INTRODUCING SCALEGRAPH : AN X10 LIBRARY FOR BILLION SCALE GRAPH ANALYTICS

Miyuru Dayarathna, Charuwat Houngkaew, and Toyotaro Suzumura

Department of Computer Science
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
Japan

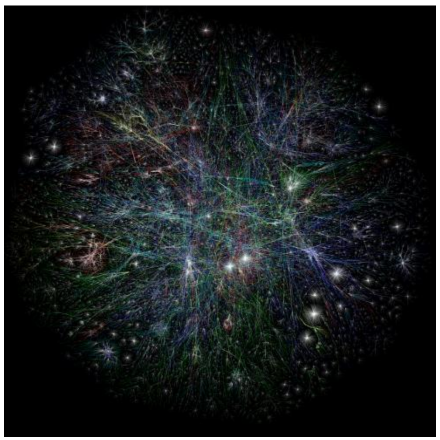
X10 Workshop 2012

6/14/2012

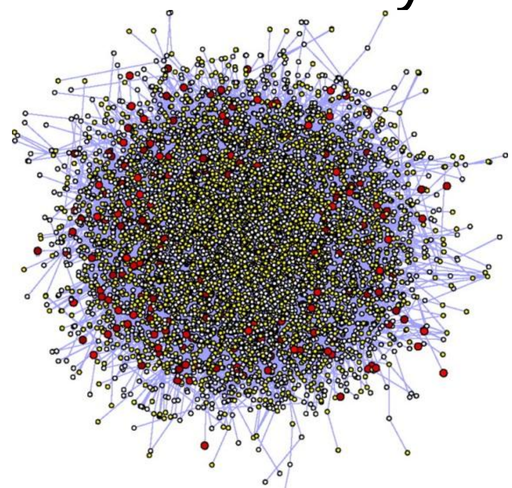
This research was partly supported by the Japan Science and Technology Agency (JST) Core Research of Evolutionary Science and Technology (CREST)

Background

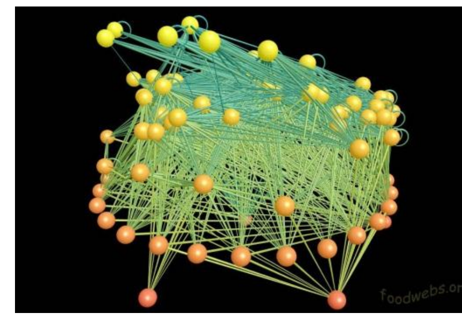
- Massive graph mining and Management has become an important research issue in recent years.



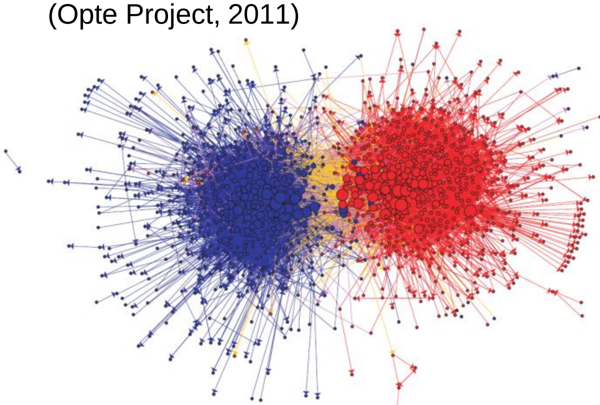
The network structure of the Internet
(Opte Project, 2011)



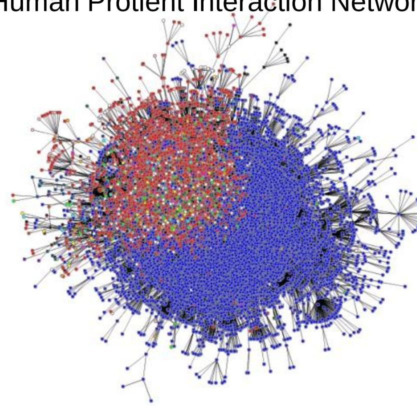
Human Protient Interaction Network (P.M. Kim et al, 2007).



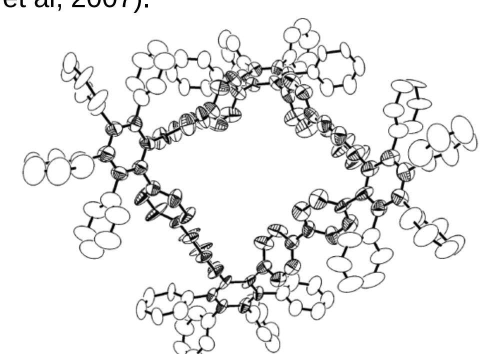
Food web of Carrbibean Reef3
(R.J. Williams et al., 2004)



Political Blogs (Adamic et al., 2005)



Blog Sphere core (M., Hurst, 2005)



Molecular Graph structure of Compound 7
(Song et al., 2005)

Background

- HPC programmer productivity is considered one of the important goals in achieving the Exascale computational capabilities.
- PGAS languages are an example for such initiatives.
- It is important for having a complex network analysis software APIs in such languages
- However there are no such libraries currently available

Current Libraries for Complex Network Analysis

- Do not aim at solving large graph problems (Beyond the scale of Billions of Vertices and Edges)
- Do not provide a complete mix of graph algorithms
- Famous example libraries,
  1. **Igraph** – by Gabor Csardi et al.
  2. **JUNG** (Java Universal Network/Graph Framework) – by Joshua O’Madadhain et al.
  3. **GraphStream** - Stefan Balev et al.
  4. **The Boost Graph Library (BGL)** – by Jeremy Siek et al.
  5. **JGraphT** - Barak Naveh et al.
  6. **Ruby Graph Library (RGL)** – by Horst Duchene
  7. **LEMON** – Alpar Juttner et al.
  8. **NetworkX** – Hagberg et al.
  9. **NG4J** – Bizer et al.

Research Problem

Comprehensive support for HPC programmers to specify highly productive, distributed, scalable graph analysis tasks for billion scale graphs has not been achieved yet.

Possible Solutions

- Create high level language wrappers for existing low level graph analysis libraries (E.g., Knowledge Discovery Toolbox [45])

[45] Adam Lugowski, David Alber, Aydin Buluç, John Gilbert, Steve Reinhardt, Yun Teng, and Andrew Waranis. A flexible open-source toolbox for scalable complex graph analysis. In SIAM Conference on Data Mining (SDM), 2012.

Presentation Outline

- Introduction
- Research Problem
- Proposed Solution
- Related Work
- Background (X10)
- Library's Design
- Implementation
- Evaluation
- Conclusion

