parameter server using a weaker consistency model (**relaxed consistency**).

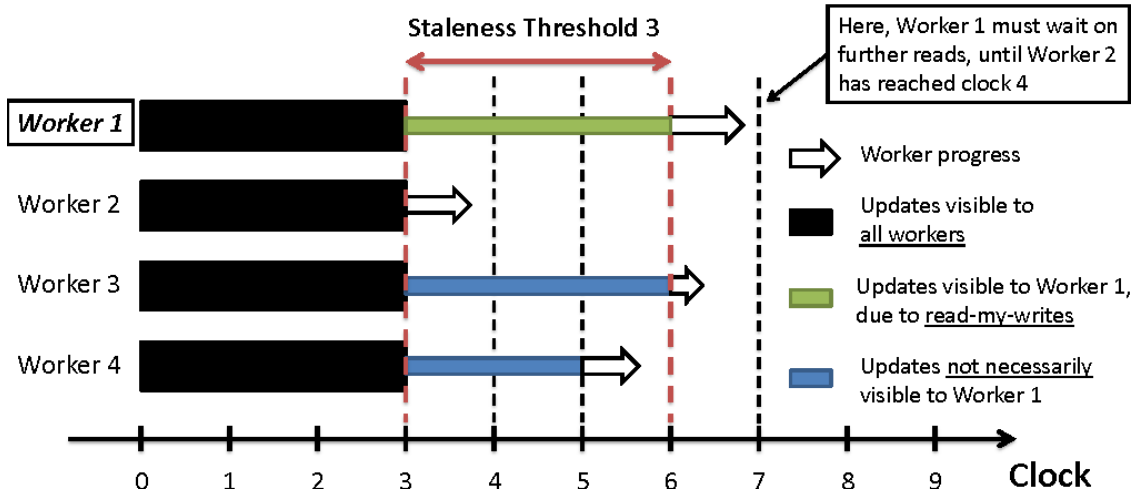


Reduces cost of synchronization to achieve higher throughput.

PETUUM

Background: Relaxed Consistency

Example: Stale-Synchronous Parallel (NIPS '13)



Workers can use stale parameters (up to a given staleness limit).

Also: staleness-aware algorithms, eg. AdaptiveRevision (NIPS '14)

PETUUM

Challenges for Elastic ML Framework

Stateful Workers **Model Scheduling** **Relaxed Consistency**

Managing application-defined worker state.

- Re-balance across a different number of nodes.

Supporting application-level model scheduling.

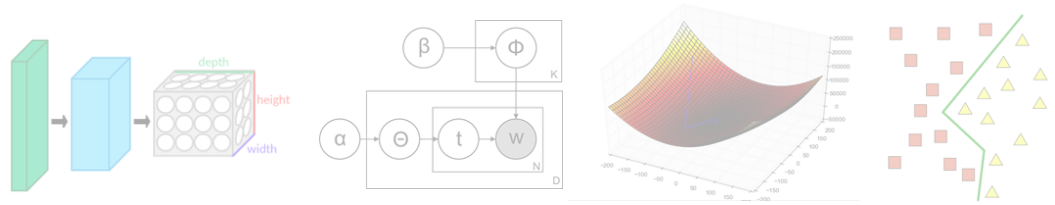
- Provide flexible tools for task scheduling.
- Hide away cluster resource details (eg. nodes joining/leaving the active job).

Supporting relaxed consistency models.

- Elastic scaling should not rely on synchronous execution.

PETUUM

Litz Elastic Framework



Event-driven Programming Model

Task-driven Consistency Model

Over-partitioned Application State

Elastic Execution System

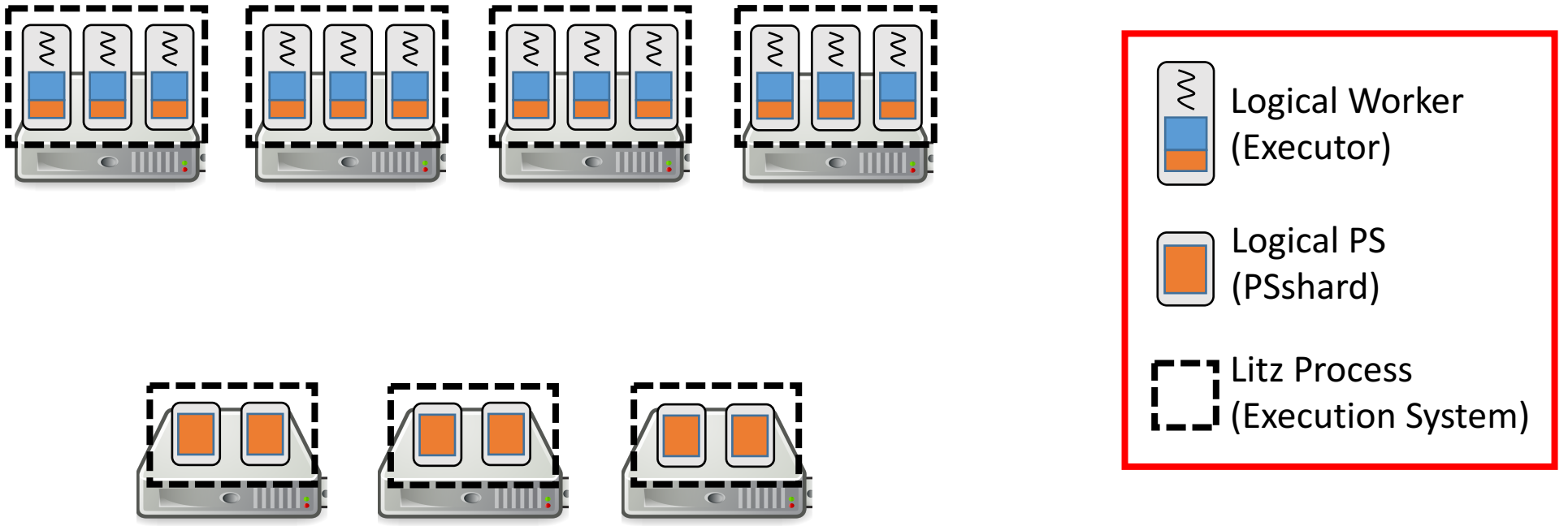


Applying **over-partitioning** to the parameter server architecture.

