Dynamic Parameter Allocation in Parameter Servers

Alexander Renz-Wieland¹, Rainer Gemulla², Steffen Zeuch¹,³, Volker Markl¹,³

¹TU Berlin, ²University of Mannheim, ³DFKI

VLDB 2020
Takeaways

▶ Key challenge in distributed Machine Learning (ML): communication overhead

▶ Parameter Servers (PSs)
  ▶ Intuitive
  ▶ Limited support for common techniques to reduce overhead

▶ How to improve support?
  ▶ Dynamic parameter allocation

▶ Is this support beneficial?
  ▶ Up to two orders of magnitude faster
Background: Distributed Machine Learning

- Distributed training is a necessity for large-scale ML tasks
- Parameter management is a key concern
- *Parameter servers* (PS) are widely used

![Logical and Physical Diagram]

- Logical:
  - Parameter Server
  - Workers (worker) communicating via `push()` and `pull()`

- Physical:
  - Workers (worker) accessing parameters and communicating directly
Problem: Communication Overhead

- Communication overhead can limit scalability
- Performance can fall behind a single node

Training knowledge graph embeddings (RESCAL, dimension 100):
Problem: Communication Overhead

- Communication overhead can limit scalability
- Performance can fall behind a single node

Training knowledge graph embeddings (RESCAL, dimension 100):

![Graph showing epoch run time in minutes for different parallelisms]
Problem: Communication Overhead

- Communication overhead can limit scalability
- Performance can fall behind a single node

Training knowledge graph embeddings (RESCAL, dimension 100):

![Graph showing epoch run time in minutes for different parallelism (nodes x threads) settings for Classic PS (PS-Lite), Classic PS with fast local access, and Dynamic Allocation PS (Lapse), incl. fast local access. Times include: 4.5h, 1.2h, 4h, 0.6h, 2.4h, 0.4h, 1.5h, 0.2h.]
How to reduce communication overhead?

- Common techniques to reduce overhead:
  - Data clustering
  - Parameter blocking
  - Latency hiding

- Key is to avoid remote accesses
- Do PSs support these techniques?
  - Techniques require local access at different nodes over time
  - But PSs allocate parameters statically
How to reduce communication overhead?

- **Common techniques to reduce overhead:**
  - **Data clustering**
  - **Parameter blocking**
  - **Latency hiding**

  - Key is to avoid remote accesses
  - Do PSs support these techniques?
    - Techniques require local access at different nodes over time
    - But PSs allocate parameters statically
Dynamic Parameter Allocation

What if the PS could allocate parameters dynamically?

Localize(parameters)

Would provide support for

- Data clustering ✓
- Parameter blocking ✓
- Latency hiding ✓

We call this **dynamic parameter allocation**
The Lapse Parameter Server

- **Features**
  - Dynamic allocation
  - Location transparency
  - Retains sequential consistency

PS per-key consistency guarantees (for synchronous operations)

<table>
<thead>
<tr>
<th></th>
<th>Classic</th>
<th>Lapse</th>
<th>Stale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PRAM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Causal</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Sequential</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Serializability</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

- **Many system challenges (see paper)**
  - Manage parameter locations
  - Route parameter accesses to current location
  - Relocate parameters
  - Handle reads and writes during relocations
  - All while maintaining sequential consistency
Experimental study

Tasks: matrix factorization, knowledge graph embeddings, word vectors
Cluster: 1–8 nodes, each with 4 worker threads, 10 GBit Ethernet

1. Performance of Classic PSs
   ▶ 2–8 nodes barely outperformed 1 node in all tested tasks

2. Effect of dynamic parameter allocation
   ▶ 4–203x faster than a Classic PSs, up to linear speed-ups

---

![Graph showing performance comparison between Classic PSs and Dynamic Allocation PSs](graph.png)
Comparison to Bounded Staleness PS

- Matrix factorization (matrix with 1b entries, rank 100)
- Parameter blocking

![Graph showing comparison of Epoch run time in minutes across different parallelisms (nodes x threads) for Bounded staleness PS (Petuum), client sync., Bounded staleness PS (Petuum), server sync., and Dynamic Allocation PS (Lapse). The graph indicates a significant reduction in run time with increasing parallelism, with Bounded staleness PS showing a notable decrease compared to Dynamic Allocation PS. The y-axis represents Epoch run time in minutes, and the x-axis represents Parallelism (nodes x threads). The graph highlights the performance gain with 0.6x, 2.9x, and 8.4x improvements for different parallelism settings.]
Comparison to Bounded Staleness PS

- Matrix factorization (matrix with 1b entries, rank 100)
- Parameter blocking

![Graph comparing different PS methods](image)

- Bounded staleness PS (Petuum), client sync.
- Bounded staleness PS (Petuum), server sync.
- Dynamic Allocation PS (Lapse)

- Single-node overhead
- Non-linear scaling

Graph parameters:
- Parallelism (nodes x threads)
- Epoch run time in minutes

Legend:
- Bounded staleness PS (Petuum), client sync.
- Bounded staleness PS (Petuum), server sync.
- Dynamic Allocation PS (Lapse)

Comparative analysis showing efficiency and scalability across different parallelism levels.
Experimental study

Tasks: matrix factorization, knowledge graph embeddings, word vectors
Cluster: 1–8 nodes, each with 4 worker threads, 10 GBit Ethernet

1. Performance of Classic PSs
   ▶ 2–8 nodes barely outperformed 1 node in all tested tasks

2. Effect of dynamic parameter allocation
   ▶ 4–203x faster than a Classic PSs, up to linear speed-ups

3. Comparison to bounded staleness PSs
   ▶ 2–28x faster and more scalable

4. Comparison to manual management
   ▶ Competitive to a specialized low-level implementation

5. Ablation study
   ▶ Combining fast local access and dynamic allocation is key
Dynamic Parameter Allocation in Parameter Servers

- Key challenge in distributed Machine Learning (ML): communication overhead
- Parameter Servers (PSs)
  - Intuitive
  - Limited support for common techniques to reduce overhead
- How to improve support?
  - Dynamic parameter allocation
- Is this support beneficial?
  - Up to two orders of magnitude faster
- Lapse is open source:
  https://github.com/alexrenz/lapse-ps