Presentation
Gilbert: A sparse linear algebra environment

Till Rohrmann
till.rohrmann@campus.tu-berlin.de

Technische Universität Berlin

April 2, 2014

Table of Contents

1 Motivation

2 Gilbert

3 Approach

4 Current state

5 Outlook

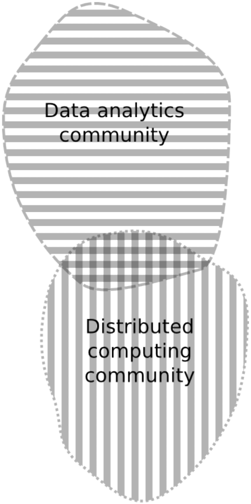
6 Related Work

Big data analytics

- Gathered data grows exponentially
- Obtain new insights
- Analytic methods have to scale up ⇒ Parallelization
- Methods developed within linear algebra systems
- Explicit parallelization tedious and error-prone

Distributed computing and data analytics

- Experts familiar with both domains countable
- Laborious to become acquainted with new domain
- Huge existing code base
- Can't we bring both worlds together?
- Solution: Gilbert



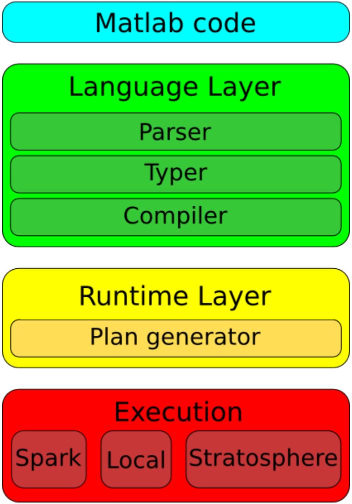
Gilbert

- Distributed sparse linear algebra system
- Matlab frontend for distributed computing frameworks
- Allows to almost seamlessly move from local execution to distributed execution in a heterogenous environment
- Enable machine learning and data analytics algorithms to be run distributedly

Approach

1. Compiling Matlab code into intermediate representation
2. Apply optimizations indenpendently of runtime specific system
3. Compiling intermediate representation into runtime specific format

System architecture



Language

- Matlab-like language
- Support of basic linear algebra operations
- Some built-in functions, repmat, linspace, pdist2
- Loop support with static and dynamic termination criterion
- Language expressive enough to support variety of algorithms, Pagerank, K-means, NNMF

```
A'*B

f = @(x) x.^2.0

eps = 0.1

c = @(p,c) norm(p-c,2) < eps

fixpoint(1/2, f, 10, c)
```

Compilation: Intermediate format

- Scala's combinator parsing tool box
- Powerful enough for our language
- Matlab is dynamically typed
- Execution on Stratosphere requires type knowledge at compilation time
- Hindley-Milner type inference algorithm to infer types and dimensions

Example: Parsing and typing

Input

```
A = ones(2,10);
B = eye(10,3);
A*B
```

Parsed and typed

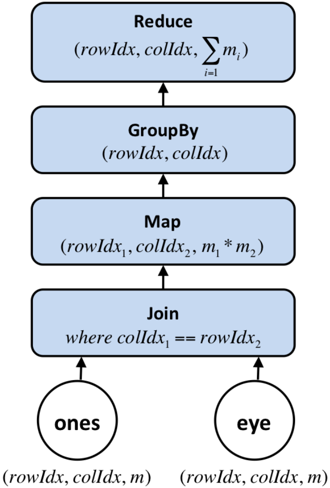
```
A = ones(2,10):MatrixType[Double,2,10];
B = eye(10,3):MatrixType[Double,10,3];
A*B:MatrixType[Double,2,3]
```

Example: Intermediate code

Compiled

```
MatrixMult(
   ones(
      IntValue (2),
      IntValue(10)
   ),
   eye(
      IntValue (10),
      IntValue(3)
   )
)
```

Example: Stratosphere execution plan



Current state

- Language layer implemented and working
- Runtime layer: Execution on Stratosphere with iteration and convergence support
- Implemented algorithms: PageRank, NNMF and K-means