PETUUM

Over-partitioning

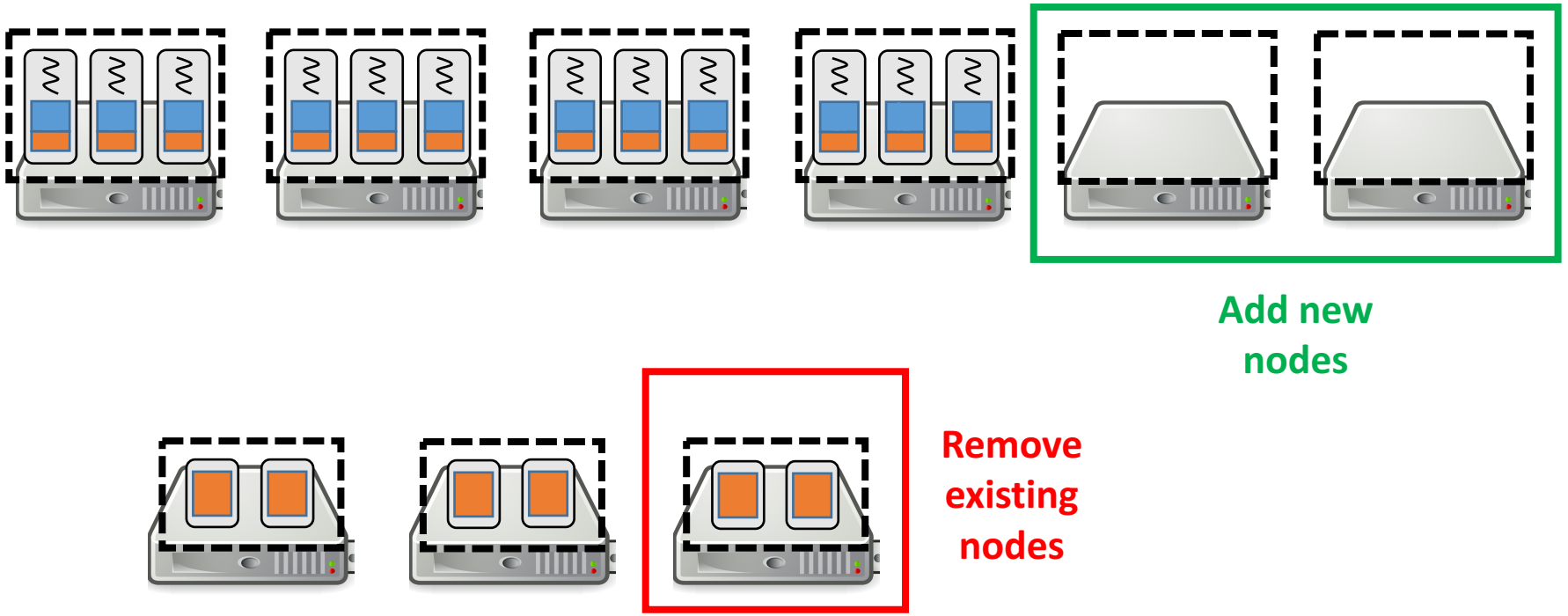
Partition data and parameters across many *logical* workers (**Executors**) and *logical* parameter servers (**PSshards**).



PETUUM

Over-partitioning

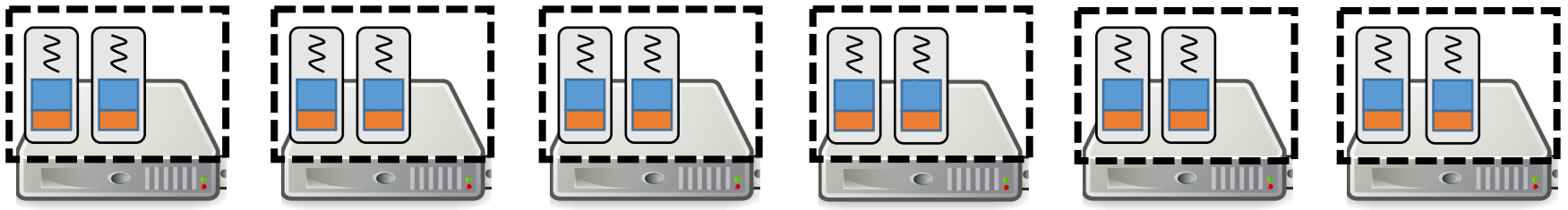
Total number of **Executors** and **PSshards** doesn't change, but can be re-balanced between physical nodes when resource availability changes.

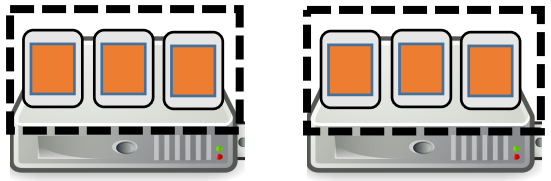


PETUUM

Over-partitioning

Total number of **Executors** and **PSshards** doesn't change, but can be re-balanced between physical machines when their availability changes.





