Omni Graph Mining: Graph Mining using RDBMS

Jin Kyu Kim
Dept. of Computer Science
CMU
jinkyuk@cs.cmu.edu

Alex Degtiar
Dept. of Computer Science
CMU
adegtiar@andrew.cmu.edu

Jay Yoon Lee
Dept. of Computer Science
CMU jaylee@andrew.cmu.edu

The motivations

- Huge amount of Graph Data is stored in RDBMS.
- Not all graph data is in billion scale.
- Query engine has been extremely optimized.

  (Automatic management of index, Disk I/O optimization, well implemented operations such as join, sorting)
- People love SQL!
  - Readily available and heavily used in enterprises
  - sets standard from its popularity and easiness to learn.

We think that there would be ***"sweet spot"*** where RBDMS can outperform other frameworks such as Map Reduce and GraphChi.

Target Algorithms

We implemented six algorithms using SQL and plpgsql (Procedural Programming Language by PostgreSQL)

Algorithms	Description
Eigen values	Finds the eigenvalue decomposition of the graph's adjacency matrix.
Diameter	Measures the distance of the furthest-apart in the graph
Connected Components	Partitions the graph into groups of nodes mutually connected via their edges
PageRank	Measures the relative importance of nodes via the nodes that link to it.
FastBP	Find an efficient way to solve inference problems based on local message passing
Degree Distribution	Find the number of in- and out- edges among nodes of the graph.

Common Matrix Operations (SQL)

- Many algorithms distilled into core set of matrix operations
- Matrix operations are easily expressed in SQL
- SQL execute sparse matrix efficiently via optimized joins, etc.

y = y + ax (*saxpy* vector update)

```
UPDATE y
    SET y.val=y.val+a*x.val
    FROM x
    WHERE y.i = x.i;
```

A*y matrix-vector multiplication (*SMVM*)

```
SELECT A.row,
SUM(A.weight*y.value)
    FROM A, y
    WHERE A= y.node
    GROUP BY A.row;
```

Sum(|y|)^1/p (vector LP norm)

```
SELECT (sum((y.val)^p)^1/p
    FROM y;
```

A * B matrix-matrix multiplication (*SMMM*)

```
SELECT A.i, B.j,
SUM(A.val*B.val)
    FROM A, B
    WHERE A.j = B.i
    GROUP BY A.i, B.j;
```

PageRank and Eigensolver

- **PageRank**: Update is performed by SMVM

Input	E: Edge list (Pair of Src, Dst), V: Vertex list and d: damping factor
SQL	BUILD column normalized adjacency matrix: A INSERT INTO product(node, val) SELECT A.dest, SUM(A.weight * pagerank.val) FROM A, pagerank WHERE A.src = pagerank.node GROUP BY A.dst; UPDATE pagerank_new SET val = d*product.val + (1-d)/|V| IF L1(pagerank_new – pagerank) < delta EXIT

- **Eigensolver**: Uses *Saxpy*, *Smvm*, *Smmm*, *dotProduct*, *etc…*

Input	A: Adjacency matrix(i, j, val), b: random vector, m: num steps, e error thresh
Summ	1. Until convergence or max iteration: 1. Generate a new basis vector 2. Orthogonalize against previous two 3. Update tridiagonal matrix using scalar components of these vectors 4. Selectively orthogonalize against all previous vectors 2. Perform eigen decomposition on tridiagonal matrix for eigenvalues and Q 3. Compute eigenvectors using Q and the basis vectors

Degree Dist. and Diameter in SQL

- **Degree**: Update is performed by Scan & Aggregate of SQL

Input	Input) E: Edge list (Pair of Src, Dst) and V: Vertex list

SQL	Update V set incnt = (select count(*) from E where E.dst = V.id); Update V set outcnt = (select count(*) from E where E.src = V.id); Update V set sum = V.incnt + V.outcnt;

- **Diameter**: Radius of each vertex is updated by Join and user defined function and aggregate. (HADI based)

SQL	Insert into New V select E.src, BIT-OR(FMB1), …. , BIT-OR(FMB32) From E, V Where E.dst = V.id Group By E.src; *BIT-OR: User defined BIT-OR aggregate *FMB: Flajolet-Martin Bit string to estimate the size of set * Iterate until all bit string are stabilized. In next iteration, New V is used as V.

Other Frameworks

- Pegasus: Peta-Scale Graph Mining framework with MR

  Pegasus considers graph mining algorithms as a sequence of matrix multiplication

  - GIM-V library performs matrix multiplication in Hadoop
  - Graph mining algorithms are implemented using GIM-V Library.

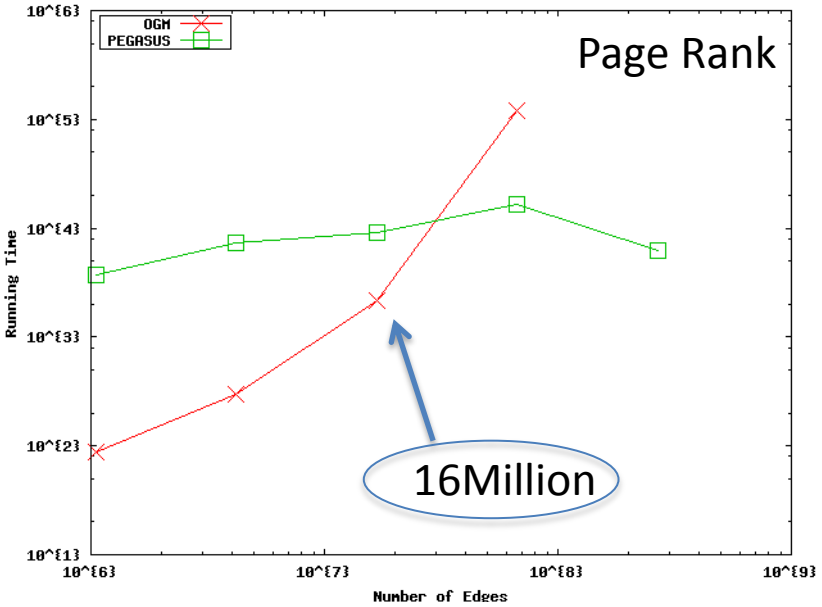
  For small data, MR framework could be a overkill.

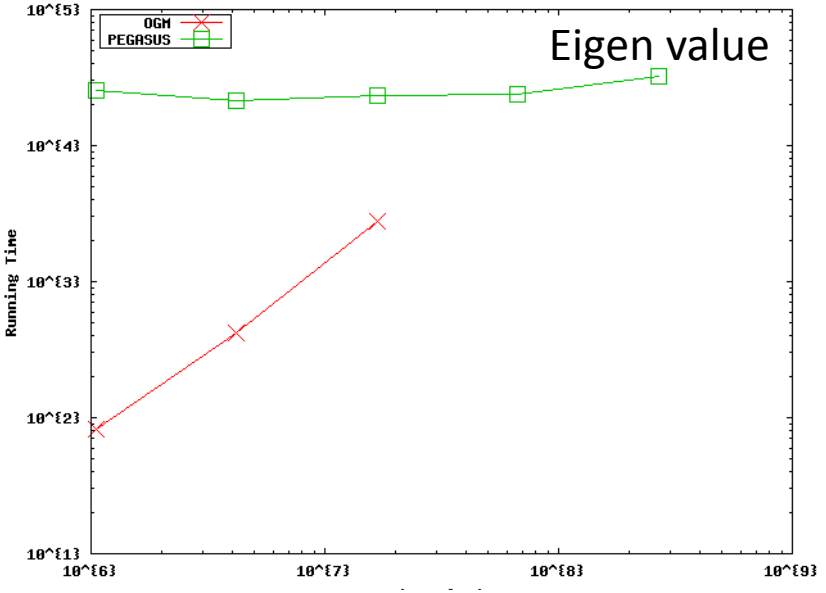
- GraphChi: A Disk based Graph Mining framework on single machine

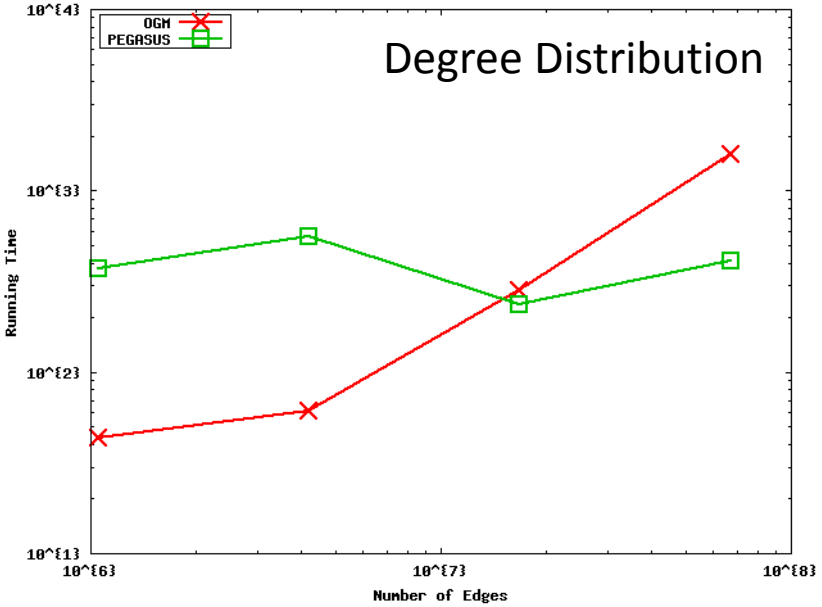
  - In preprocessing stage, edge and vertex lists are partitioned and sorted.
  - In computation stage, compute a sub-graph at a time. All information for processing a sub-graph are loaded into main memory by the limited number of bulk disk I/O.