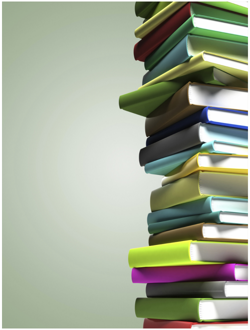
Live demonstration

Outlook

- Typing system with constraints, similar to Haskell typing system with type class support
- System evaluation: Runtime, scalability
- Comparison to specialized algorithms
- Optimizations for the intermediate representation

Related Work

- SystemML [2]
  - Higher-level language for linear algebra
  - Compiled to MapReduce jobs
- Apache Mahout [1]
  - Specialized implementations of algorithms of various kind
  - No native linear algebra support
- Pegasus [3]
  - Generalized iterative matrix vector multiplication
- Spark [4]
  - MapReduce extension



Bibliography I

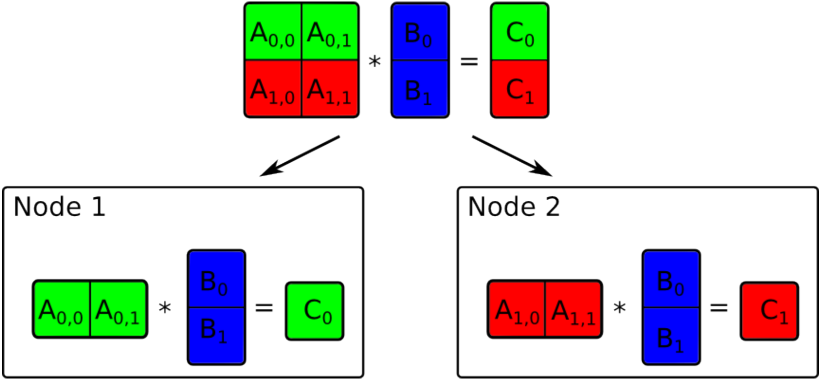
Apache. *Apache Mahout*. Cited January 9th 2014. 2011.

A. Ghoting et al. "SystemML: Declarative machine learning on MapReduce". In: *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE. 2011, pp. 231–242.

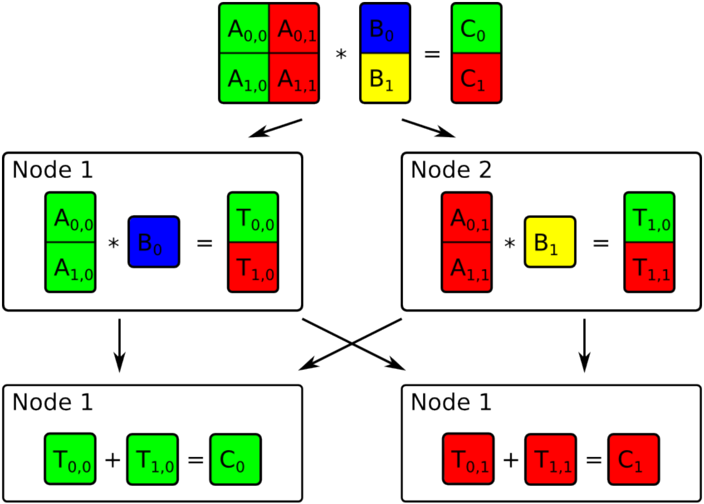
U Kang, C. E. Tsourakakis, and C. Faloutsos. "Pegasus: A peta-scale graph mining system implementation and observations". In: *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE. 2009, pp. 229–238.

M. Zaharia et al. "Spark: cluster computing with working sets". In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 2010, pp. 10–10.

Replication based matrix multiplication



Cross product based matrix multiplication



Communication costs

- Matrix A blocked with $M \times K$ blocks and B blocked with $K \times N$ blocks
- Network and IO costs are dominant
- $costs_{RMM} \in O\left(network(|A| \cdot N + |B| \cdot M) + io(|A| + |B| + |C|)\right)$
- $costs_{CPMM} \in O\left(network(|A| + |B| + r \cdot |C|) + io(|A| + |B| + |C|)\right)$ with r being the number of reducer

Stratosphere execution plan

- Assuming row-wise partitioning of matrix A
- Execution plans for RMM and CPMM identical
- Differ only in chosen strategy for join operator
- RMM: Hybrid-hash join
- CPMM: Sort-merge join
- Stratosphere's optimizer chooses right plan