PETUUM: A NEW PLATFORM FOR DISTRIBUTED MACHINE LEARNING ON BIG DATA

-Ajay Deshpande
What is Petuum?

• Petuum is a distributed machine learning framework.
• It aims to tackle running ML programs at scale:
  • Big Data: very large data.
  • Big Model: many parameters.
• Programming interfaces available for C++ and Java(limited).
• Key components:
  • Bösen: a bounded-asynchronous distributed key-value store.
  • Strads: a dynamic scheduler.
  • Poseidon: a distributed GPU deep learning framework.
Inspiration

• Research to production gap.
• Implementations of specific ML algorithms
  • YahooLDA, Vowpal Wabbit, Torch, etc.
  • Specialized implementations of some ML algorithms.
• Platforms that allow to write ML programs:
  • Hadoop, Spark, GraphLab, etc.
• Tradeoff on efficiency, correctness, programmability and generality.
Observation

• ML programs defined by an explicit objective function over data (e.g., likelihood, error loss, graph cut), and the goal is to attain optimality of this function

• Common Iterative convergent view of an ML algorithm:

\[ A^{(t)} = F(A^{(t-1)}, \Delta_{\ell}(A^{(t-1)}, D)) \]
Types of parallelism

• Data and model too big to fit into memory.

• Data parallelism:

\[ A^{(t)} = F(A^{(t-1)}, \sum_{p=1}^{P} \Delta (A^{(t-1)}, D_p)). \]

• Model parallelism:

\[ A^{(t)} = F \left( A^{(t-1)}, \{ \Delta (A^{(t-1)}, S_p^{(t-1)}(A^{(t-1)})) \}_{p=1}^{P} \right). \]
Fundamental properties for fast convergence:

- Error tolerance - iterative-convergent algorithms are often robust against limited errors in intermediate calculations.
- Dynamic structural dependency - during execution, the changing correlation strengths between model parameters are critical to efficient parallelization.
- Non-uniform convergence – the number of steps required for a parameter to converge can be highly skewed across parameters.
Petuum framework:

- **Parameter Server System (Bosen):**
  - A distributed shared memory interface.
  - Bounded-asynchronous consistency model.

- **Scheduler (Strads):**
  - Fine-grained control over the parallel ordering of model-parallel updates.

- **Workers:**
  - Runs the parameter update in parallel.
  - Model synchronization takes place at the same time.
Async vs sync data parallelism:

- For synchronous communication, as in the BSP (Bulk Synchronous Parallel) approach (used in MapReduce), the workers synchronize state and exchange messages after each round.
  - This is slow, but accurate.
- In asynchronous communication, data might get lost.
  - This is usually fast.
Bösen

- Stale Synchronous Parallel:

  - Fastest and the slowest workers not allowed to drift >s clocks apart.
  - Guaranteed to receive all updates from all workers computed at and before iteration c − s − 1
  - Determining the value of s requires domain expertise.
Strads

• Scheduler interface for model parallel programming.

• Exploits Structural dependencies and non-uniform convergence.

• Through the user-defined scheduling function: schedule().

• Common patterns for schedule:
  • Fixed schedule, dependency aware, prioritized scheduling.

• Overlap schedule computation with worker computation for speed.
Sample program structure:

```c
// Petuum Program Structure

schedule() {
    // This is the (optional) scheduling function
    // It is executed on the scheduler machines
    A_local = PS.get(A) // Parameter server read
    PS.inc(A, change) // Can write to PS here if needed
    // Choose variables for push() and return
    svars = my_scheduling(DATA, A_local)
    return svars
}

push(p = worker_id(), svars = schedule()) {
    // This is the parallel update function
    // It is executed on each of P worker machines
    A_local = PS.get(A) // Parameter server read
    // Perform computation and send return values to pull()
    // Or just write directly to PS
    change1 = my_update1(DATA, p, A_local)
    change2 = my_update2(DATA, p, A_local)
    PS.inc(A, change1) // Parameter server increment
    return change2
}

pull(svars = schedule(), updates = (push(1), ..., push(P)) ) {
    // This is the (optional) aggregation function
    // It is executed on the scheduler machines
    A_local = PS.get(A) // Parameter server read
    // Aggregate updates from push(1..P) and write to PS
    my_aggregate(A_local, updates)
    PS.put(A, change) // Parameter server overwrite
}
```

Figure 4: Petuum Program Structure.
Performance
Performance
Performance
Conclusion

• Provides a decent framework for implementing ML algorithms.
• Would probably require decent ML and domain expertise to write a fast schedule.
• Continuous development and open source.
References

• http://www.petuum.com/

• http://petuum.github.io/

• Installation instructions:

• Bosen reference manual: