MadLINQ: Large-Scale Distributed Matrix Computation for the Cloud

Zhengping Qian, Xiuwei Chen, Nanxi Kang, Mingcheng Chen, Yuan Yu, Thomas Moscibroda and Zheng Zhang

Presented by Brett Lagerwall

University of Cambridge

February 24, 2013

Motivation

Many existing methods for performing large-scale distributed matrix computations fall short.

MPI-based solutions:

- Requires understanding of low level MPI primitives.
- Entire problem must be maintained in memory for efficiency.

MapReduce-based solutions:

- Difficult to program.
- No reduce operation can proceed until all maps are finished.

Overview – Research Goals

The two goals of MadLINQ are:

- To provide a scalable, efficient, fault-tolerant, usable matrix computation engine.
- To integrate this engine into a general purpose parallel computing platform.

Overview – Features

They introduce:

- A new programming model
- Fine-grained pipelining
- A new fault tolerance mechanism
- Optimizations (e.g. auto-switching of block representation)

Methodology and Contribution – Programming Model

The new programming model increases usability:

- Can handle dense and sparse matrices.
- Allows for easy representation of graph algorithms.
- Tile algorithms have an intuitive representation.

Methodology and Contribution – Language Integration

They provide a unified programming model of MadLINQ/DryadLINQ/C#.

Benefits of this include:

- MadLINQ functionality can be encapsulated in a large C# application.
- Can handle both linear algebra and relational algebra.
- Interoperability in a general purpose computing model.

Fine-grained pipelining intends to improve efficiency. It works as follows:

- Each vertex must produce data at a finer granularity (block).
- Tiling algorithm must work at the block level.
- Computation engine must be able to output partial results.

Fine-grained pipelining has a number of performance benefits.

Methodology and Contribution – Fault Tolerance

Previous fault tolerance mechanisms do not work.
MadLINQ uses lightweight dependency tracking as follows:

- Assumed that each set of output blocks can derive the set of input blocks needed to compute it.
- Query downstream vertices discovering the set of blocks it still requires.
- Process can be done recursively.

Methodology and Contribution – Additional Features

Extra features added were:

- Auto-switching of block representation – increases scalability.
- Pre-loading ready vertices onto occupied nodes which are about to finish.
- Adding order preference for requesting vertices.

Results – CPU Utilization

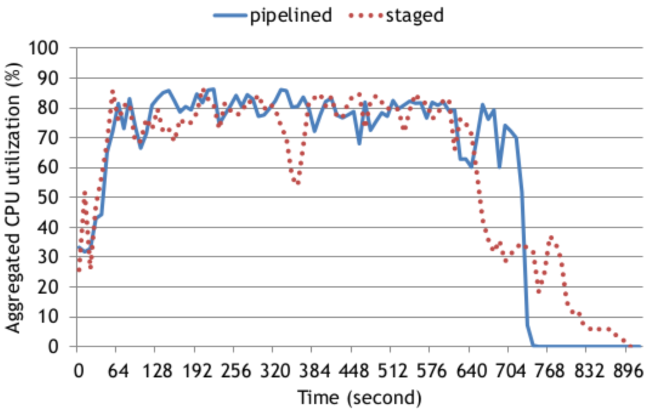

Figure: Source: Qian et al. (2012)

Results – Network Traffic

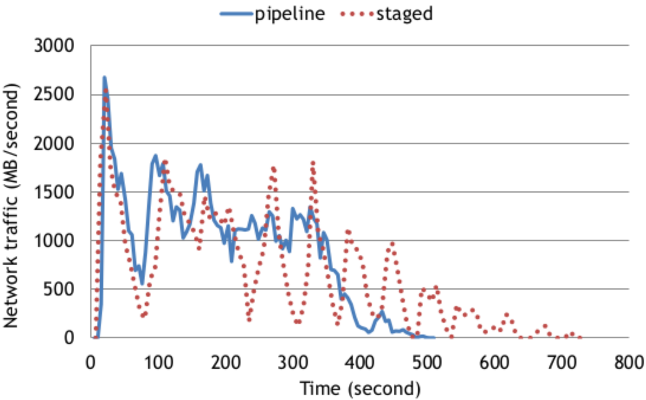

Figure: Source: Qian et al. (2012)

Results – Fault Tolerance

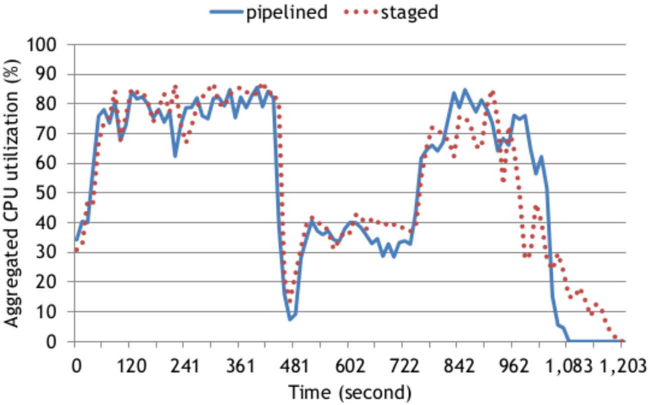

Figure: Source: Qian et al. (2012)

Results – Comparison of Performance

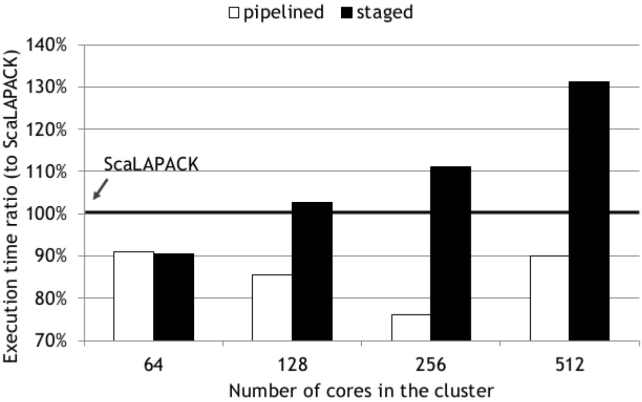

Figure: Source: Qian et al. (2012)

Related Work

Three approaches have previously been used for large-scale matrix computation:

- HPC solutions
- MapReduce-based solutions
- Direct DAG execution

Related Work

Examples of the three approaches are:

- ScaLAPACK is an example of an HPC solution. Compared to MadLINQ, it has weaker scaling and fault tolerance.
- HAMA performs matrix computations using MapReduce. Constrained by the semantics of MapReduce.
- DAGuE is an architecture for scheduling DAGs. It has no fault tolerance and its parallelism is bound by the tile level.

Future Work

Ideas which could be investigated in the future are:

- Auto-tiling
- Dynamic re-tiling and re-blocking
- Better handling of sparse matrices

Criticism

- There is no mention of fault tolerance, backups or checkpointing for the centralized scheduler.
- The fault tolerance does not handle non-determinism (edges being randomly added to a graph).
- There is no evidence or explanation of how stragglers may be handled.

Criticism

- No insight is given into how one may choose parameters for execution.
- How costly is the action of switching between matrix representations in their auto-switching algorithm? Does the performance gain overcome the cost?
- More studies should be done into the real-world applicability of their pipelining and fault tolerance mechanisms.

Conclusion

To conclude:

- The authors have demonstrated a need for a new approach.
- They also achieve their original goals – producing a scalable, efficient, fault-tolerant, matrix computation engine which is integrated into a general purpose framework.
- There are smaller issues which need to be solved and more real-world tests need to be done.