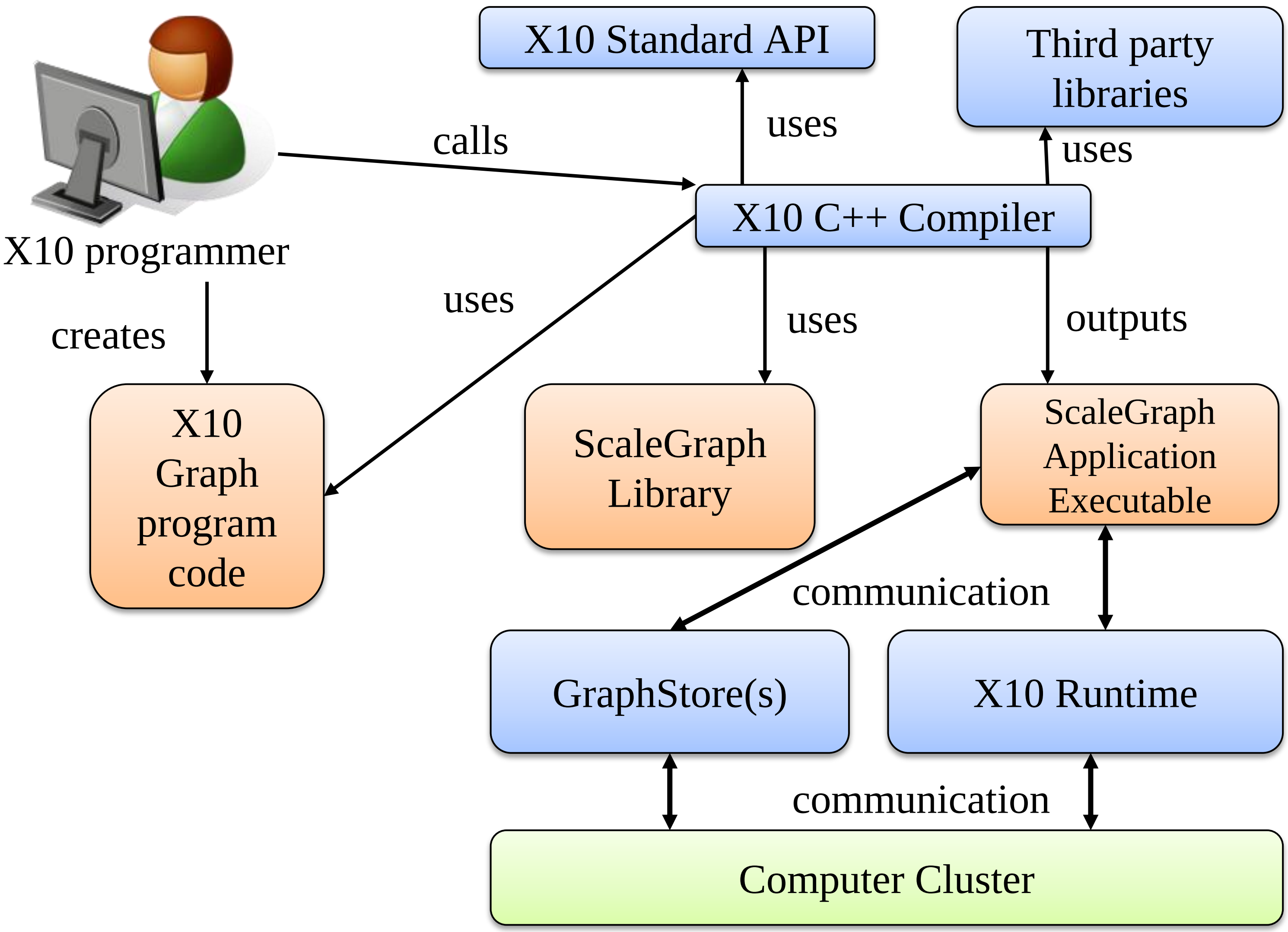
Aim and Objectives of ScaleGraph

- **Aim** - Create an X10 graph processing library which can efficiently process massive graphs (beyond the scale of billions of vertices and edges).
- **Objectives**
  - To define concrete abstractions for Massive Graph Processing
  - To investigate use of X10 (I.e., PGAS languages) for massive graph processing
  - To support significant amount of graph algorithms including algorithms (E.g., structural properties, clustering, community detection, etc.)
  - To create well defined interfaces to Graph Stores
  - To evaluate performance of each measurement algorithms and applicability of ScaleGraph using real/synthetic graphs in HPC environments.

Goal and Contributions of the Paper

- Establish the baseline architecture of ScaleGraph library
- **Contributions**
  1. We specify a graph API with graph representations, and algorithms for specifying graph processing in the scale of billions of vertices and edges
  2. We cover a wide range of graph representation standards which will enable complex network analysts to easily use their Massive (ranging from GB to TB) datasets.
  3. We make an initial scalability study of our API in Peta scale computer systems

ScaleGraph Architecture



Related Work (I)

- Complex Network Research - Igraph [15], SNAP [16]
  - Run only on workstations.
  - May scale only for few billion edges
- Graph Libraries - GGCL [17], BGL [18], JUNG [43]
- Generic Libraries – STAPL [2]
  - Our library is for distributed processing
  - Vertex and Edge Attributes (Colorful Graphs)

[15] G. Csardi and T. Nepusz. The igraph software package for complex network research. InterJournal, Complex Systems:1695, 2006. URL: http://igraph.sf.net.
[16] J. Leskovec. Snap: Stanford network analysis project. URL: http://snap.stanford.edu/, Jan. 2012.
[17] L.-Q. Lee, J. G. Siek, and A. Lumsdaine. The generic graph component library. SIGPLAN Not., 34:399–414, October 1999. ISSN 0362-1340.
[18] D. Batenkov. Boosting productivity with the boost graph library.XRDS, 17:31–32, Mar. 2011. ISSN 1528-4972.
[43] Sourceforge. Jung - java universal network/graph framework. URL:http://jung.sourceforge.net/index.html, Jan. 2012.
[2] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. Stapl: an adaptive, generic parallel c++ library. In Proceedings of the 14th international conference on Languages and compilers for parallel computing, LCPC'01, pages 193–208, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540- 04029-3.

Related Work (II)

- Distributed Graph Libraries – PBGL [21], ParGraph [24], ComBLAS [10]
  - Programmer productivity
- Shared Memory Graph Libraries – MTGL [7], SNAP (Georgia Tech) [46]
  - Need specialized hardware

[21] D. Gregor and A. Lumsdaine. Lifting sequential graph algorithms for distributed-memory parallel computation. SIGPLAN Not., 40:423–437, October 2005. ISSN 0362-1340.
[24] F. Hielscher and P. Gottschling. Pargraph. URL: http://pargraph.sourceforge.net/, Jan. 2012.
[10] A. Buluc¸ and J. R. Gilbert. The combinatorial blas: design, implementation, and applications. International Journal of High Performance Computing Applications, 25(4):496–509, 2011. http://hpc.sagepub.com/content/25/4/496
[7] J. Berry, B. Hendrickson, S. Kahan, and P. Konecny. Software and algorithms for graph queries on multithreaded architectures. In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pages 1 –14, march 2007.
[46] Kamesh Madduri: SNAP (Small-World Network Analysis and Partitioning) Framework. Encyclopedia of Parallel Computing 2011: 1832-1837

Related Work (III)

- Graph Analysis using X10 – Cong *et al.*[13][14]
  - We focus on Graph API
- Other Computational Models – Pregel [35]
  - We can implement programming models like Pregel in X10
- Importance of well defined abstractions – Kulkarni *et al.*[28]

[13] G. Cong, G. Almasi, and V. Saraswat. Fast pgas connected components algorithms. PGAS ’09, pages 13:1–13:6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-836-0.

[14] G. Cong, G. Almasi, and V. Saraswat. Fast pgas implementation of distributed graph algorithms. SC ’10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-7559-9.

[35] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 international conference on Management of data, SIGMOD ’10, pages 135–146, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0032-2.

[28] M. Kulkarni, K. Pingali, B. Walter, G. Ramanarayanan, K. Bala, and L. P. Chew. Optimistic parallelism requires abstractions. In Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation, PLDI ’07, pages 211–222, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-633-2.

X10 – An Overview

- X10 is a PGAS language being developed by IBM Research in collaboration with academic partners

X10 provides a programming model that can withstand architectural challenges posed by multiple cores, hardware accelerators, clusters, and super computers

↓

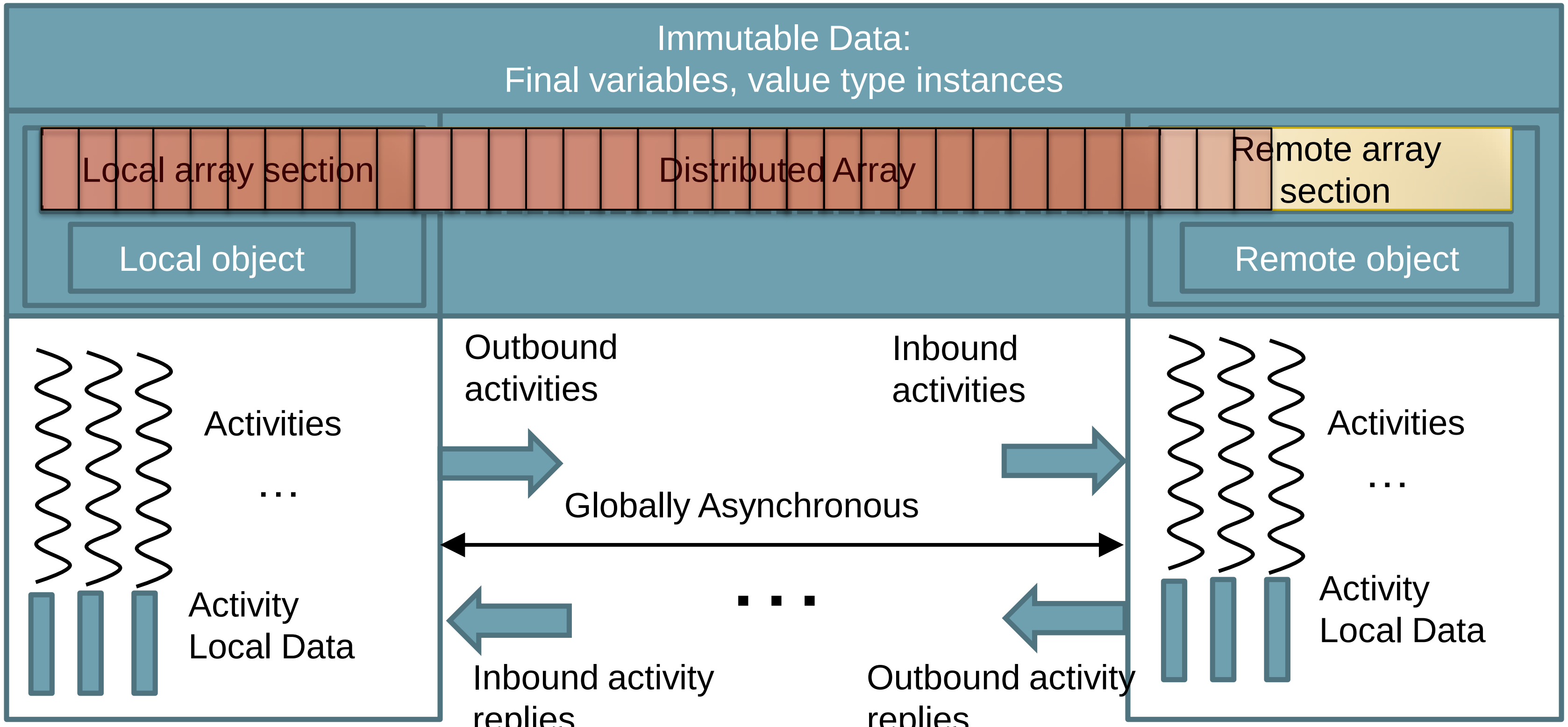
Increased programming productivity for future systems such as Exascale computing systems