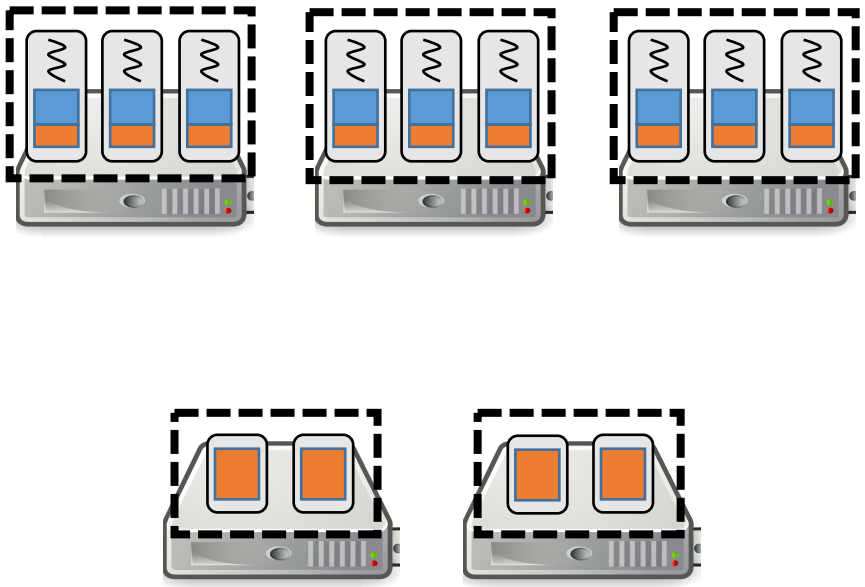
Automatically migrated by **elastic execution system**

PETUUM

Over-partitioning

Advantages:

- App can define arbitrary state.
  - Treated as black box by the execution system.
- App does not need to handle addition/removal of workers.
  - Tasks can be scheduled assuming a fixed number of Executors.



PETUUM

Over-partitioning

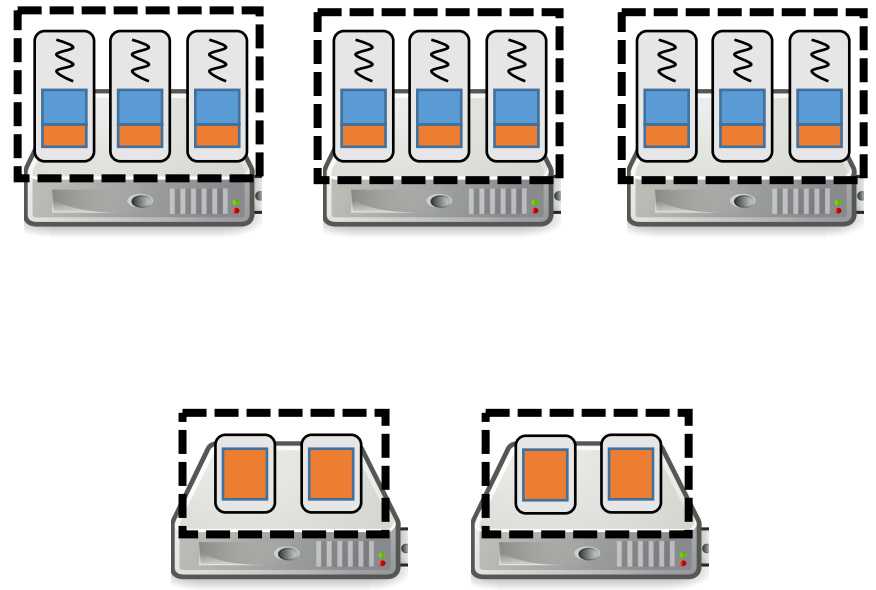
Caveat:

More Executors and PSshards are needed to achieve:

- Higher max parallelism.
- Better load balancing.

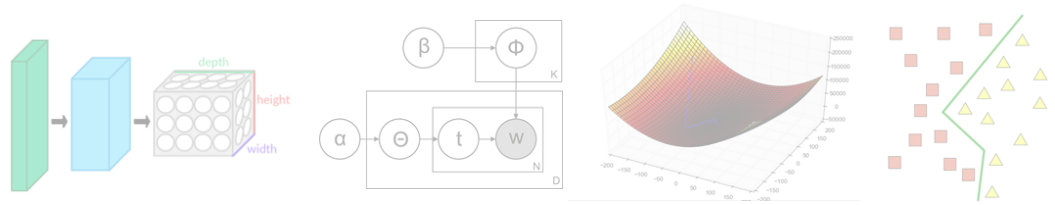
Worse performance overhead for execution system.

Major goal: Mitigate the performance overhead of over-partitioning.



PETUUM

Litz Elastic Framework



Event-driven Programming Model

Task-driven Consistency Model

Over-partitioned Application State

Elastic Execution System



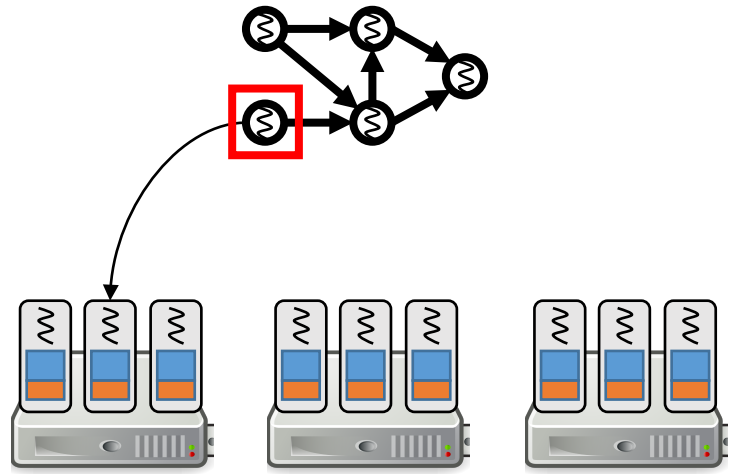
Decompose update computations into **micro-tasks**, which are scheduled and executed by the application using an **event-driven** API.

