

HAMA: An Efficient Matrix Computation with the MapReduce Framework

IEEE CLOUDCOM 2010 Workshop, Indianapolis, USA
Sangwon Seo, Edward J. Yoon / Apache HAMA TEAM

Introduction

- ▶ The volume of information is increasing as evolving modern science
- ▶ Data-intensive processing is required
 - ▶ MapReduce is one of the data-intensive programming model
- ▶ Massive matrix/graph computations are often used as primary functionalities
- ▶ Thus, we provide **easy-of-use tool** for data-intensive scientific computation known as HAMA

Apache HAMA

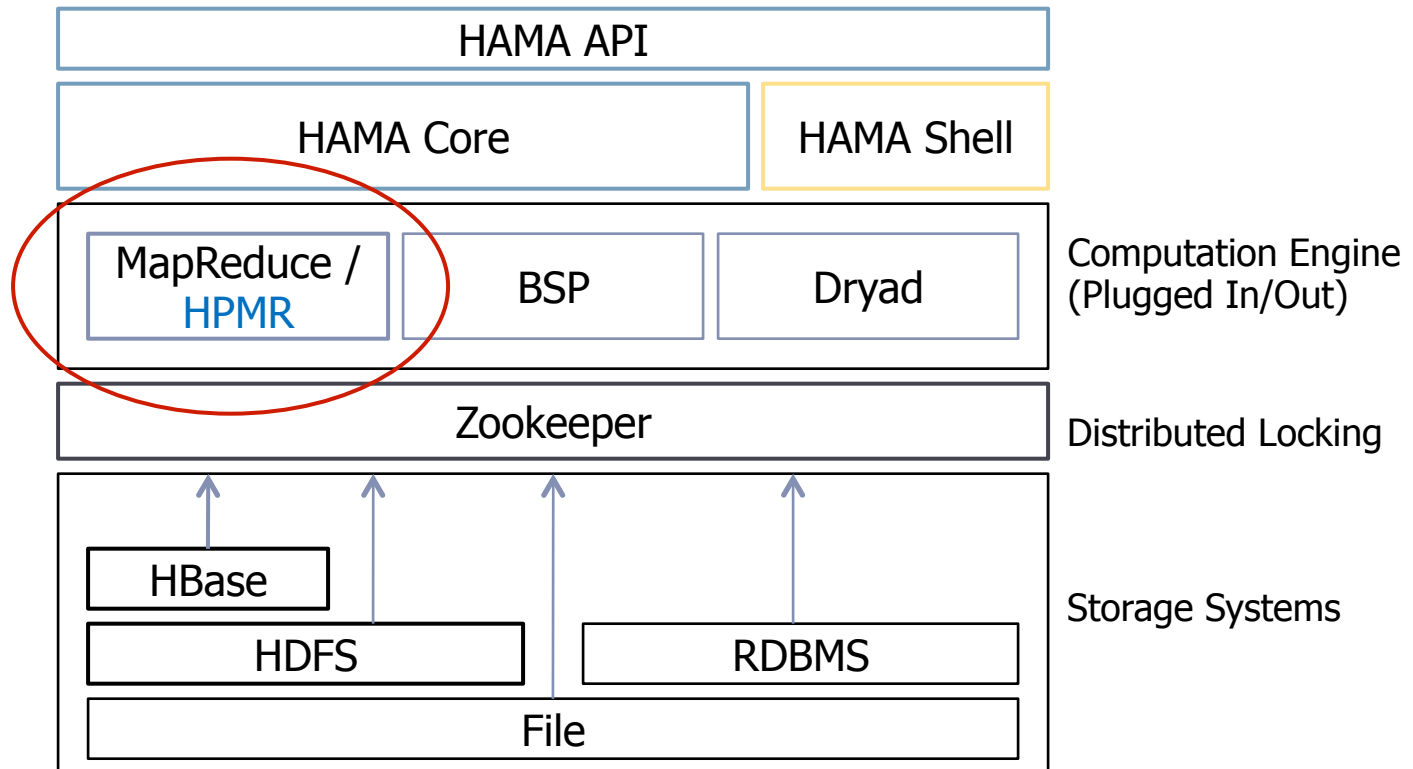
- ▶ Now incubating project in ASF
- ▶ Fundamental design is changed from **MapReduce** with matrix computation to **BSP** with graph processing
- ▶ Mimic of Pregel running on HDFS
 - ▶ Use zookeeper as a synchronization barrier