X10 – An Overview

- X10 Language Features
  - Strongly typed
  - Object-oriented
  - Static type-checking
  - Static expression of program invariants
    - Supports the motivation of improving programmer productivity and performance
  - Latest Major Release X10 2.2 – source-to-source compilation
    - ScaleGraph uses native X10
  - Supports GPU
    - Currently ScaleGraph does not use GPU programming features

X10 – An Overview (Contd.)

- X10 Language Features
  - Place – A collection of non-migrating mutable data objects and the activities that operate on the data



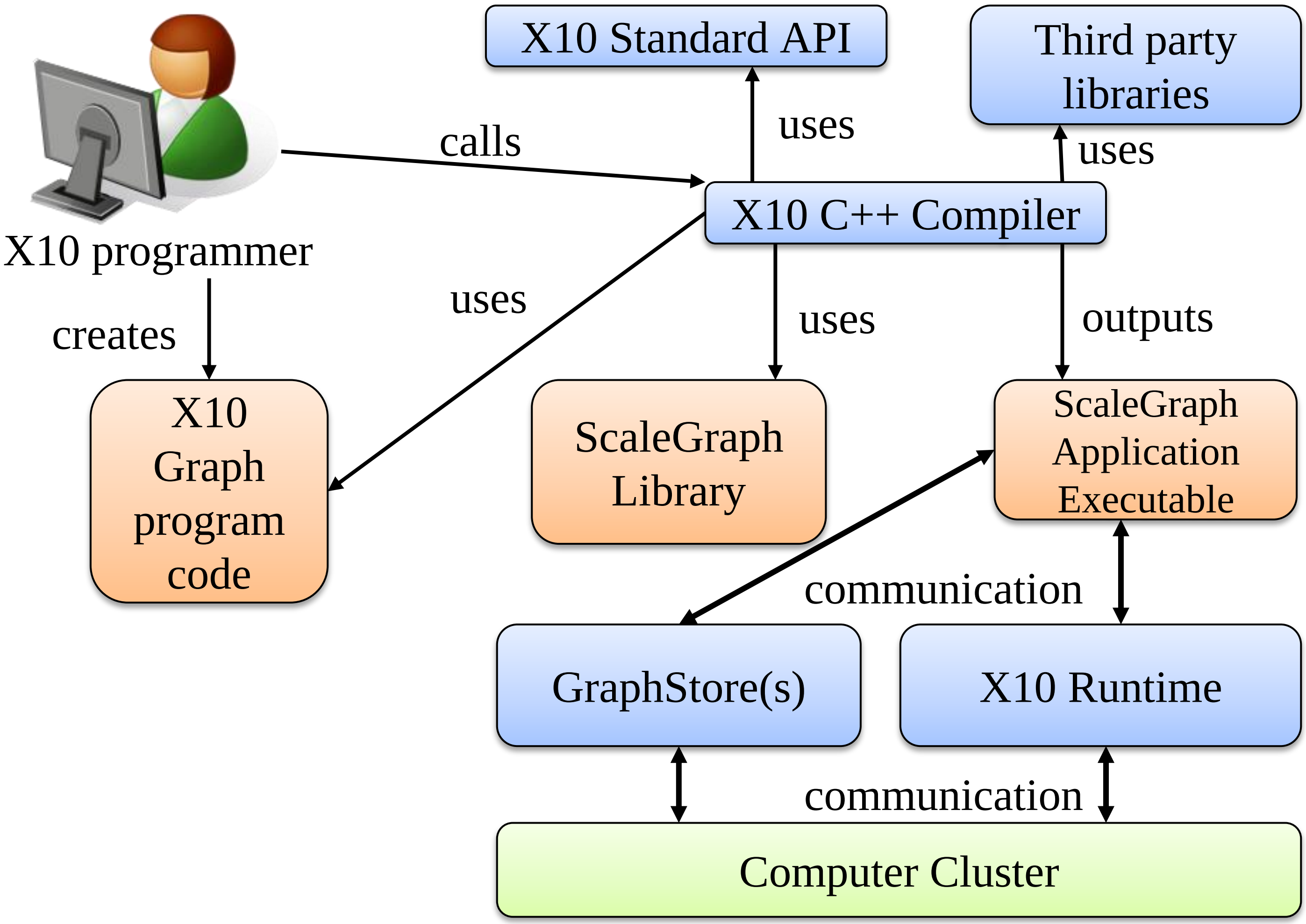
(P. Charles, et. al. 2005)

X10 – An Overview (Contd.)

- DistArray
  - Used for creating graph abstractions
- Annotation system of X10 allows extensions
  - We use @Native(lang, code) for implementing C++ language specific functions that are not implemented in current X10
    - Directory listing
    - GML Reader
- GlobalRef
  - Used as a support for coordinating activities between different places

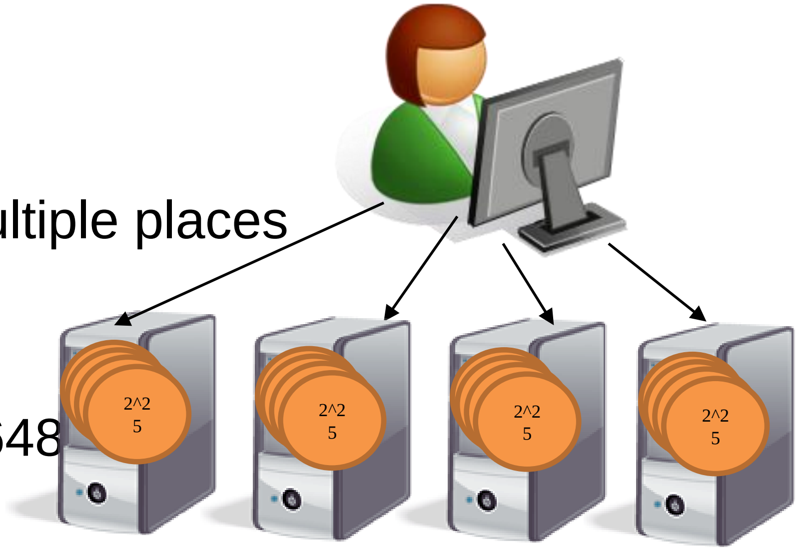
Library Design

- Aim : Define solid abstractions for billion scale graph processing



ScaleGraph Application types

- SMALL (n:n > 0, n ∈ ℕ)
  - Graph applications that run in a single place
  - To support complex network analysis community at large
    - Use the library in single node settings
  - Entire graph is stored in place 0.
  - Maximum 2^n vertices
  - E.g., n = 16, 2^{16} = 65,536 vertices
- MEDIUM (m:m > 0, m ∈ ℕ)
  - In memory graphs that is stored in multiple places
  - Maximum (2^m * *numberOfPlaces)* vertices
  - E.g. m = 24, (2^{24} * 128) = 2,147,483,648



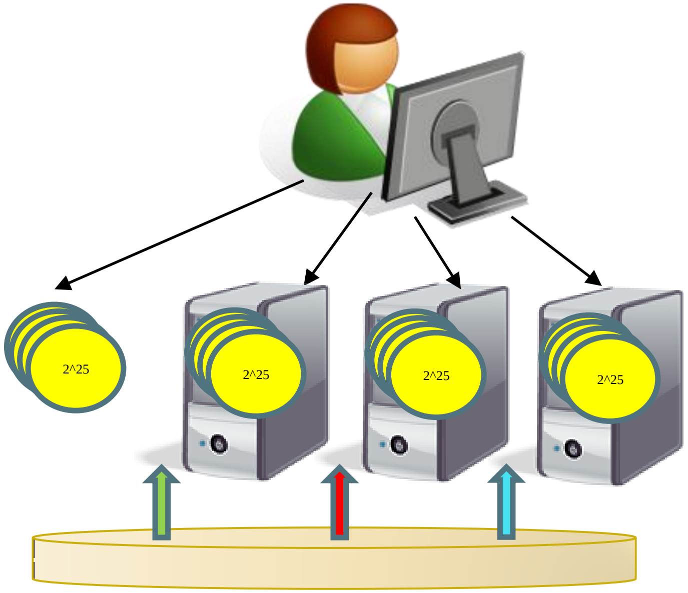
MEDIUM scale with four machines each machine holds 32 places (i.e., Total 128 places).