PETUUM

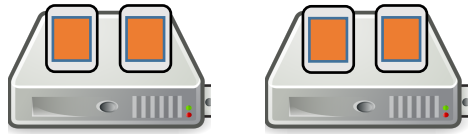
Event-Driven Programming Model

Micro-task: app-defined short-lived task.

1. Runs on a single Executor.
   - Can access local state and training data.
2. Can read values from PS.
3. Can submit updates to PS.



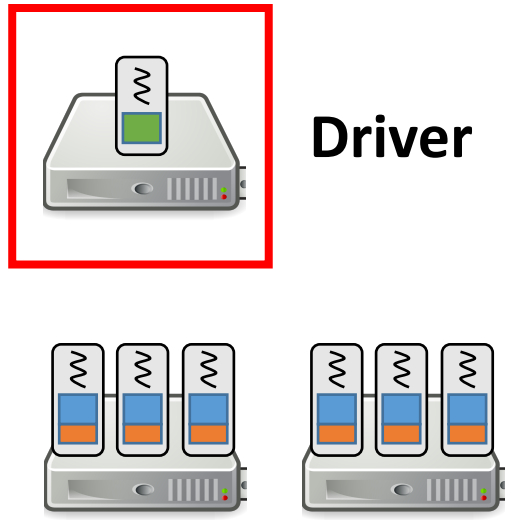
- Application provides *task function*:
  - **RunTask(args)**
- PS access provided by Litz framework:
  - **PSGet(key), PSUpdate(key, update)**



PETUUM

Event-Driven Programming Model

Driver: micro-task scheduler.

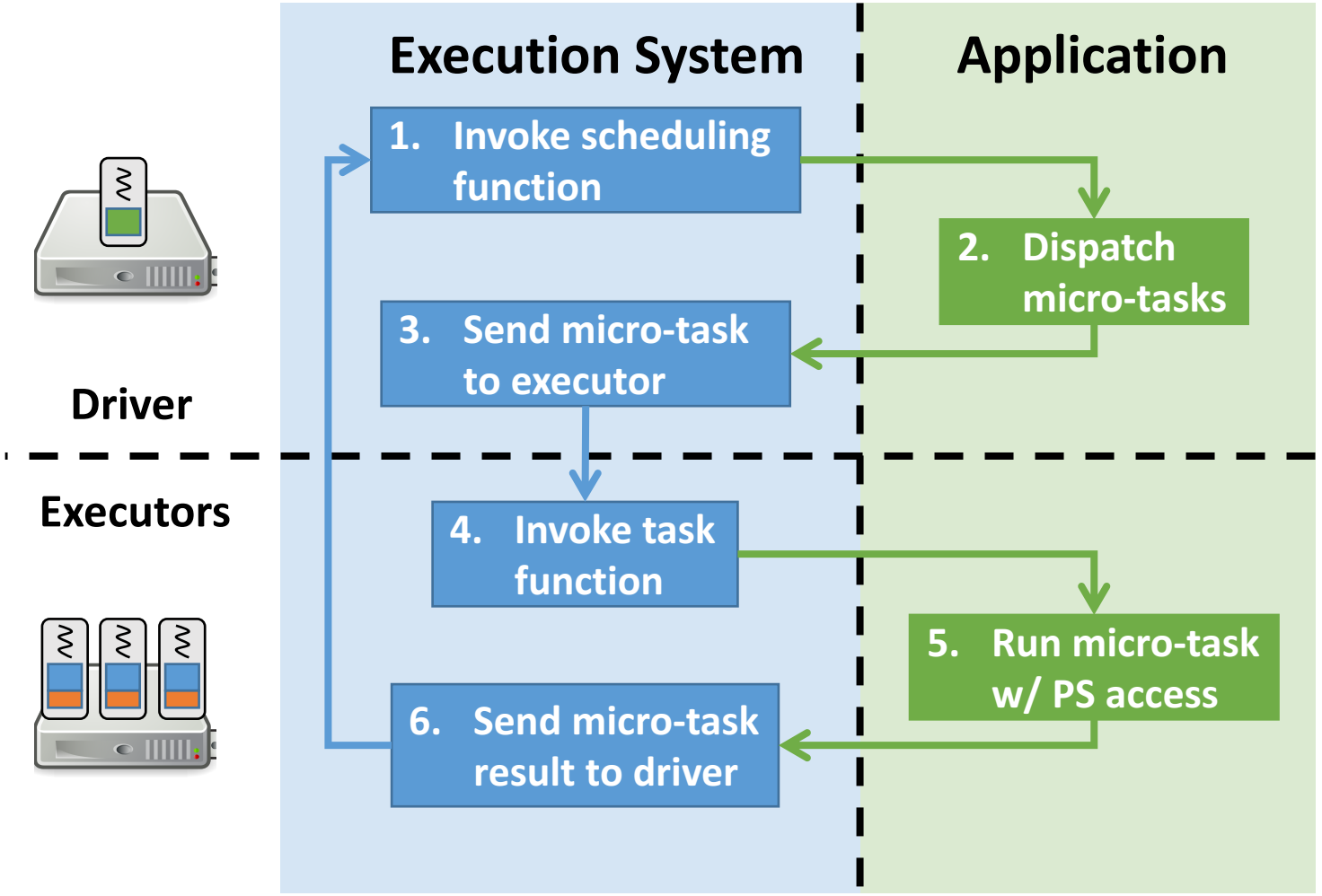


- Application provides *scheduling functions*:
  - **HandleTaskCompletion(result)**
  - **DispatchInitialTasks()**

- Litz framework provides dispatch function:
  - **DispatchTask(executor, args)**

PETUUM

Event-Driven Programming Model



Execution system can migrate Executors and PSshards between invoking app-defined functions.