Our Focus

- ▶ This paper is a story about previous version of HAMA
- ▶ Only focus on matrix computation with MapReduce
- ▶ Shows simple case studies

The HAMA Architecture

- ▶ We propose distributed scientific framework called **HAMA** (based on HPMR)
 - ▶ Provide transparent matrix/graph primitives



Case Study

- ▶ With case study approach, we introduce two basic primitives with MapReduce model running on HAMA
 - ▶ matrix multiplication and finding linear solution
- ▶ And compare with MPI versions of these primitives

Case Study

▶ Representing matrices

- ▶ As a default, HAMA use HBase (No-SQL database)
 - ▶ HBase is modeled after Google's Bigtable
 - ▶ Column oriented, semi-structured distributed database with high scalability

Row	Column Families
matrix_row_index	column_vector
metadata	alias:name
	attribute:columns
	attribute:rows
	attribute:type
	eival:value
	eival:ind
	eivec:value
cache:value	

Table 2: HBase table schema for actual matrix.

Case Study – Multiplication (1/3)

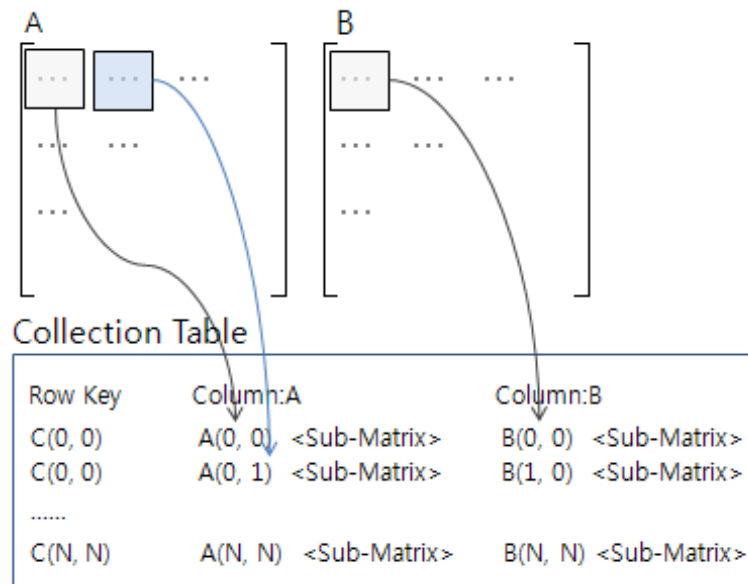
▶ Iterative approach (Algorithm)

```
INPUT: key, /* the row index of B */  
value, /* the column vector of the row */  
context /* IO interface (HBase) */  
void map(ImmutableBytesWritable key, Result value, Context context)  
{  
    double ij-th = currVector.get(key);  
    SparseVector mult /* Multiplication */ = new SparseVector(value).scale(ij-th);  
    context.write(nKey, mult.getEntries());  
}
```

```
INPUT: key, /* key by map task */  
value, /* value by map task */  
context /* IO interface (HBase) */  
void reduce(IntWritable key, Iterable<MapWritable> values, Context context)  
{  
    SparseVector sum = new SparseVector();  
    for (MapWritable value : values) {  
        sum.add(new SparseVector(value));  
    }  
}
```

Case Study – Multiplication (2/3)

- ▶ Block approach
 - ▶ Minimize data movement (network cost)



<reduce>
 ▶ $C_block(1,1) = \underbrace{A_block(1,1)*B_block(1,1)}_{<map>} + \underbrace{A_block(1,2)*B_block(2,1)}_{<map>}$

Case Study – Multiplication (3/3)