ScaleGraph Application types

- LARGE
  - End user does not have enough compute resources to instantiate sufficient amount of resources to hold billion scale graphs
    - Users with small compute clusters
    - Resourceful clusters such as super computers when the processed graphs need to reside on disks

Why three scales?

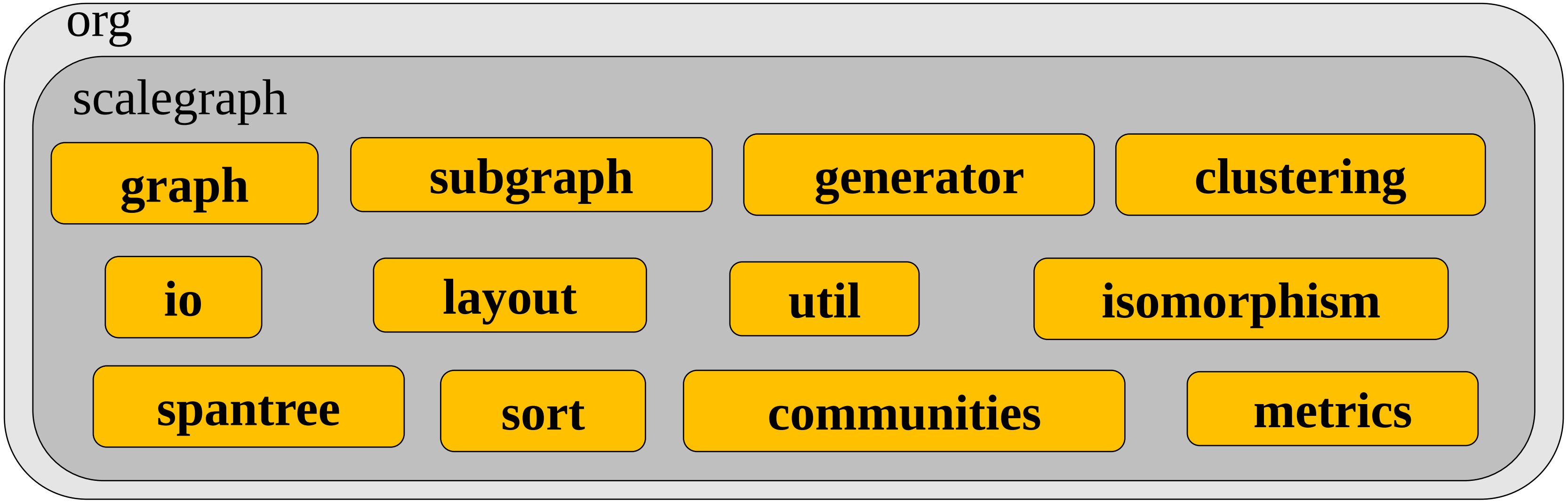
Performance tradeoffs and resource availability issues present in many graph analysis applications



LARGE scale with four machines each machine holds 32 places (i.e., Total 128 places). However only a portion of the graph is loaded on to the machines.

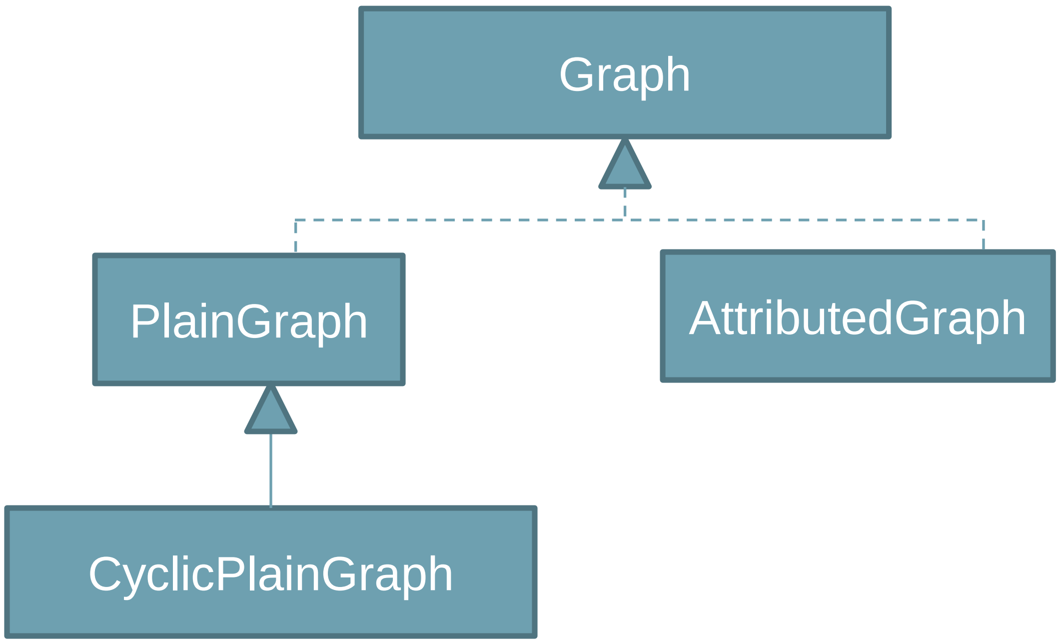
Software Design

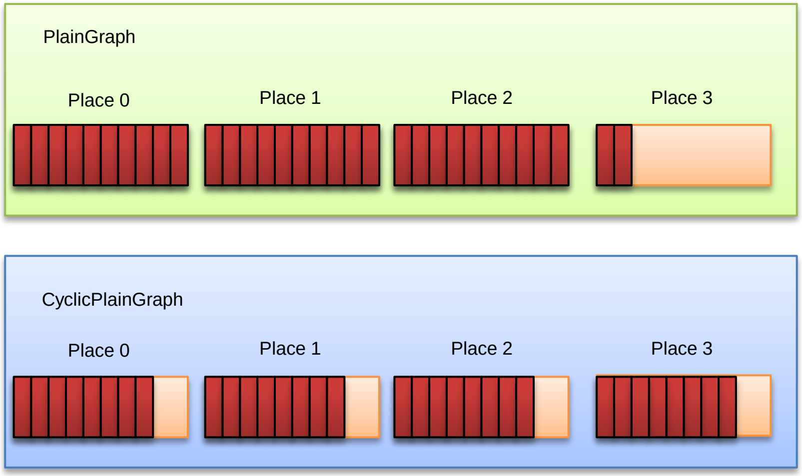
- Current Design consist if six main categories of classes : graph, I/O, generators, metrics, clustering, and communities