⇩

Elastic scaling transparent to app.

Keeping micro-tasks short gives frequent chances to re-balance.

⇩

Better response time for elastic scaling events.

PETUUM

Litz Elastic Framework

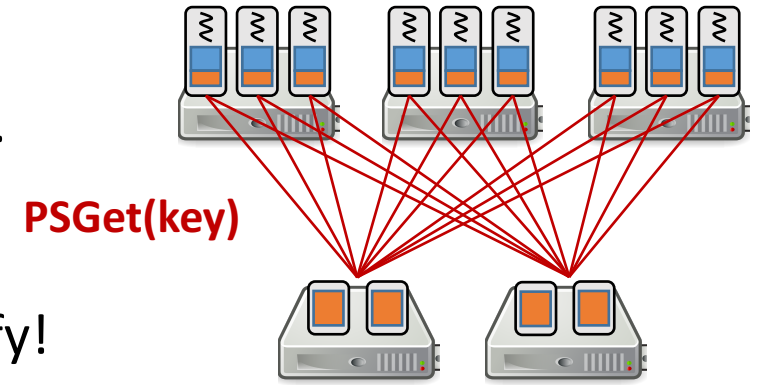
Over-Partitioning + Event-Driven Programming Model

- Apps have flexible control over their state and task scheduling.
- Litz can automatically re-balance workload during elastic scaling.

However…

Over-partitioning + Micro-tasks = High Overhead

- Fetching parameters creates high communication load.
- Want workers to cache and re-use parameters.
- How to decide when to use cached parameters?
- Too many dependencies between micro-tasks to specify!



PETUUM

Task-Driven Consistency Model

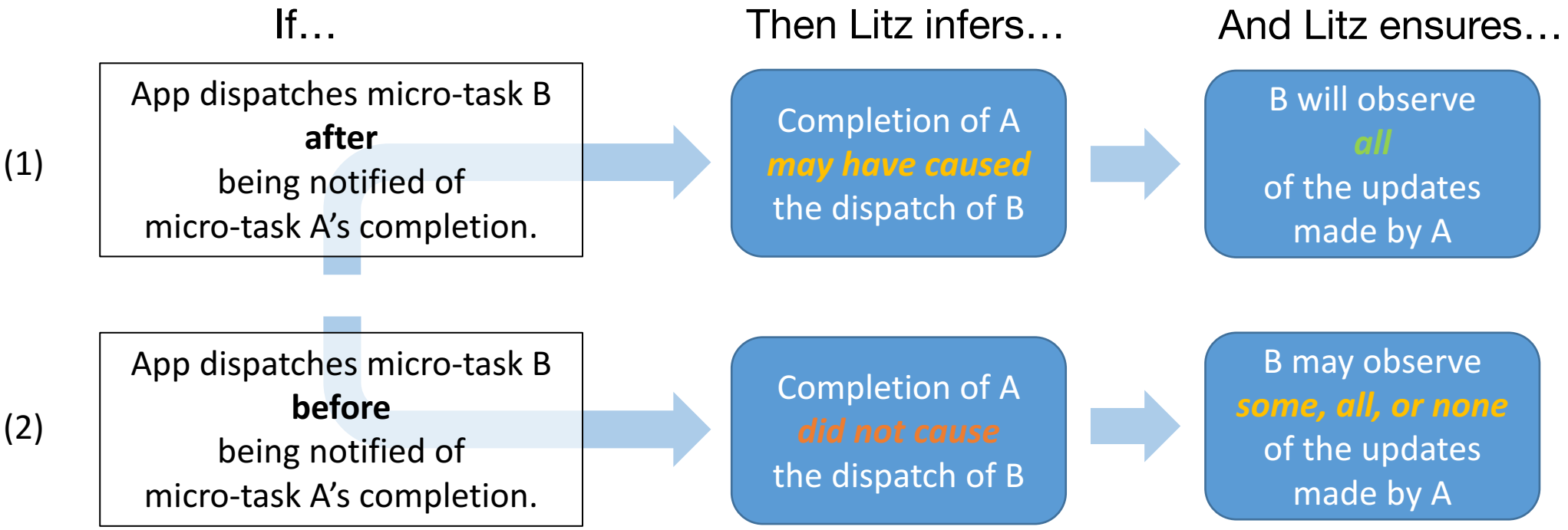
Idea 

- Since the execution system invokes app-defined functions, it can observe how the application makes scheduling decisions.
- Automatically infer causal relationships between micro-tasks based on:
  1. When the application dispatches a micro-task.
  2. When the application is notified of a micro-task's completion.

Task-driven Consistency Model

PETUUM

Task-Driven Consistency Model



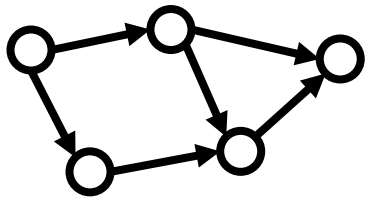
(1) Strong ordering guarantee, needed for app-level model scheduling.

(2) Intentionally weak, allows execution system to automatically use cached parameter values.