▶ Block approach (Algorithm)

```
INPUT: key, /* the row index of B */  
value, /* the column vector of the row */  
context /* IO interface (HBase) */  
void map(ImmutableBytesWritable key, Result value, Context context)  
{  
    SubMatrix a = new SubMatrix(value,0);  
    SubMatrix b = new SubMatrix(value,1);  
    SubMatrix c = a.mult(b); /* In-memory */  
    context.write(new BlockID(key.get()), new BytesWritable(c.getBytes()));  
}
```

```
INPUT: key, /* key by map task */  
value, /* value by map task */  
context /* IO interface (HBase) */  
void reduce(BlockID key, Iterable<BytesWritable> values, Context context)  
{  
    SubMatrix s = null;  
    for (BytesWritable value : values) {  
        SubMatrix b = new SubMatrix(value);  
        if (s == null) { s = b; }  
        else { s = s.add(b); }  
    }  
    context.write(...);  
}
```

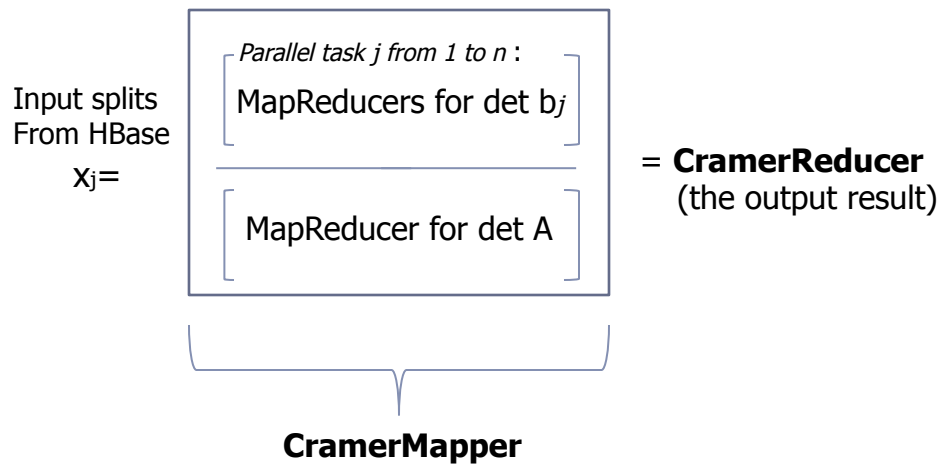
Case Study - Finding linear solution

- ▶ Finding linear solution
 - ▶ Cramer's rule
 - ▶ Conjugate Gradient Method

Case Study - Finding linear solution

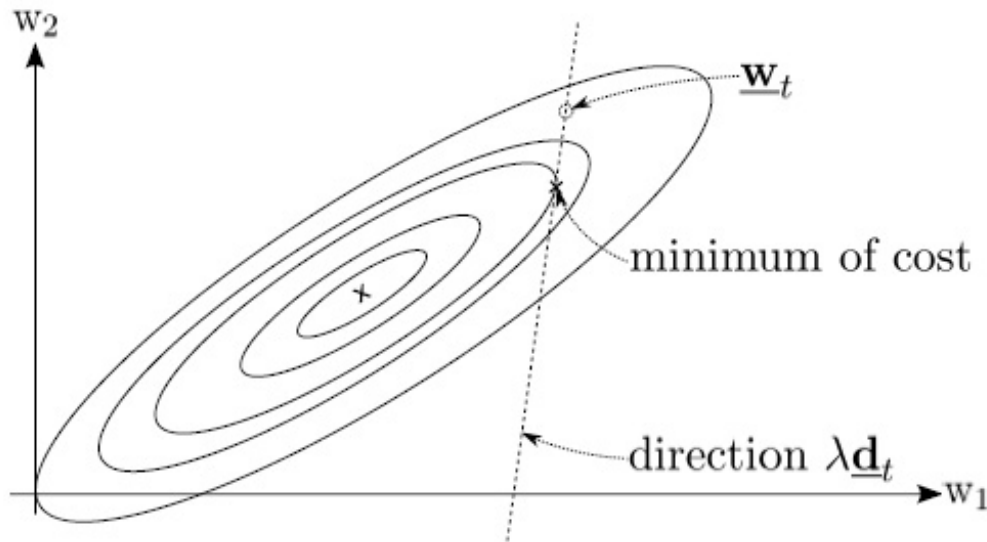
▶ Cramer's rule

$$x_j = \frac{\det B^j}{\det A}, \text{ where } B^j = \begin{bmatrix} a_{11} & a_{12} & \mathbf{b}_1 & a_{1n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \mathbf{b}_n & a_{nn} \end{bmatrix} \text{ has } \mathbf{b} \text{ in column } j.$$