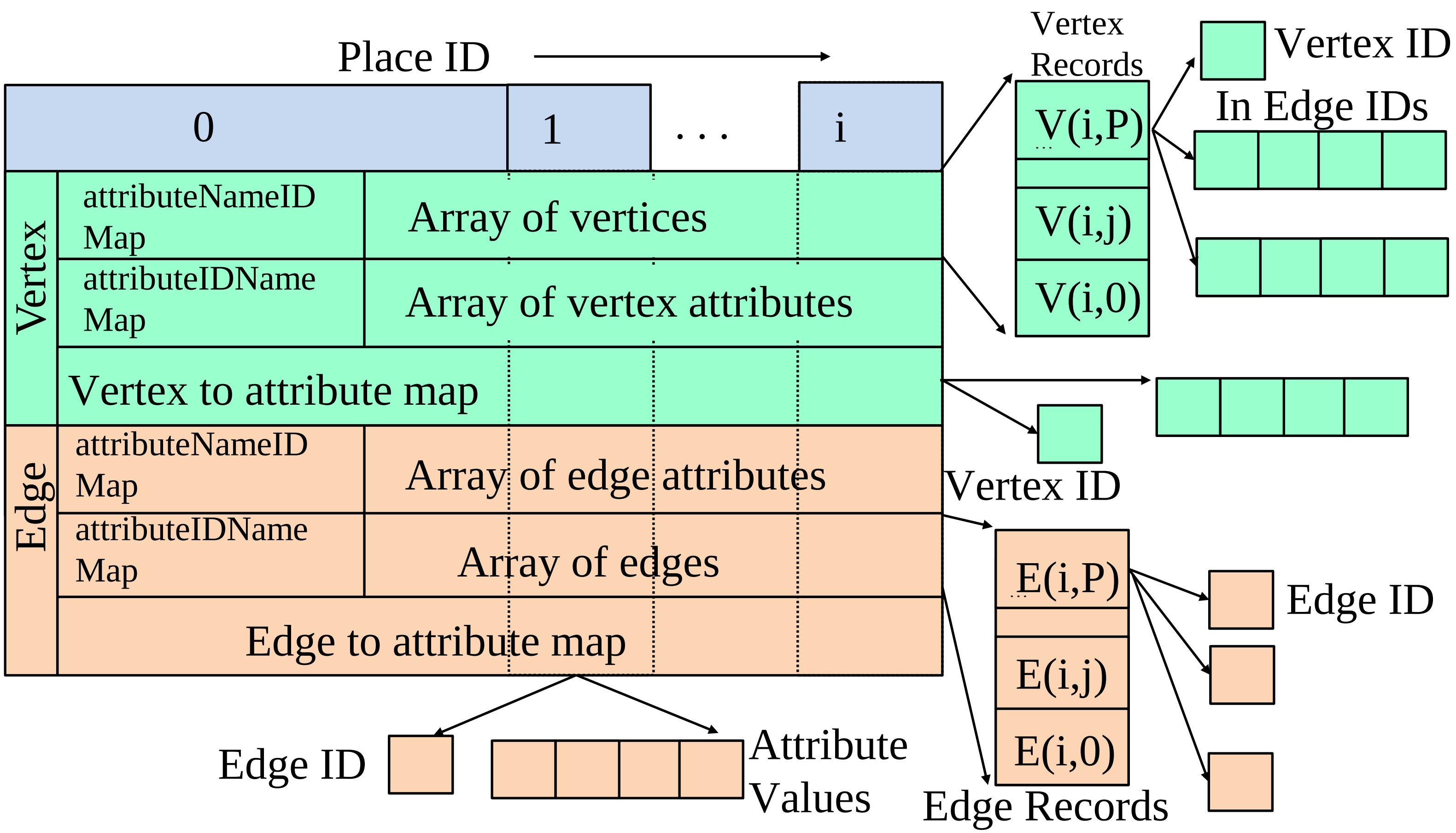
Software Design : Graph Representation

- Graph is just a data structure. Graph algorithms are coded separately.
- Graphs are represented as adjacency lists.
  - Most of the real world graphs are sparse

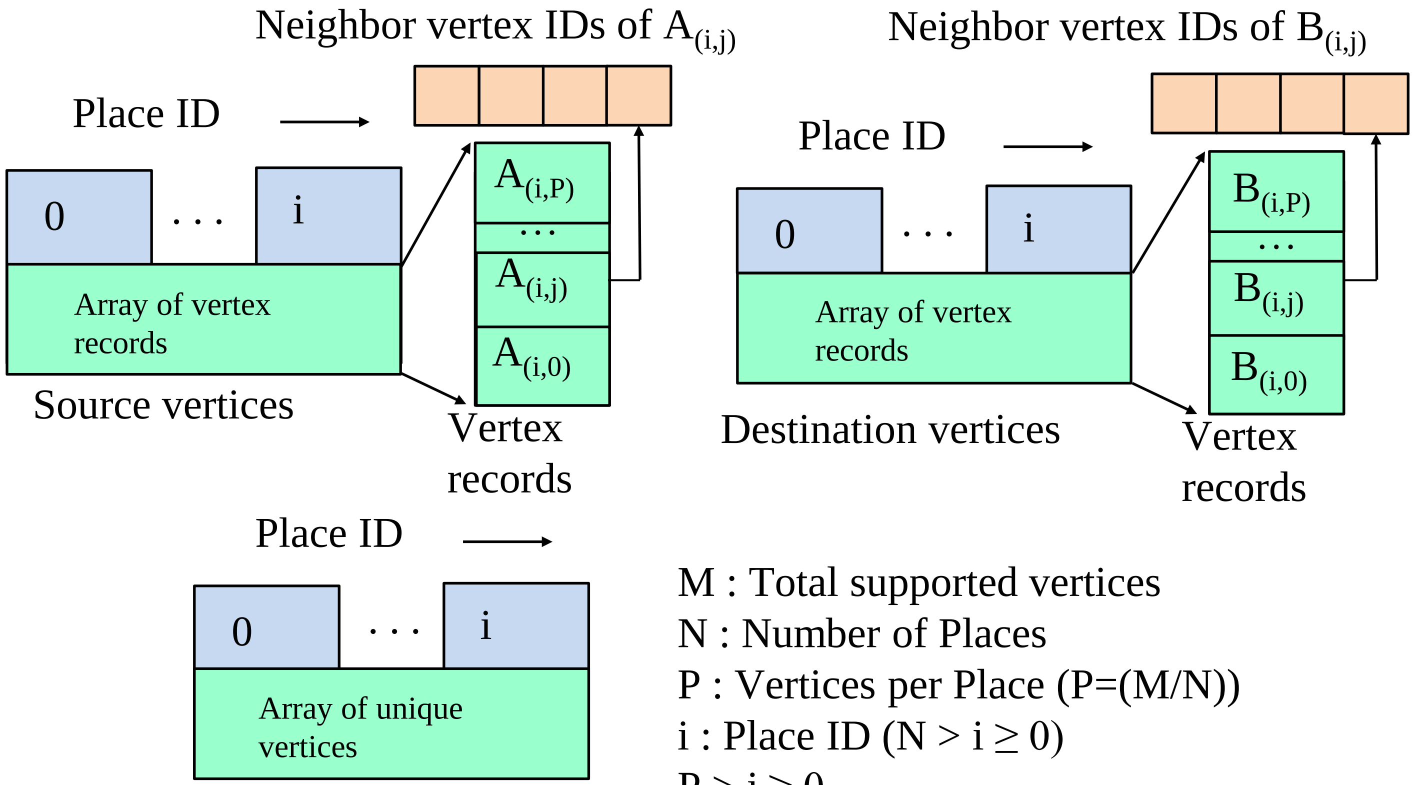




Software Design : Data Representation of AttributedGraph



Software Design : Data Representation of PlainGraph



M : Total supported vertices
N : Number of Places
P : Vertices per Place (P=(M/N))
i : Place ID ($N > i \geq 0$)
$P > j \geq 0$

Software Design : Graph Storage Formats

- There are variety of graph storage formats in use.

GEXF

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.1draft"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.gexf.net/1.1draft http://www.gexf.net/1.1draft/gexf.xsd"
   version="1.1">
   <graph mode="static" defaultedgetype="undirected">
      <nodes>
         <node id="4941" label="YBR236C"/>
         <node id="4942" label="YOR151C"/>
         <node id="4943" label="YML010W"/>
         <node id="4944" label="YNR016C"/>
         <!-- Rest of the Contents .... -->
         <edge id="20367" source="7276" target="7277"/>
         <edge id="20368" source="7278" target="7279"/>
         <edge id="20369" source="7293" target="7294"/>
      </edges>
   </graph>
</gexf>
```

GML

```
Creator "Mark Newman on Sat Jul 22 05:41:45 2006"
graph
[
  directed 0
  node
  [
    id 0
    label "8001"
  ]
  node
  [
    id 1
    label "64666"
  ]
  node
  [
    id 2
    label "7018"
  ]
```

GraphML

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
   <node id="n0"/>
   <node id="n1"/>
   <node id="n2"/>
   <node id="n3"/>
   <edge source="n0" target="n2"/>
   <edge source="n1" target="n2"/>
   <edge source="n2" target="n3"/>
  </graph>
</graphml>
```

Pajek

```
% US power grid - unweighted network
% from Panayiotis Tsaparas:
%   http://www.cs.helsinki.fi/u/tsaparas/MACN2006/data-code.html
% adapted for Pajek, V. Batagelj, March 19, 2006
%   0 -> 4941
*vertices 4941
*edgeslist
4941 386 395 451
1 3553 3586 3587 3637
2 3583
3 4930
4 88
5 13 120
6 8
7 8
8 6 7 9
9 8 10 61 75 205 208
```

Software Design : Graph Storage Readers/Writers

- A set of classes for reading and writing graph files located at org.scalegraph.io
- E.g.
  - EdgeListReader, EdgeListWriter
  - ScatteredEdgeListReader, ScatteredEdgeListWriter
  - GEXFReader, GEXFWriter
  - GMLReader, GMLWriter

Attributed Graphs	Non-attributed Graphs
GML	CSV
GEXF	DIMACS
GraphML	LGL
CSV	Pajek
GDF	
GraphViz	

