CS 6453: Parameter Server

Soumya Basu
March 7, 2017

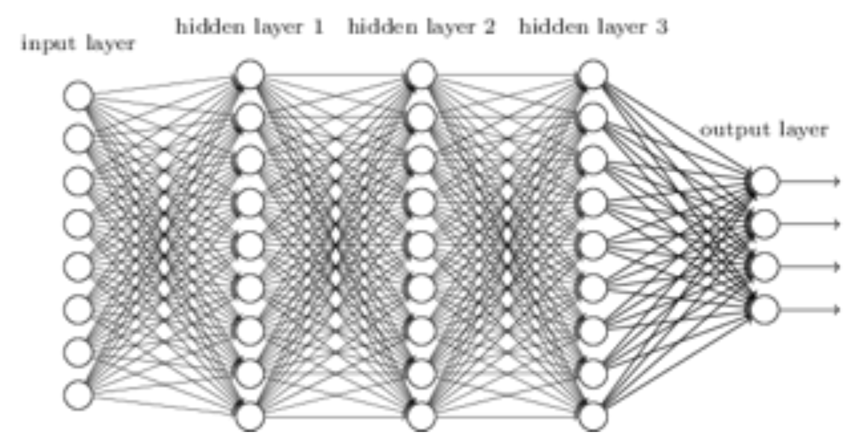
What is a Parameter Server?

- Server for large scale machine learning problems
- Machine learning tasks in a nutshell:

Feature Extraction →

(1, 1, 1)
(2, -1, 3)
(5, 6, 7)
…

Training →



- Design a server that makes the above fast!

Why Now?

- Machine learning is important!
  - Read the news to see why…
- Feature extraction fits nicely into Map-Reduce
  - **Many** systems take care of this problem…
- So, parameter server focuses on training models

Training in ML

- Training consists of the following steps:
    1. Initialize model with small random values
    2. Try to guess the right answer for your input set
    3. Adjust the model
    4. Repeat step 2-3 until your error is small enough

Systems view of Training

- Initialize model with small random values
    - Paid once- fairly trivial to parallelize
- Try to guess the right answer for your input set
    - Iterate through the input set many many times
- Adjust the model
    - Send a small update to the model parameters

Key Challenges

- Three main challenges of implementing a parameter server:
  - Accessing parameters requires lots of network bandwidth
  - Training is sequential and synchronization is hard to scale
  - Fault tolerance at scale (~25% failure rate for 10k machine-hour jobs)

First Attempts

- First attempts used memcached for synchronization [VLDB 2010]
  - Key-value stores have very large overheads
  - Synchronization costs are expensive and not always necessary

Second Generation

- Second generation of attempts were application specific parameter servers [WDSM 2012, NIPS 2012, NIPS 2013]
  - Fails to factor out common difficulties between many different types of problems
  - Difficult to deploy multiple algorithms in parallel

General Purpose ML

- General purpose machine-learning frameworks
  - Many have synchronization points -> difficult to scale
  - Key observation: cache state between iterations

GraphLab

- Distributed GraphLab [PVLDB 2012]
  - Uses coarse-grained snapshots for fault tolerance, impeding scalability
  - Doesn’t scale elastically like map-reduce frameworks
  - Asynchronous task scheduling is the main contribution

Piccolo

- Piccolo [OSDI 2010]
  - Most similar to this paper
  - Is not optimized for Machine Learning though

Technical Contribution

- Recall the three main challenges:
  - Accessing parameters requires lots of network bandwidth
  - Training is sequential and synchronization is hard to scale
  - Fault tolerance at scale

Dealing with Parameters

- What are parameters of a ML model?
  - Usually an element of a vector, matrix, etc.
  - Need to do lots of linear algebra operations
- Introduce new constraint: ordered keys
  - Typically some index into a linear algebra structure

Dealing with Parameters

- What are parameters of a ML model?
  - Usually an element of a vector, matrix, etc.
  - Need to do lots of linear algebra operations
- Introduce new constraint: ordered keys
  - Typically some index into a linear algebra structure