Case Study - Finding linear solution

- ▶ Conjugate Gradient Method
 - ▶ Find a direction (conjugate direction)
 - ▶ Find a step size (Line search)



Case Study - Finding linear solution

▶ Conjugate Gradient Method (Algorithm)

```
/* Using nested-map interface */
void map(ImmutableBytesWritable key, Result value, Context context)
{
    /* For line search */
    g = g.add(-1.0, mult(x).getRow(0));
    alpha_new = g.transpose().mult(d) / d.transpose().mult(q);
    /* Find the conjugate direction */
    d = g.mult(-1).add(d.mult(alpha));
    q = A.mult(d);
    alpha = g.transpose().mult(d) / d.transpose().mult(q);
    /* Update x with gradient(alpha) */
    x = x.add(d.mult(alpha));
    /* Termination check method, such that
    length of direction is sufficiently
    small or x is converged into fixed value */
    if (checkTermination(d, x.getRow(0)))
    {
        /* Pass the solution (x) to reducer */
        context.write(new BlockID(key.get()), new BytesWritable(x.getBytes()));
    }
    context.write(new BlockID(key.get()), null);
}
```

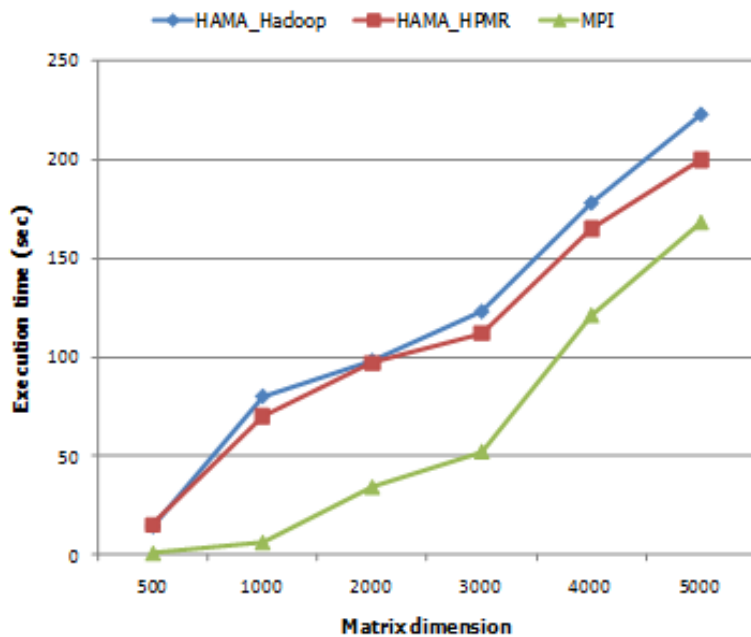
Evaluations

- ▶ TUSCI (TU Berlin SCI) Cluster
 - ▶ 16 nodes, two Intel P4 Xeon processors, 1GB memory
 - ▶ Connected with SCI (Scalable Coherent Interface) network interface in a 2D torus topology
 - ▶ Running in OpenCCS (similar environment of HOD)
- ▶ Test sets

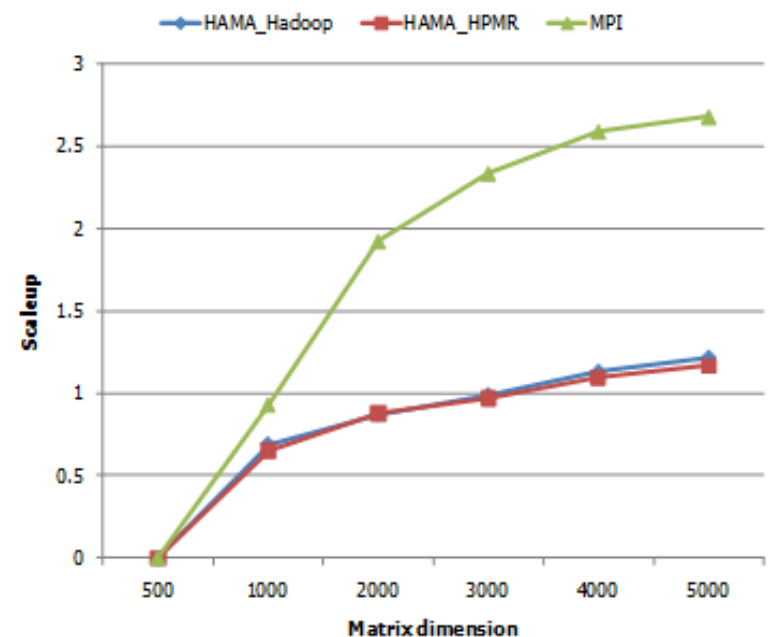
Workload	HAMA		MPI
Matrix Multiplication	Hadoop	HPMR	CXML (HP)
Conjugate Gradient (CG)	Hadoop	HPMR	CXML (HP)