Software Design : Graph Generators

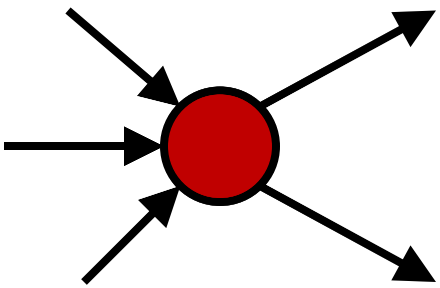
- Include a collection of synthetic graph generators
- Have implemented R-MAT generator
- Working on
  - BarabasiAlbertGenerator
  - CitationgraphGenerator
  - ErdosRenyiGenerator

Software Design : Graph Structural Properties

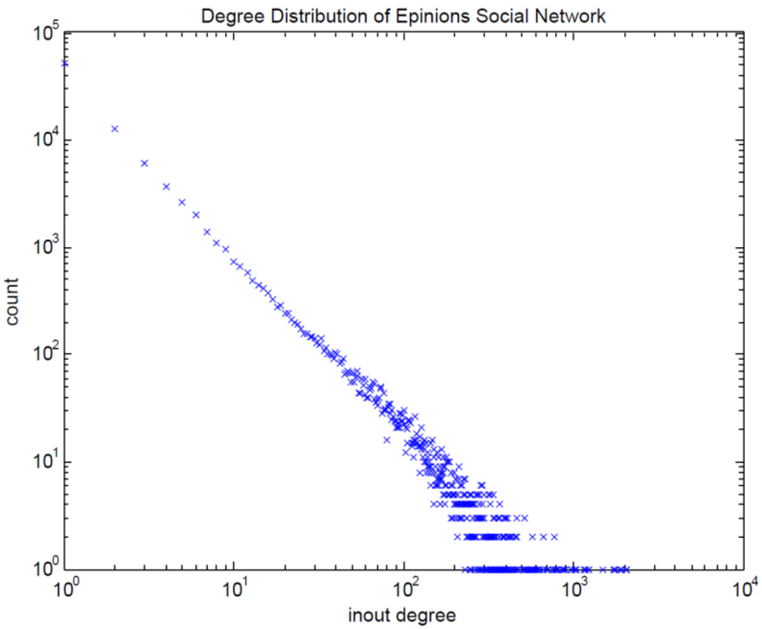
- Graphs contains specific topological features which characterize their connectivity.
- Implemented
  - Degree Distribution Calculation (in-degree, out-degree, in/out-degree)
  - Betweeness Centrality (BC)
  - PageRank/RWR
  - Clusters (E.g., Spectral Clustering)
- Planned other metrics
  - Diameter
  - Density
  - Complexity
  - Cliques
  - Kcores
  - Mincut
  - Connected Component

Implementation : Background – Degree Distribution Calculation, R-MAT Scale

- If one denotes degree by k, then the degree distribution can be represented by pk.



In degree = 3
Out degree = 2
In/Out degree = 5

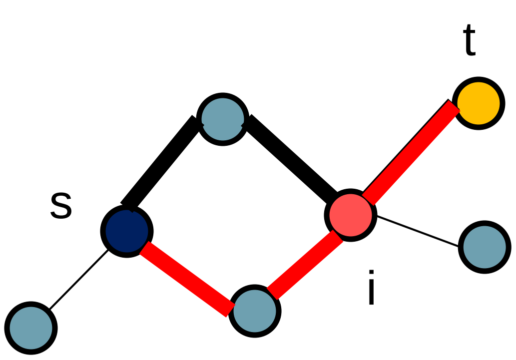


- R-MAT scale is an integer that specifies the number of vertices available in a graph. E.g. Scale 10 graph has 1024 vertices

Implementation : Background – Betweeness Centrality (BC)

- BC measures the extent to which a vertex lies on paths between other vertices
- If n^i_{st} be the number of geodesic paths from s to t that pass through i (s, t, and i are vertices of the graph, s≠t≠i)
- If total number of geodesic paths from s to t is denoted as g_{st}
- Betweenness Centrality Can be specified as follows,

$$x_i = \sum_{st} n^i_{st} / g_{st}$$



BC score of i = 2/2

Implementation : An example for use of AttributedGraph

Define attribute schema

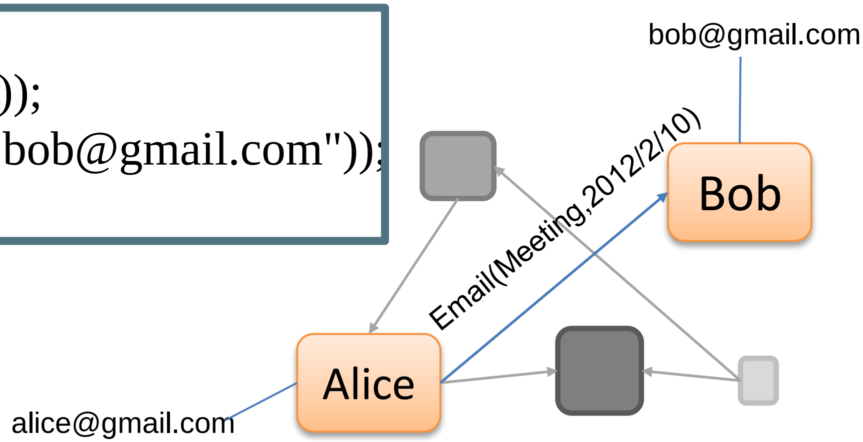
```
val attrArray:ArrayList[Attribute] = null;
schema: AttributeSchema = new AttributeSchema();
schema.add("fname",           AttributeSchema.StringAttribute);
schema.add("email_add",       AttributeSchema.StringAttribute);
schema.add("age", AttributeSchema.IntAttribute);
```

First vertex

```
attrArray = new ArrayList[Attribute]();
attrArray.add(new StringAttribute("fname", "Alice"));
attrArray.add(new StringAttribute("email_add",   "alice@gmail.com"));
v0:Vertex = new Vertex(attrArray);
```

```
attrArray = new ArrayList[Attribute]();
attrArray.add(new StringAttribute("fname", "Bob"));
attrArray.add(new StringAttribute("email_add",   "bob@gmail.com"));
v1:Vertex = new Vertex(attrArray);
```

Second vertex



Implementation : An example for use of AttributedGraph (Contd.)

```
g: AttributedGraph = AttributedGraph.make();
g.setVertexAttributeSchema(schema);
g.addVertex(v0);
g.addVertex(v1);
```

Initialize the graph and add the two vertices

```
schema: AttributeSchema = new AttributeSchema();
schema.add("title", AttributeSchema.DateAttribute);
schema.add("dtime", AttributeSchema.DateAttribute);
g.setEdgeAttributeSchema(schema);
```

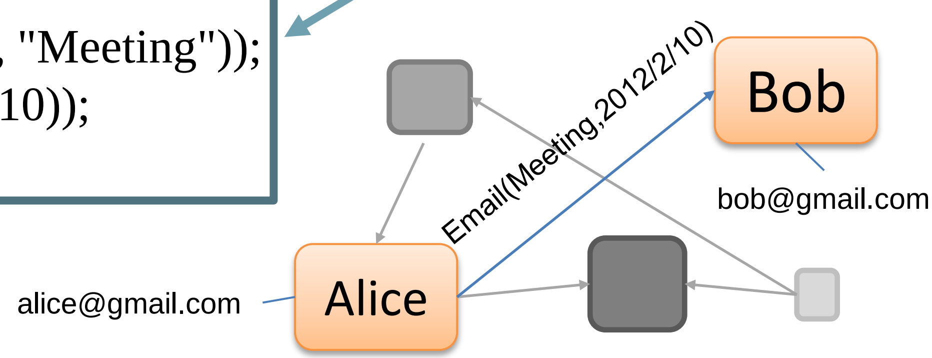
Create edge attribute schema

```
attrArray = new ArrayList[Attribute]();
attrArray.add(new StringAttribute("title", "Meeting"));
attrArray.add(new DateAttribute(2012,2,10));
e0: Edge = new Edge(v0,v1, attrArray);
```

Create the edge

```
g.addEdge(e0);
```

Add the edge



Implementation : Run Betweeness Centrality on AttributedGraph

```
var graph: AttributedGraph;

//Load the graph data from secondary storage
graph = GMLReader.loadFromFile("/data/power_grid.gml");

//Run the Betweeness Centrality calculation
val result = BetweennessCentrality.run(graph, false);
```

Implementation : Betweeness Centrality on PlainGraph

Initialize the data structures

```
finish {
  val distVertexList:DistArray[Long] = this.plainGraph.getVertexList();
  val localVertices : Array[Long]{self.rank == 1} =
distVertexList.getLocalPortion();
  val numLocalVertices: Int = localVertices.size;
  val numThreads = Runtime.NTHREADS;
  val chunkSize = numLocalVertices / numThreads;
  val remainder = numLocalVertices % numThreads;

  var startIndex: Int = 0;

  for(threadId in 0..(numThreads -1 )) {
      async doBfsOnPlainGraph(threadId, numThreads, localVertices);
  }
}
```

Distributed BFS

Implementation : Betweeness Centrality on PlainGraph (Contd.)

