Optimizing Software Performance with IL{P}P: A novel approach

Enrico Zanardo

Researcher @ BEEZ

Overview

Context and Motivation

Optimizing Software Performance & Machine Learning Models

- Distributed Systems & Datasets [RQ1];
- Parallel computation power [RQ1];
- Explainable AI (ILPP) [RQ2];

Context and Motivation

[RQ1]: Is it possible to efficiently train machine learning models in a decentralized and distribute network without owning the entire dataset?

- Single-node machine learning model training is difficult;
- The dataset determines the accuracy of machine learning models;

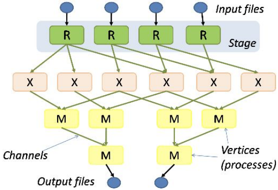
[RQ2]: Is it possible to turn a machine learning model human readable? And viceversa?

- We need explanations;
- Machine / Human Bias (in Datasets, Parameters ...);

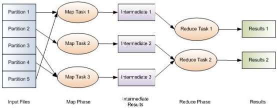
Related work [RQ1]

Distributed Execution Engines & Distributed data-flows

- Existing distributed execution engines (MapReduce and Dryad) were inefficient for iterative algorithms.
    - They often rely on **data replication** to ensure fault tolerance. In iterative algorithms, the same data is processed multiple times, leading to unnecessary replication and storage overhead.
    - They were **not specifically designed for iterative algorithms** and complex, multi-stage workflows, which are common in many data processing and machine learning applications.



MapReduce job



Dryad job

Related work [RQ1]

We add and tested iteration capabilities to MapReduce by using libraries and frameworks such as:

- **CGL-MapReduce** (Cyclic Graph Library for MapReduce: Introduces the concept of a cyclic graph to express iterative dependencies)
- **HaLoop** (reduce the overhead of starting new MapReduce jobs for each iteration)
- **Apache Mahout** (leverage MapReduce for distributed computation)

They differ from our approach in the following ways:

- Do not provide transparent fault tolerance.
- Do not support task dependency graphs.
- Job latency is increased by consecutive iterations.

Related work [RQ1]

We tried providing data-dependent control flows:

- Pregel (Google's execution engine)
  - Composition of multiple computations not possible.
  - Only operates on a single dataset.
- Piccolo (data-centric programming model)
  - Does not provide transparent scaling.
  - Fault tolerance involves checkpointing.

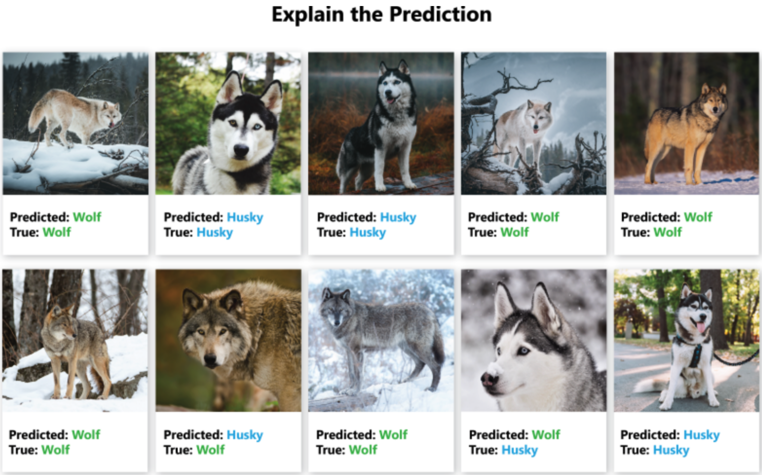
The GDPR's Right to receive an explanations! [RQ2]

General Data Protection Regulations [European Union 2018]

Art.22 - The data subject shall have the right **not** to be subject to a decision based solely on automated processing, including profiling, which produces legal effects concerning him or her or similarly significantly affects him or her.

Related work [RQ2]

The Husky-wolf problem



Related work [RQ2]

Understand why the black box made that choice?

Related work [RQ2]

Can we do better?

Yes ..

- Use IL{P}P that combines logic programming and data mining.
- Enabling automated and data-driven decision-making in software optimization.
- ILP leverages data to derive logical rules and dependencies between tasks within software.

Example problem [RQ2]

Our solution, like many others in the field needs to have:

- An examples file;
- A background knowledge (BK) file;
- A bias file;

Example problem [RQ2]

Example file:

```
pos(grandparent(ann,amelia)).
pos(grandparent(steve,amelia)).
pos(grandparent(ann,spongebob)).
pos(grandparent(steve,spongebob)).
pos(grandparent(linda,amelia)).
neg(grandparent(amy,amelia)).

```

BK file contains other information about the problem:

```
mother(ann,amy).
mother(ann,andy).
mother(amy,amelia).
mother(linda,gavin).
father(steve,amy).
father(steve,andy).
father(gavin,amelia).
father(andy,spongebob).

```

Example problem [RQ2]

A bias file contains information necessary to restrict the search space:

```
max_clauses(4).
max_vars(4).
max_body(3).

```

The output will be:

```
grandparent(A,B):-mother(A,C),father(C,B).
grandparent(A,B):-father(A,C),mother(C,B).
grandparent(A,B):-father(A,C),father(C,B).
grandparent(A,B):-mother(A,C),mother(C,B).

```

% Precision:1.00, Recall:1.00, TP:5, FN:0, TN:1, FP:0 that is also **Explainable!**

Results

- Dynamic control flow
- Dynamic task dependencies
- Transparent tasks fault tolerance
- Transparent network scaling
- Logically explain AI models

Conclusion

- This approach enables clients to run *"explainable"* iterative and recursive algorithms in a highly parallelized manner with transparent fault tolerance and transparent scaling
- At the moment it is designed for coarse-grained (simplified) parallelism across large data sets
- For fine-grained parallelism, work-stealing schemes are better:
  - If data fits into RAM, Piccolo is more efficient.
  - If jobs share a lot of data, OpenMP is more appropriate.
  - For better scalability and performance use MPI.

Thanks a lot!