Evaluations

- ▶ The comparison of average execution time and scaleup with Matrix Multiplication



(a) Execution time of Matrix multiplication

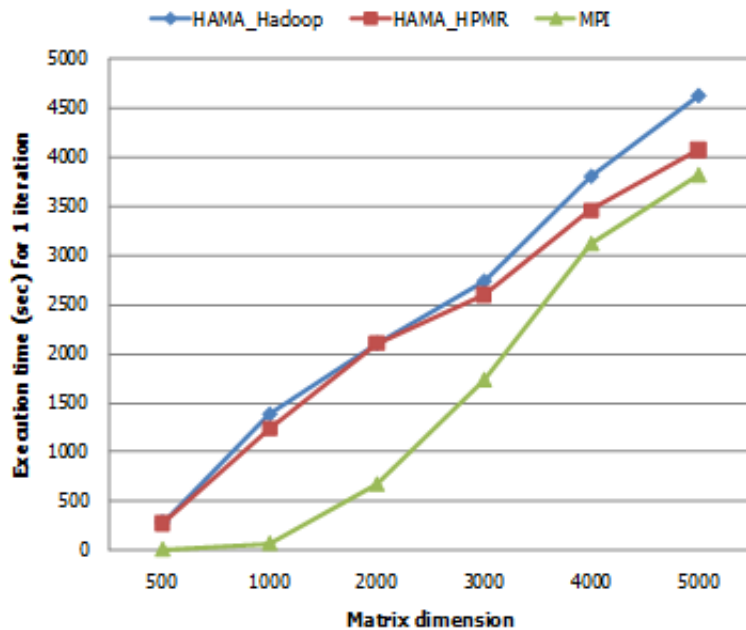


(b) Scaleup of Matrix multiplication

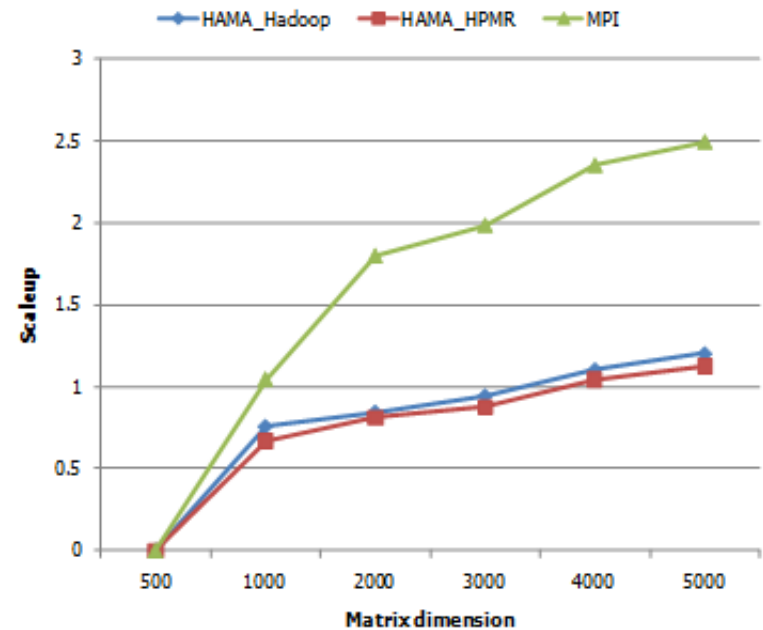
$$\text{scaleup} = \log(T(\text{dimension}) / T(500))$$