Dealing with Parameters

- High model complexity leads to overfitting
  - Updates don’t touch many parameters
- Range push-and-pull: Can update a range of values in a row instead of single key
- When sending ranges, use compression

Synchronization

- ML models try to find a good local min/min
- Need updates to be *generally* in the right direction
- Not important to have strong consistency guarantees all the time
- Parameter server introduces Bounded Delay

Fault Tolerance

- Server stores all state, workers are stateless
  - However, workers cache state across iterations
- Keys are replicated for fault tolerance
- Jobs are rerun if a worker fails

Evaluation

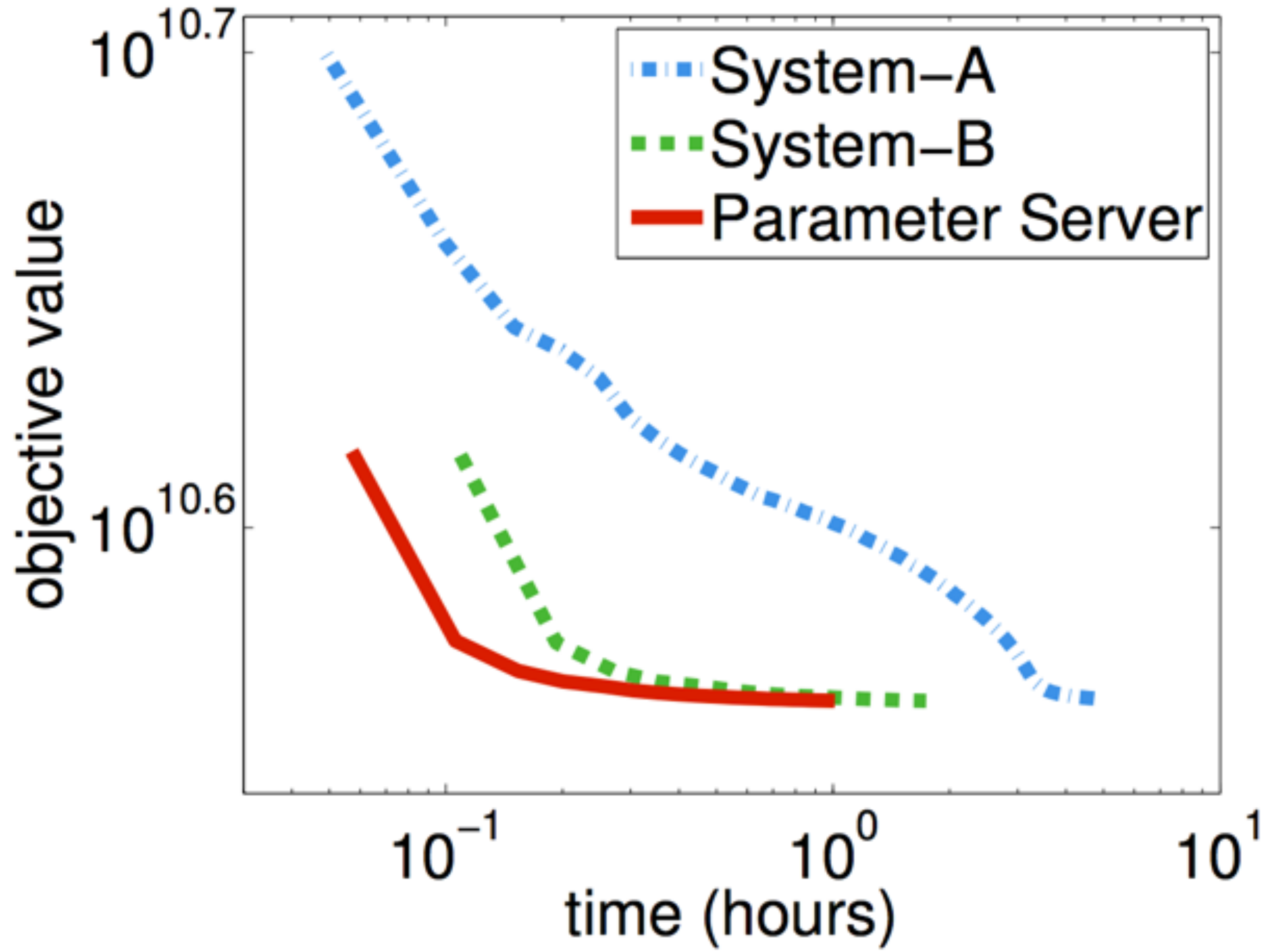


Figure 9: Convergence of sparse logistic regression. The goal is to minimize the objective rapidly.

Evaluation

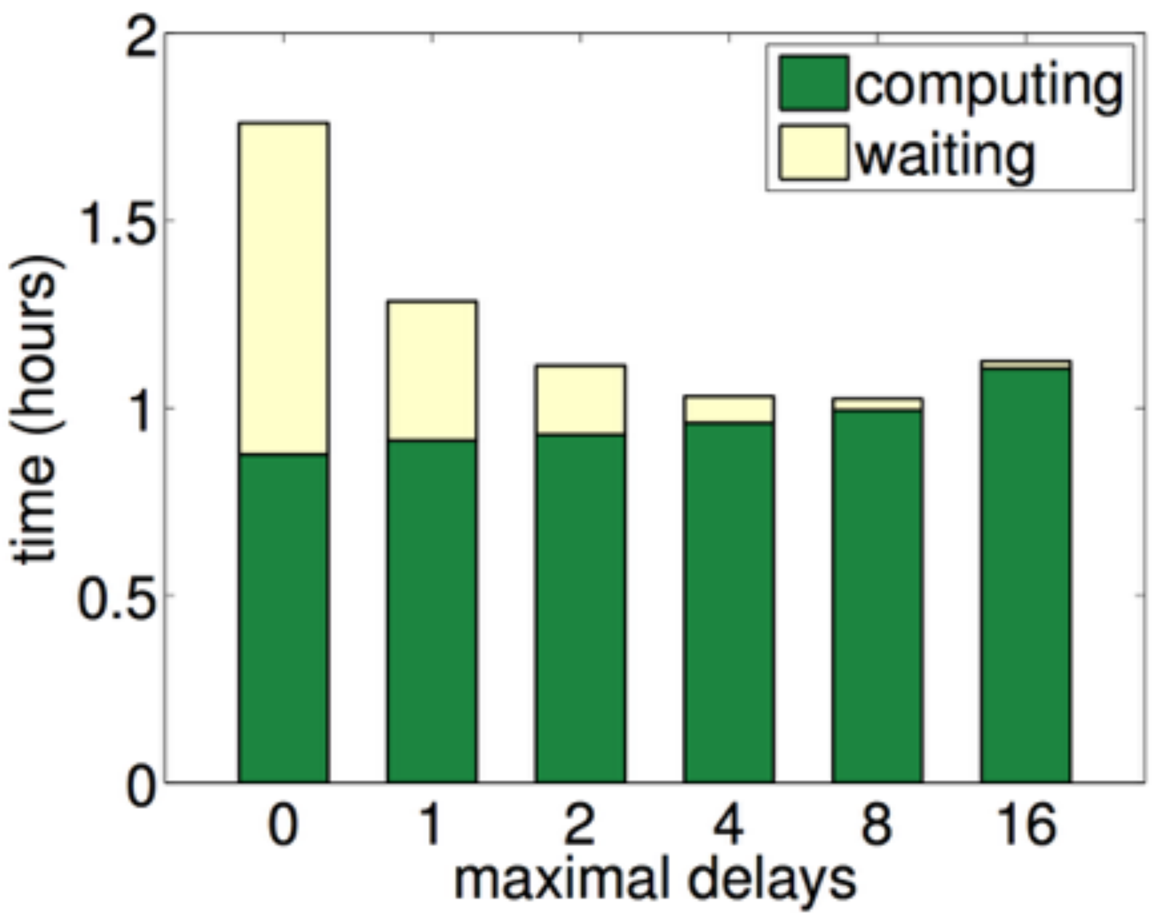


Figure 13: Time a worker spent to achieve the same convergence criteria by different maximal delays.