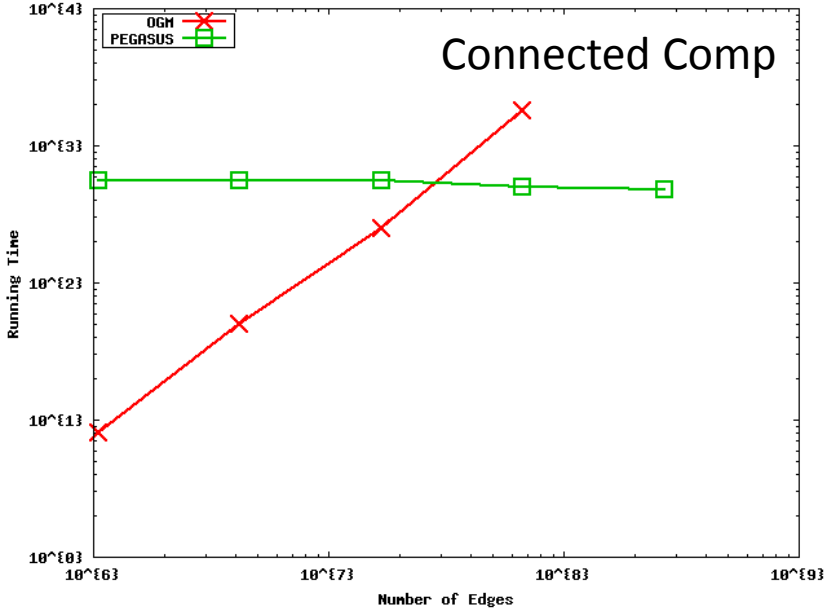
  GraphChi eliminates almost all random disk access.

Experimental Results



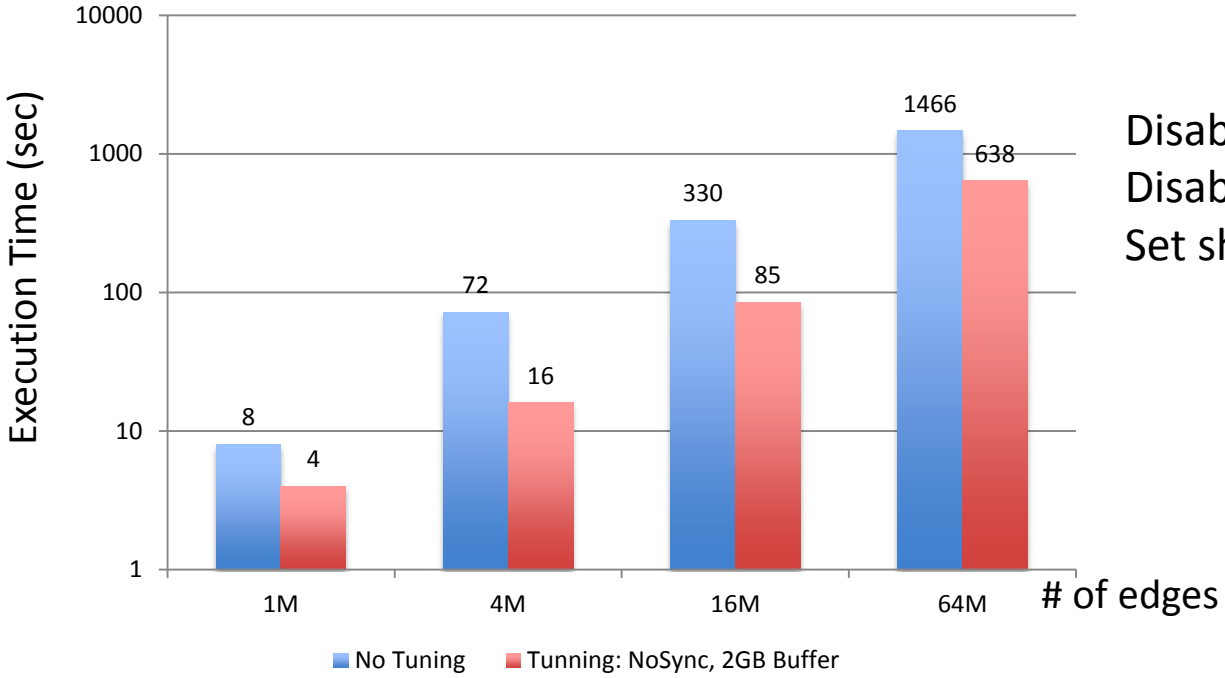






RBDMS Tuning

- Tuning of RDBMS is critical to improve the performance of SQL implementation.



Disable fsync
Disable synchronous commit
Set shared buffer to 2GB

- Disabling consistency and atomicity related options is helpful to improve the performance.

Conclusion & Future Work

- **Discussion about *"Sweet Spot"***
  - SQL-single machine outperforms Hadoop 64 machines for up to 16 million edges data set.
  - GraphChi-single machine is more than 10 times faster than the SQL for test cases.

- **Future Works**
  - More tuning on RDBMS that compromises consistency constraints for better performance
  - Pre-sorting of input data could improve memory access locality of join operation.
  - Smart management of index structures.
    Updating or insert operation on indexed table could incur extra overhead.