PETUUM

Examples

Scheduling micro-task graphs

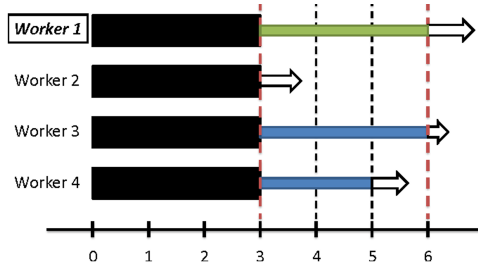


On job start: dispatch all micro-tasks with no dependencies.

On micro-task completion: dispatch all micro-tasks whose dependencies are now satisfied.

Ensures each micro-task observes all PS updates made by micro-tasks it depends on.

Stale-synchronous parallel



On job start: dispatch all micro-tasks in first **s+1** iterations, where **s** is the staleness limit.

On micro-task completion: if iter **i** completed, dispatch all micro-tasks in iteration **i+s+1**.

Micro-tasks in each iteration may observe some, all, or none of the updates made by the micro-tasks in the previous **s** iterations.

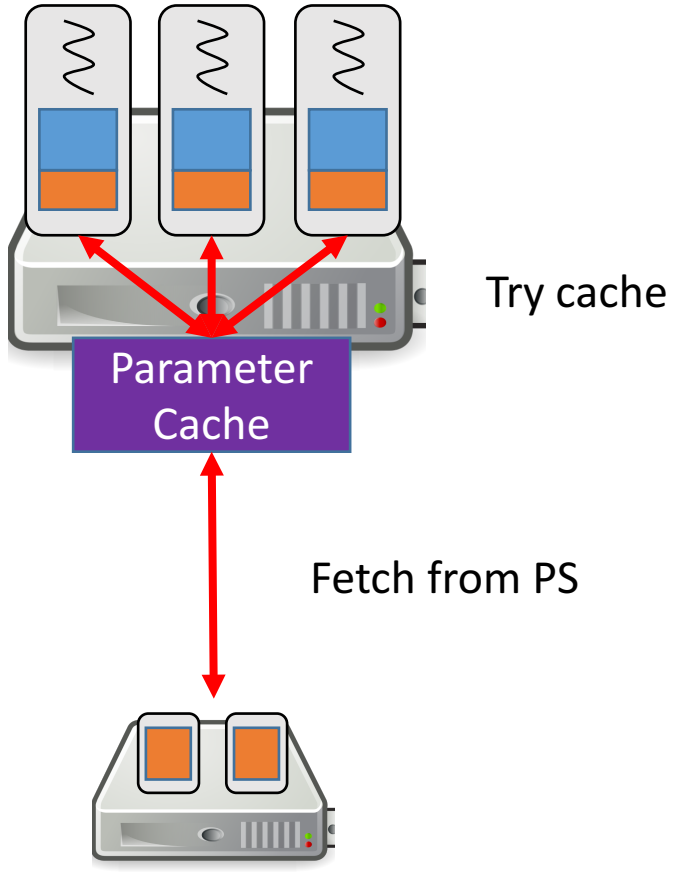
Micro-tasks may re-use cached parameters fetched within the previous **s** iterations.

PETUUM

Task-Driven Consistency Model

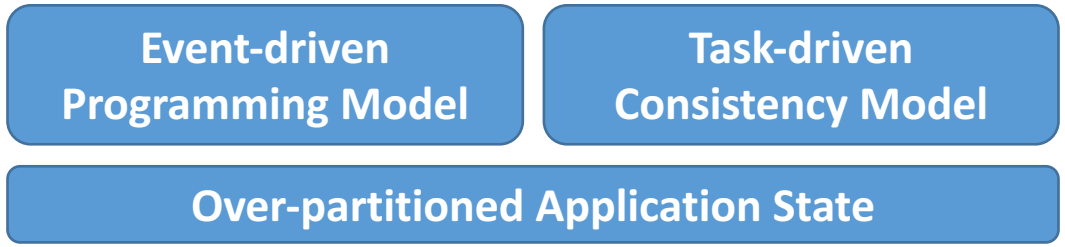
A parameter cache can be implemented efficiently in the execution system.

1. Maintain a version number, increment each time a micro-task completes.
2. Tag each micro-task with the version at the time it was dispatched.
3. Tag each cached parameter with the version at the time it was fetched.
4. Re-use cached parameter value if its version is at least the micro-task version.



PETUUM

Elastic Execution System



- Decouples ML application from underlying cluster environment.
- Provides flexible programming tools for distributed ML.

- Gives visibility and control over how application is executed.
- Execution system can provide transparent elasticity and efficient execution.



PETUUM

Elastic Execution System

Our C++ implementation:

Elastic scaling:
- Migrate Executors/PSshards to new nodes when added.
- Migrate Executors/PSshards from existing nodes when requested.
- Static load balancing strategy (balanced number of Executors/PSshards).

Optimizations:
- Parameter caching (according to task-based consistency model).
- Update aggregation (combine updates before sending to PS).
- Co-operative multitasking between Executors.

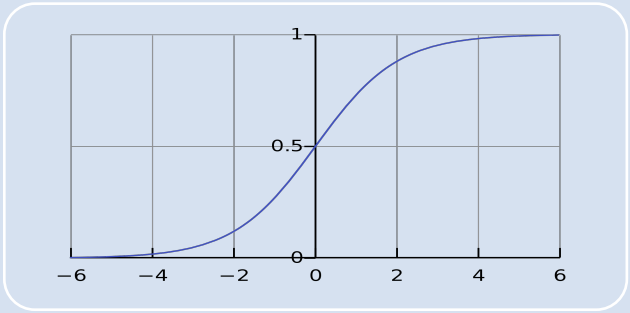
PETUUM

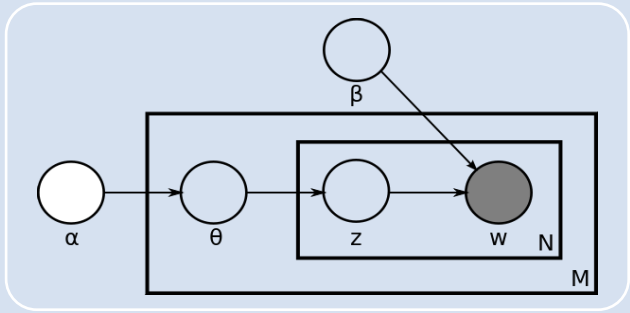
Evaluation

1. **Over-partitioning**
   - How much performance overhead?
2. **Elastic scaling**
   - Does scaling out help?
   - How quickly to scale in?
3. **Performance**
   - How fast compared to state-of-the-art non-elastic systems?