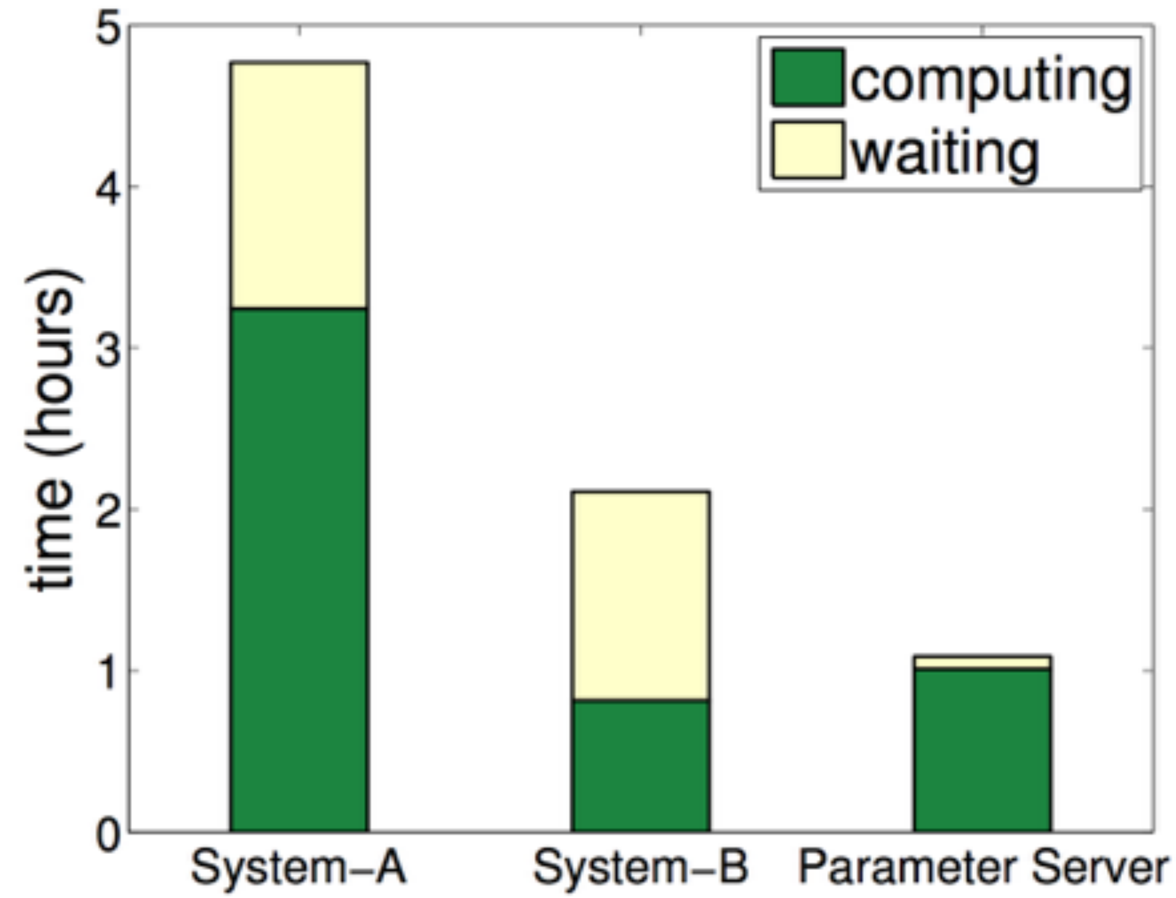


Figure 10: Time per worker spent on computation and waiting during sparse logistic regression.

Limitations

- Evaluation was done on specially designed ML algorithms
  - Distributed regression and distributed gradient descent
  - How fast is it on a sequential algorithm?
  - Count-Min Sketch is trivially parallelizable
- No neural networks evaluated?

Future Work

- What happens to sequential ML algorithms?
- Synchronization cost ignored, rather than resolved
  - Where are the bottlenecks of synchronization?
  - Lots of waiting time, but on what resource(s)?