Implementing Dimensionality Reduction in Apache Flink Using the Lanczos Algorithm

Nik Hille and Fridtjof Sander
{nik.hille, fridtjof.sander}@campus.tu-berlin.de

1. Problem Statement

Implement a generic tool using Apache Flink

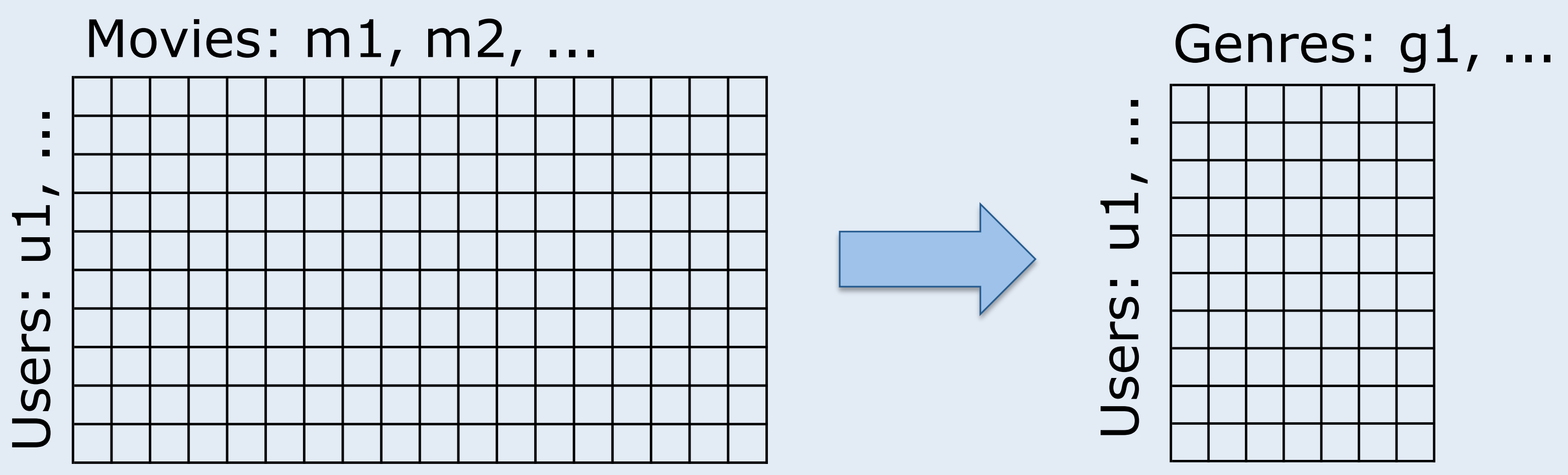
- Matrix ⇨ Tool ⇨ Singular Value Decomposition

Not scoped on analizing a specific dataset

2. Introduction

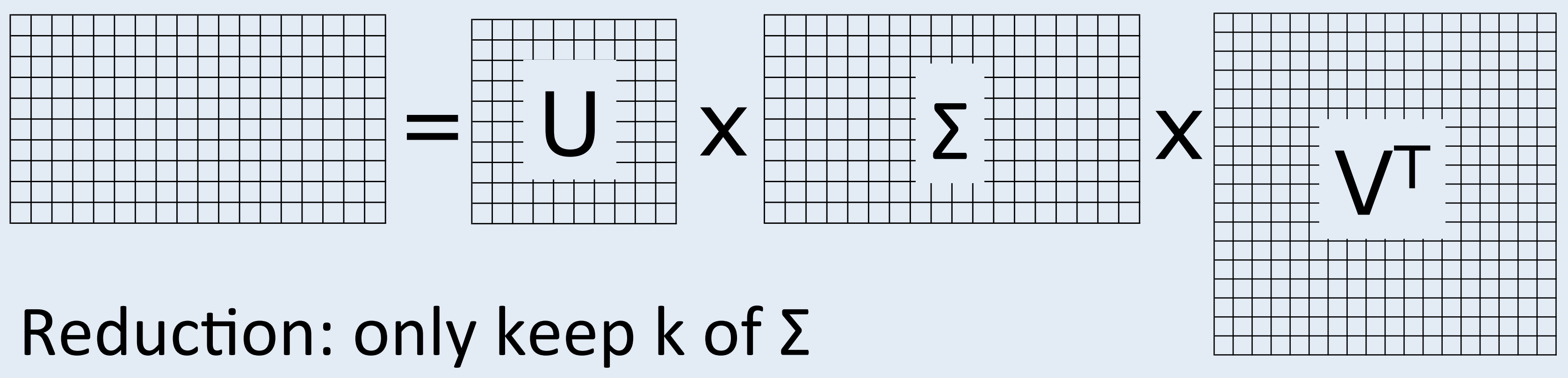
Goal of dimensionality reduction → compress data