PETUUM

Evaluation: Applications





Multinomial Logistic Regression (MLR)

- Multiclass classification
- Gradient-based optimization
- Uses relaxed consistency
- Workers are stateless
- Dataset: ImageNet 81GB

Latent Dirichlet Allocation (LDA)

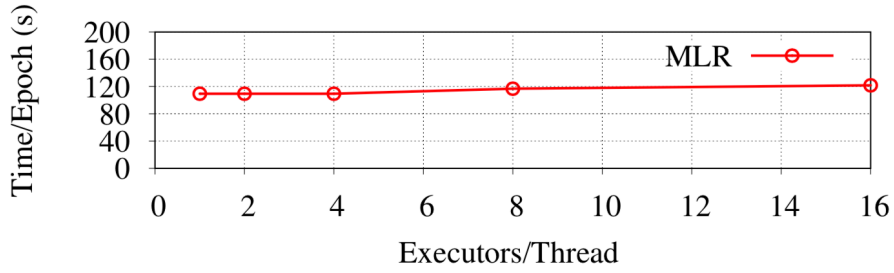
- Topic modeling
- Sampling-based approximation
- Uses model scheduling
- Workers are stateful
- Dataset: ClueWeb 88GB

PETUUM

Performance Overhead of Over-partitioning

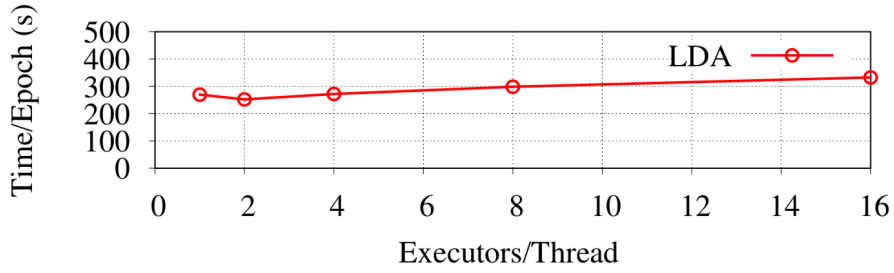
Setup: fix number of physical nodes, increase number of Executors.

MLR on 4 physical nodes



1.1x runtime

LDA on 12 physical nodes



1.2x runtime

Baseline number of Executors saturates available physical cores.

16x executors

Benefit: potential to scale out to 16x as many nodes when available.

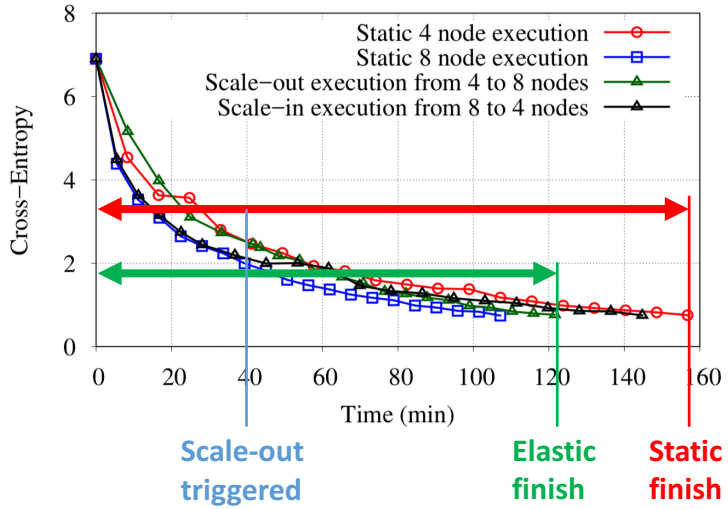
PETUUM

Does scaling out result in faster jobs?

Not obvious! ie. if scaling out takes too long, or if system is not scalable.

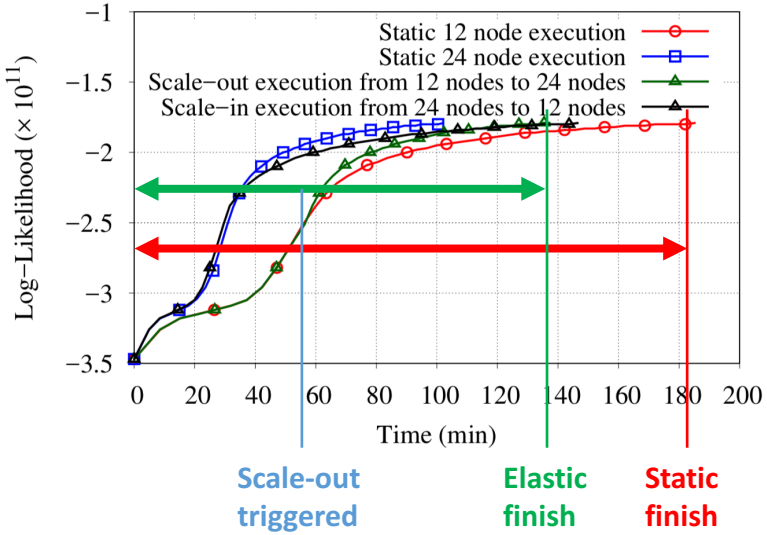
Setup: scale out to more nodes during job, compare with static execution.