Evaluations

- ▶ The comparison of average execution time and scaleup with CG



(c) Execution time of CG method

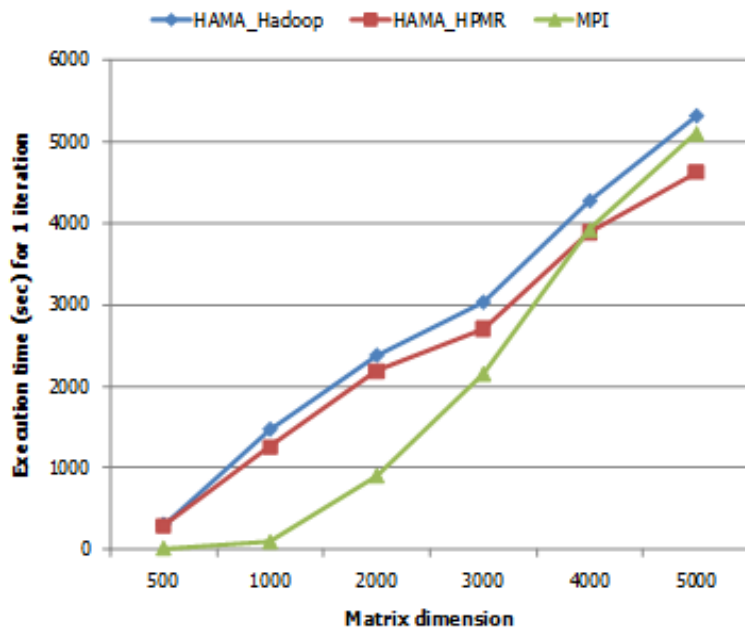


(d) Scaleup of CG method

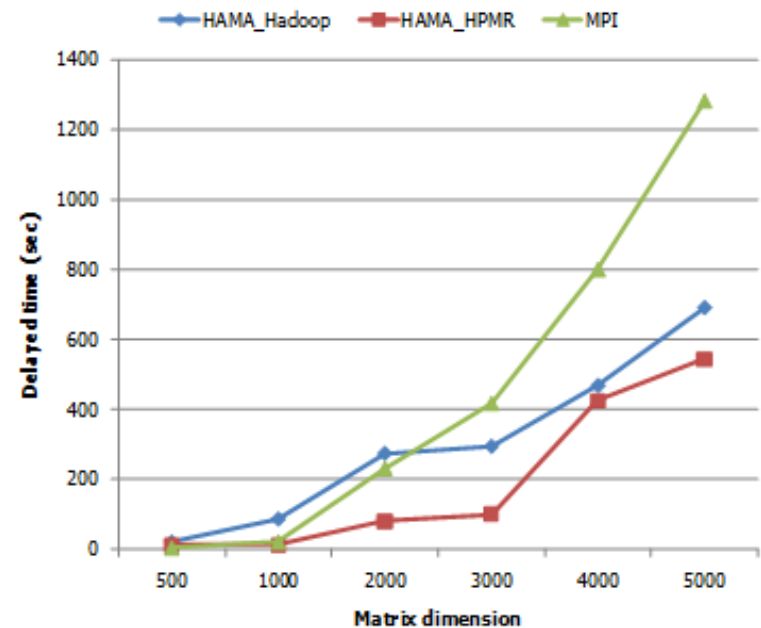
$$\text{Scaleup} = \log(T(\text{dimension}) / T(500))$$

Evaluations

- ▶ The comparison of average execution time with CG, when a single node is overloaded



(a) Average execution time



(b) Average delayed time

Conclusion

- ▶ HAMA provides the easy-of-use tool for data-intensive computations
 - ▶ Matrix computation with MapReduce
 - ▶ Graph computation with BSP
- ▶ We are going to provide the HAMA package as a *SaaS* in a cloud platform

Q & A



Thanks!

A hand-drawn illustration of a smiling face with arms raised, positioned below the word 'Thanks!'. The drawing is simple and stylized, with a circular head, a wide smile, and two arms raised in a gesture of joy or gratitude. A small '©' symbol is visible at the bottom right of the drawing.