- syntactically (bytes) & semantically (concepts)



Through Singular Value Decomposition (SVD)

- $A = U \Sigma V^T$
- U, V orthonormal eigenvectors of AA^T, A^TA
- Σ eigenvalues of U, V in descending order



Reduction: only keep k of Σ

Numerical Calculation

- Lanczos(A) = (U, TriDiag)
  - TriDiag is a symmetric, tridiagonal auxilary matrix
- EigenDecomposition(TriDiag) = (Σ, V)

Since TriDiag is small, only parallelize Lanczos

3. Lanczos

Iterative Algorithm – *not* embarassingly parallel
In each Iteration from 1 to k produce next

- $u_i = A \times u_{i-1}$ → U (plus orthonomalization)
- u_0 is random / uniformly chosen
- (a_i, b_i) → TriDiag (a diagonal, b off diagonal)

The only paralallizable part is $A \times u_{i-1}$

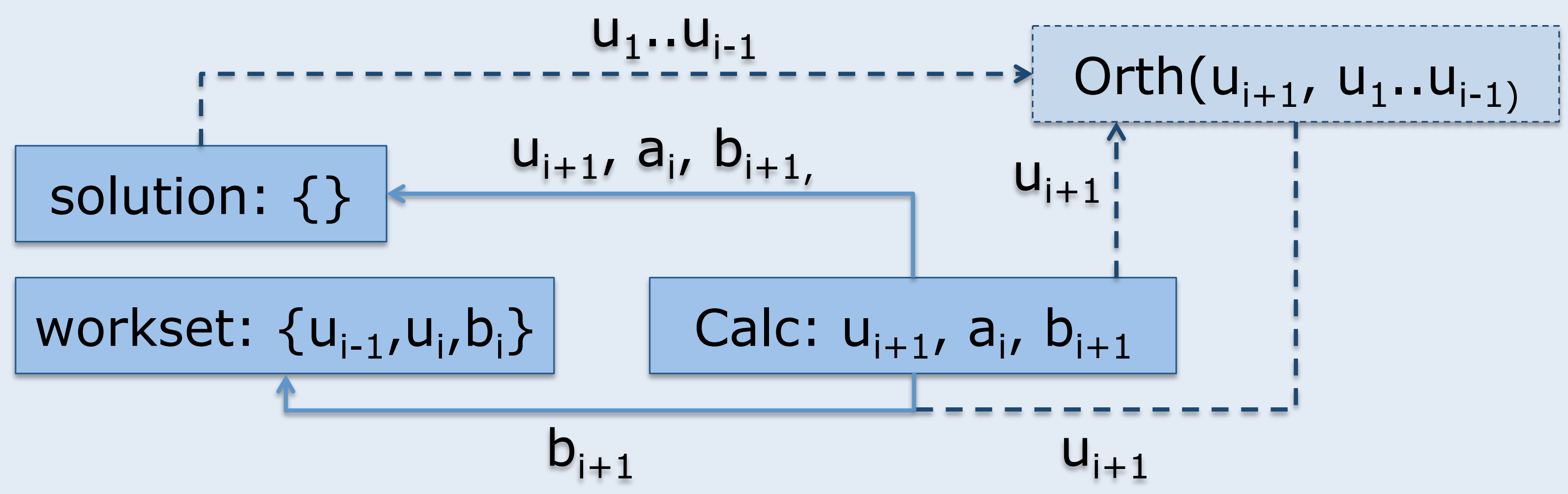
4. Approaches

1. Delta Iterations
   - data flow from scratch optimizable by Flink
2. Exploiting Mahout‘s Hadoop-Lanczos solver
   - Generic implementation (using interfaces)
   - By implementing interfaces with Flink
   - Each step one Hadoop-Job
3. Iterative dataflow construction
   - Less optimizable by Flink

5. Implementations

1. Delta Iterations

- Requires to hold Basis + TriDiag in one DataSet
- Tuple4(id,row,col,val) → „LanczosPlasma“


$u_1..u_{i-1}$
Orth(u_{i+1}, $u_1..u_{i-1}$)
u_{i+1}, a_i, b_{i+1},
u_{i+1}
solution: {}
workset: {u_{i-1},u_i,b_i}
Calc: u_{i+1}, a_i, b_{i+1}
b_{i+1}
u_{i+1}