BC score normalization

```
// If undirected graph divide by 2
if(this.plainGraph.isDirected() == false) {
 if(this.isNormalize) {
    // Undirected and normalize
    betweennessScore.map(betweennessScore, (a: Double) => a /
     (((numVertex - 1) * (numVertex - 2))) );
 } else {
    // Undirected only
    betweennessScore.map(betweennessScore, (a: Double) => a / 2 );
 }
} else {
 if(this.isNormalize) {
 // Directed and normalize
 betweennessScore.map(betweennessScore, (a: Double) => a /
  ((numVertex -1) * (numVertex - 2)) );
 }
}
```

BC results synchronization

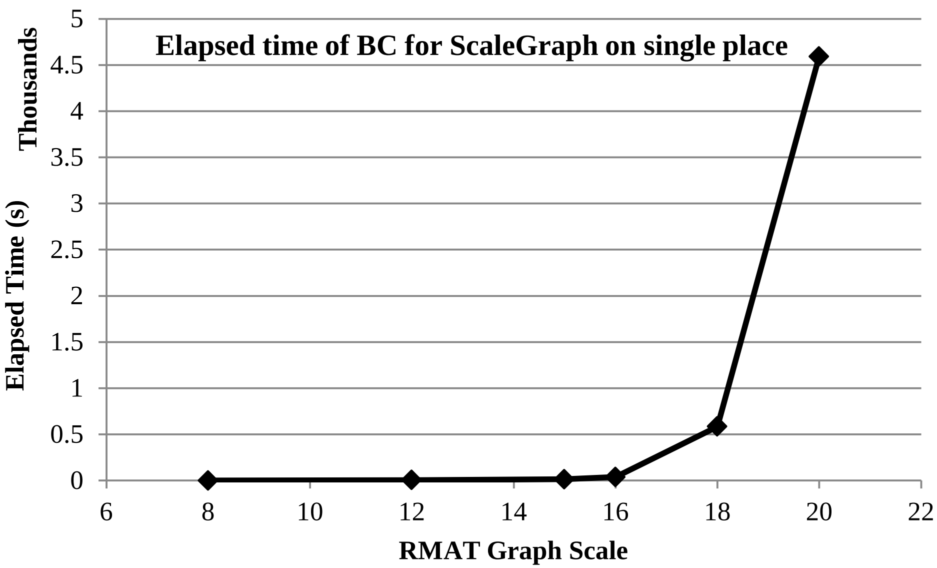
```
Team.WORLD.allreduce(here.id, betweennessScore, 0,
   betweennessScore, 0, betweennessScore.size, Team.ADD);
}
```

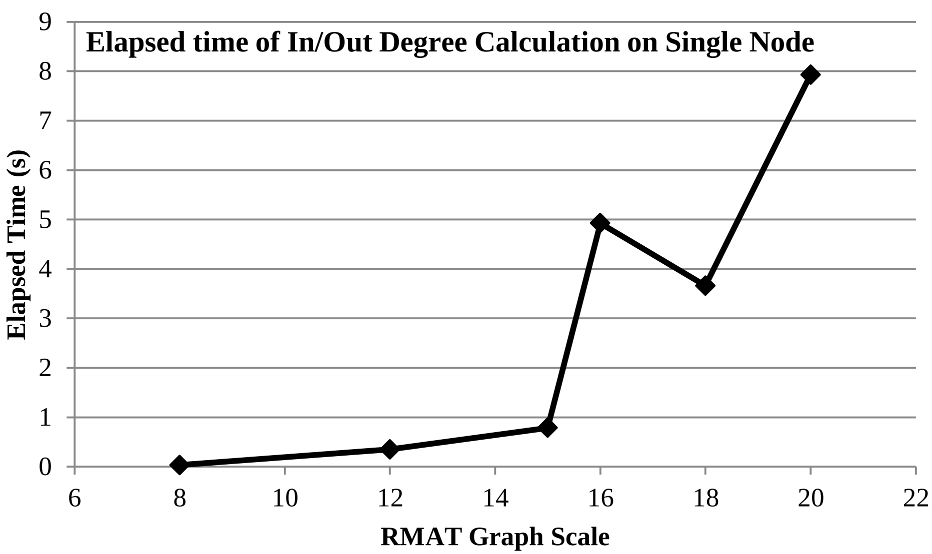
Evaluation : Environment

- Conducted on Tsubame 2.0 (5th ranked super computer on November 2011 top 500 list) on 4 nodes

CPU/Core count	Two Intel®Xeon®X5670 @ 2.93GHz CPUs each with 6 cores. Total 12 cores per node/24 hardware threads
RAM	54GB per node
Interconnect	Infiniband Network (Voltaire Grid Director 4700)
Secondary storage	GPFS/Luster file system
OS	SUSE Linux Enterprise Server 11 SP1
X10 version	X10.2.2.2
X10 Runtime	X10 native, MPI runtime. Used MPICH 2.1.4. X10 was built with following options: -DNO_CHECKS=true –Doptimize=true squeakyclean
X10 environment varaibles	X10_STATIC_THREADS=true X10_NTHREADS=22