MLR: scale out 4 to 8 nodes



22% faster completion by scaling out.

LDA: scale out 12 to 24 nodes



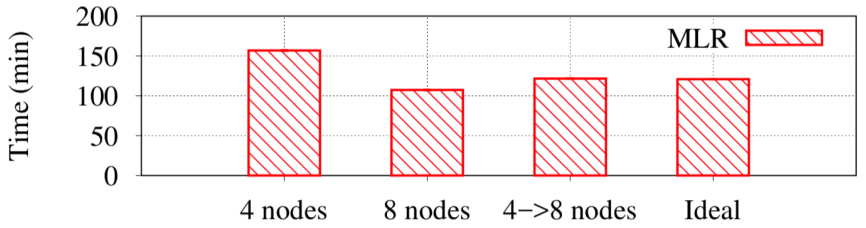
27% faster completion by scaling out.

PETUUM

Speed-up from Elastic Scale-out

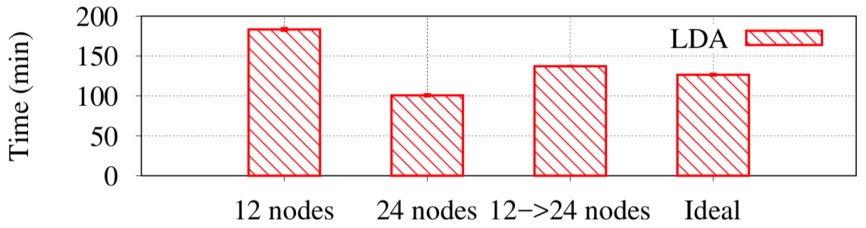
Setup: compare with ideal scale-out, calculated from static pre-scaling and post-scaling experiments assuming instantaneous re-balancing.

MLR: scale out
4 to 8 nodes



1.01x ideal

LDA: scale out
12 to 24 nodes



1.05x ideal

PETUUM

Eviction Latency from Elastic Scale-in

Can nodes be released quickly upon request?

Setup: request half of the nodes, measure time until they are released.

Application	Scaling	Time to Release
MLR	8 to 4	**2.5 sec**
LDA	24 to 12	**43 sec**

MLR: stateless Executors, only PSshards to be transferred.

LDA: 55GB of Executor state needs to be transferred.

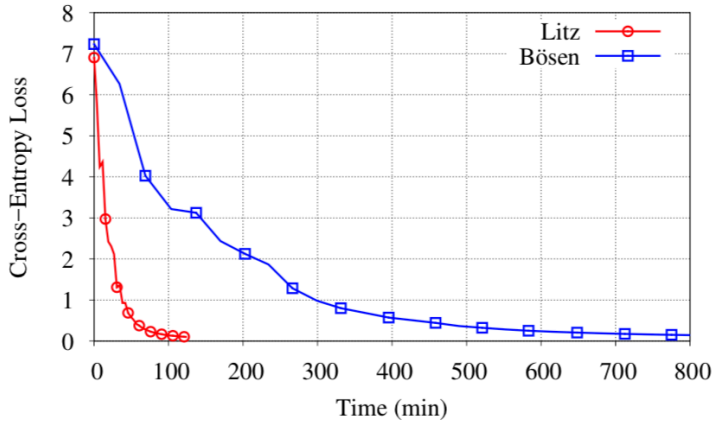
PETUUM

Non-Elastic Performance

How does Litz perform in non-elastic settings?

Setup: compare with same applications on non-elastic ML systems.

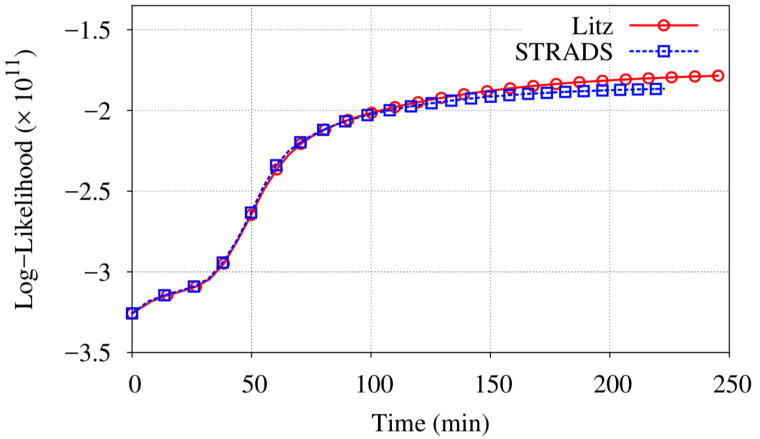
MLR compare with Bosen (SoCC '15), for stale-synchronous parallel apps.



MLR with SSP, staleness = 2

8x faster

LDA compare with STRADS (EuroSys '16), for model scheduling apps.



LDA with block-partitioned model scheduling

6% slower

PETUUM

Summary

Tackle elasticity for distributed ML using three techniques:

1. **Over-partitioning in Parameter Server**
   - Automatically re-balance application state.
2. **Event-driven Programming Model**
   - Quick response to elastic scaling events.
3. **Task-driven Consistency Model**
   - Mitigates performance overhead of the above.

The Litz framework:

- Supports stateful workers, model scheduling, relaxed consistency.
- Enables elastic execution without sacrificing performance.

PETUUM

PETUUM

Carnegie Mellon University

Thank you!