„Nested iterations are currently not supported“

2. Exploiting Mahout‘s Lanczos solver

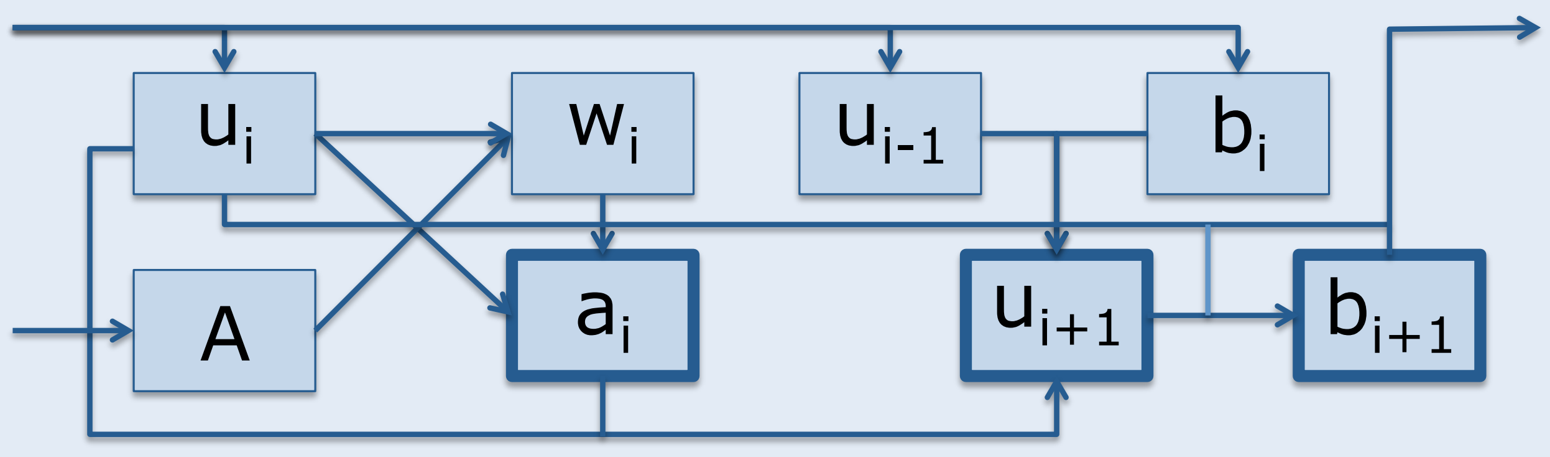
- relying on flat data types:

```
double alpha = currentVector.dot(nextVector);
```

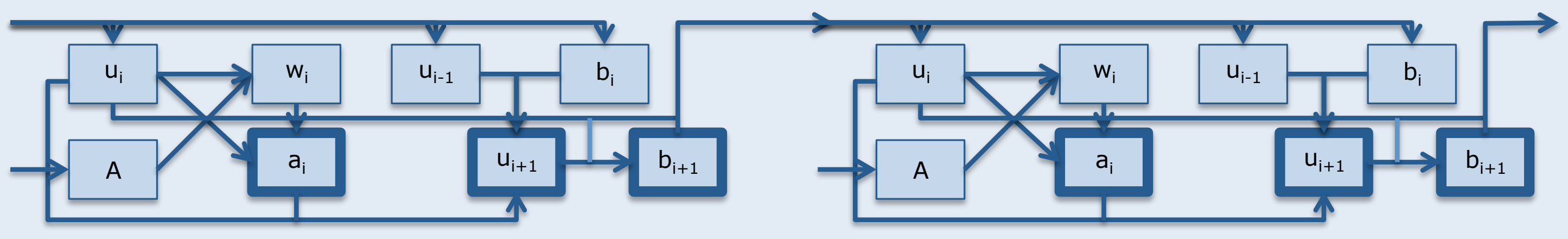
- Would require materialization in each iteration

3. Iterative dataflow construction

- One iteration step modelled as data flow



- Iteration is modelled by concatinating steps



- Compiler/Optimizer overloaded at 6 iterations

6. Results

- Idiomatic solution (delta iteration) not possible with Apache Flink‘s programming model (yet)
- Iterative data flow construction practically infeasable due to resulting DAG complexity
- Building on Mahout initially discarded due to intermediate materialization
  → only remaining approach that could work

7. Conclusion

- No satisfying solution possible with Flink 0.8.0
- Workaround by intermediate materialization is not expected to perform significantly better than Mahout + Hadoop
- Shows limitations of Apache Flink for implementing iterative algorithms
- Wait until Flink supports nested iterations