Evaluation : Elapsed time on single place



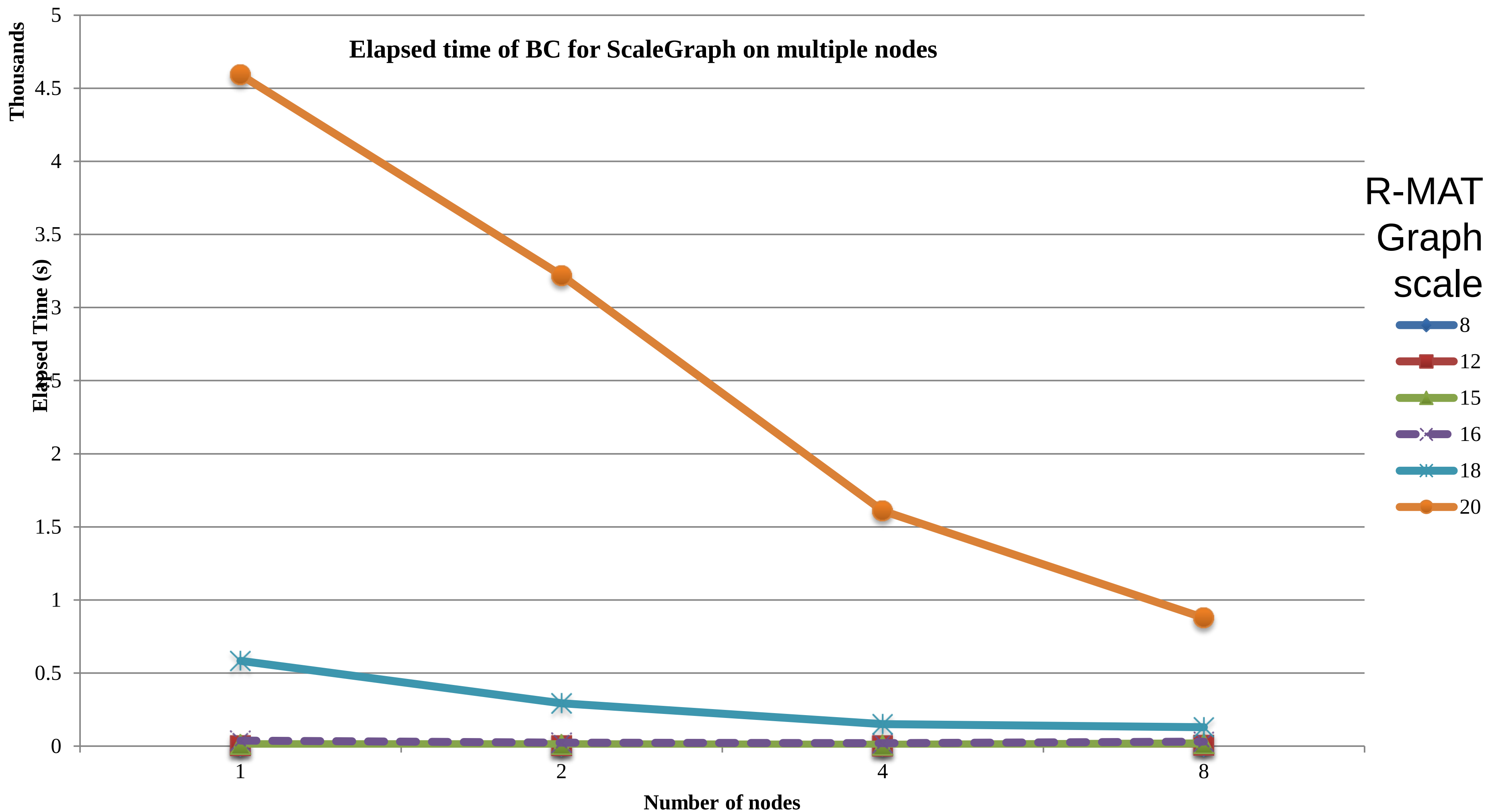


Betweenness Centrality

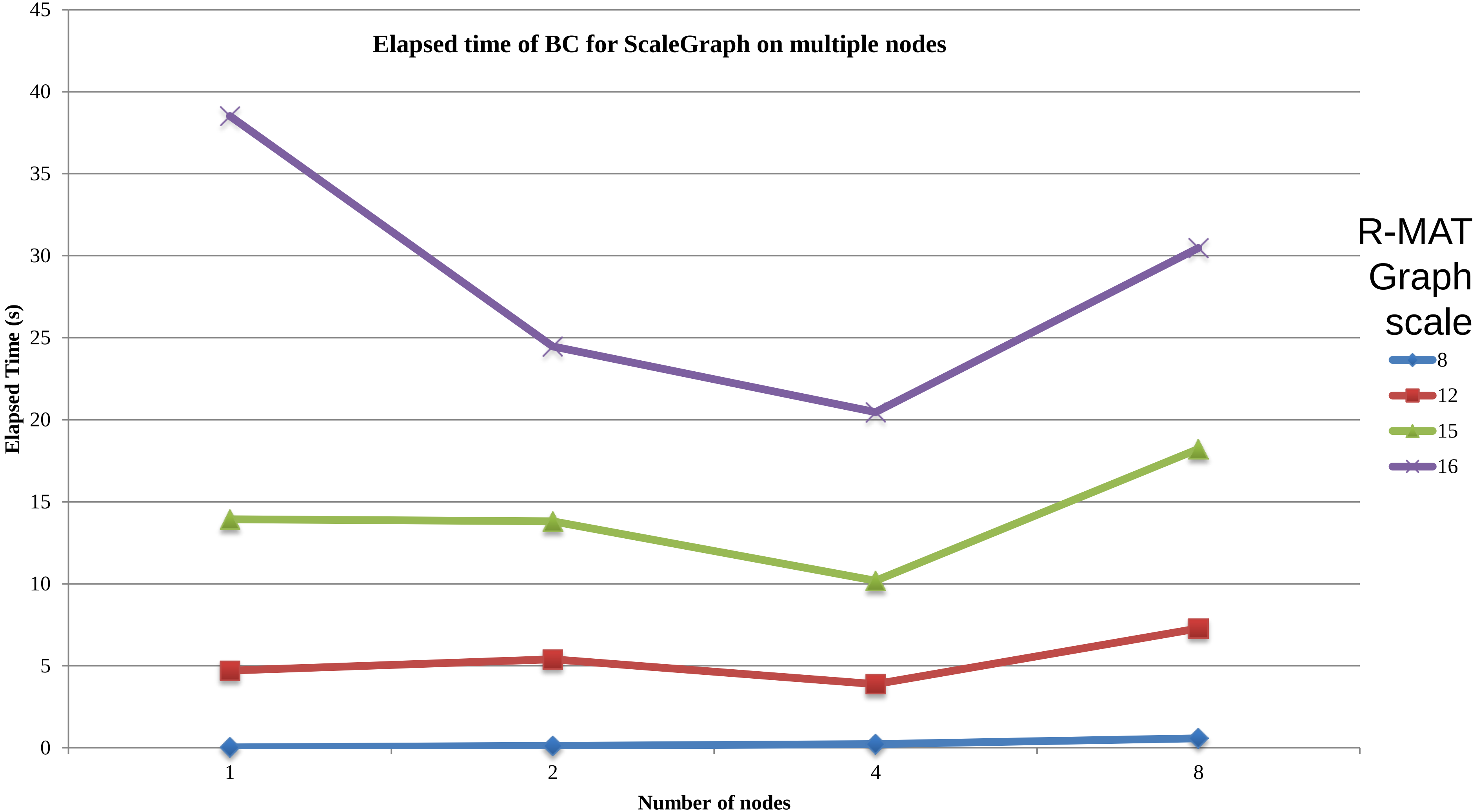
In/Out Degree Distribution

Scale 16 has a knee because it has more edges compared to scale 18

Evaluation: Elapsed time of BC of ScaleGraph on multiple nodes



Evaluation: Elapsed time of BC of ScaleGraph on multiple nodes

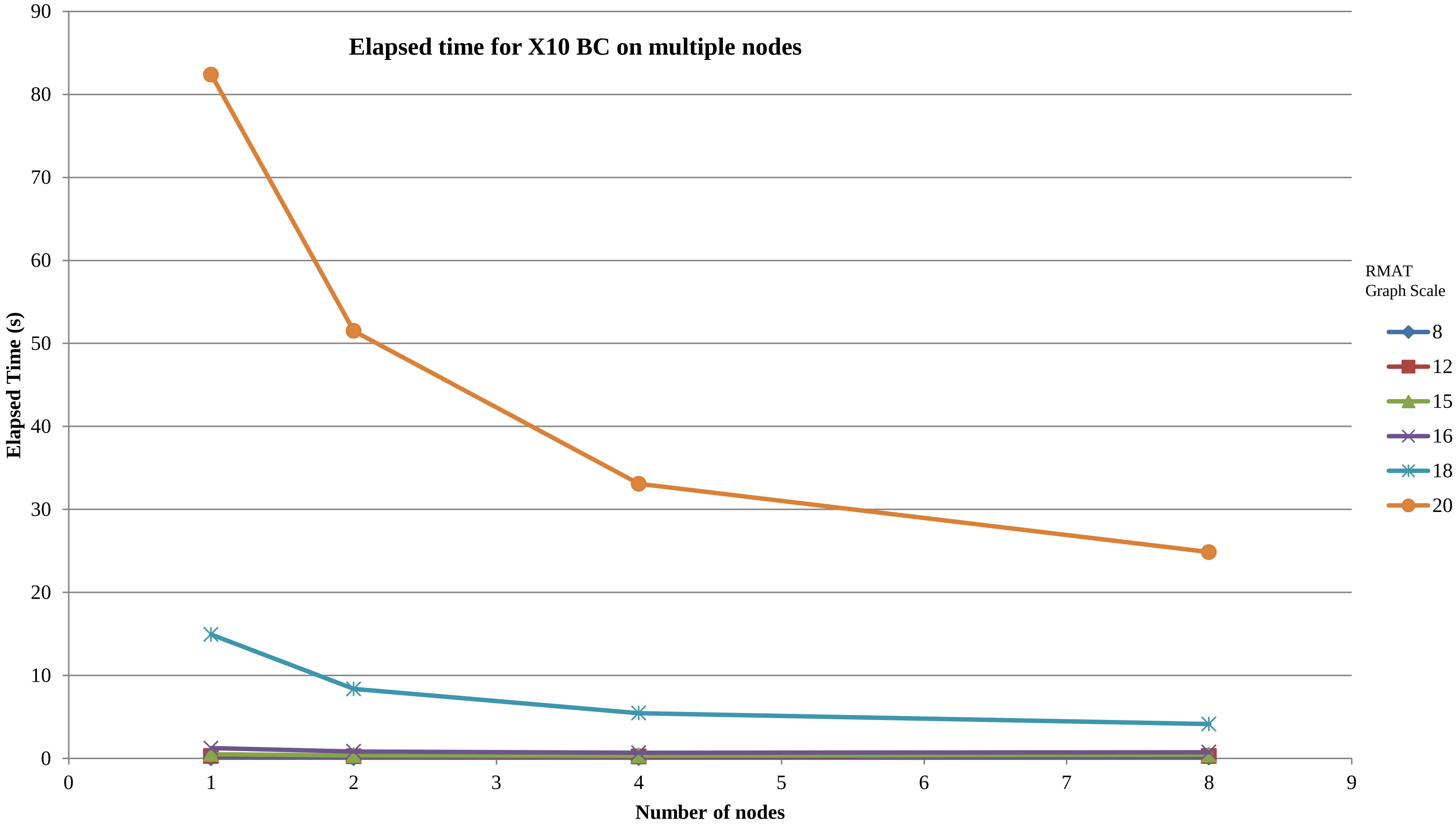


Evaluation: What is X10 BC?

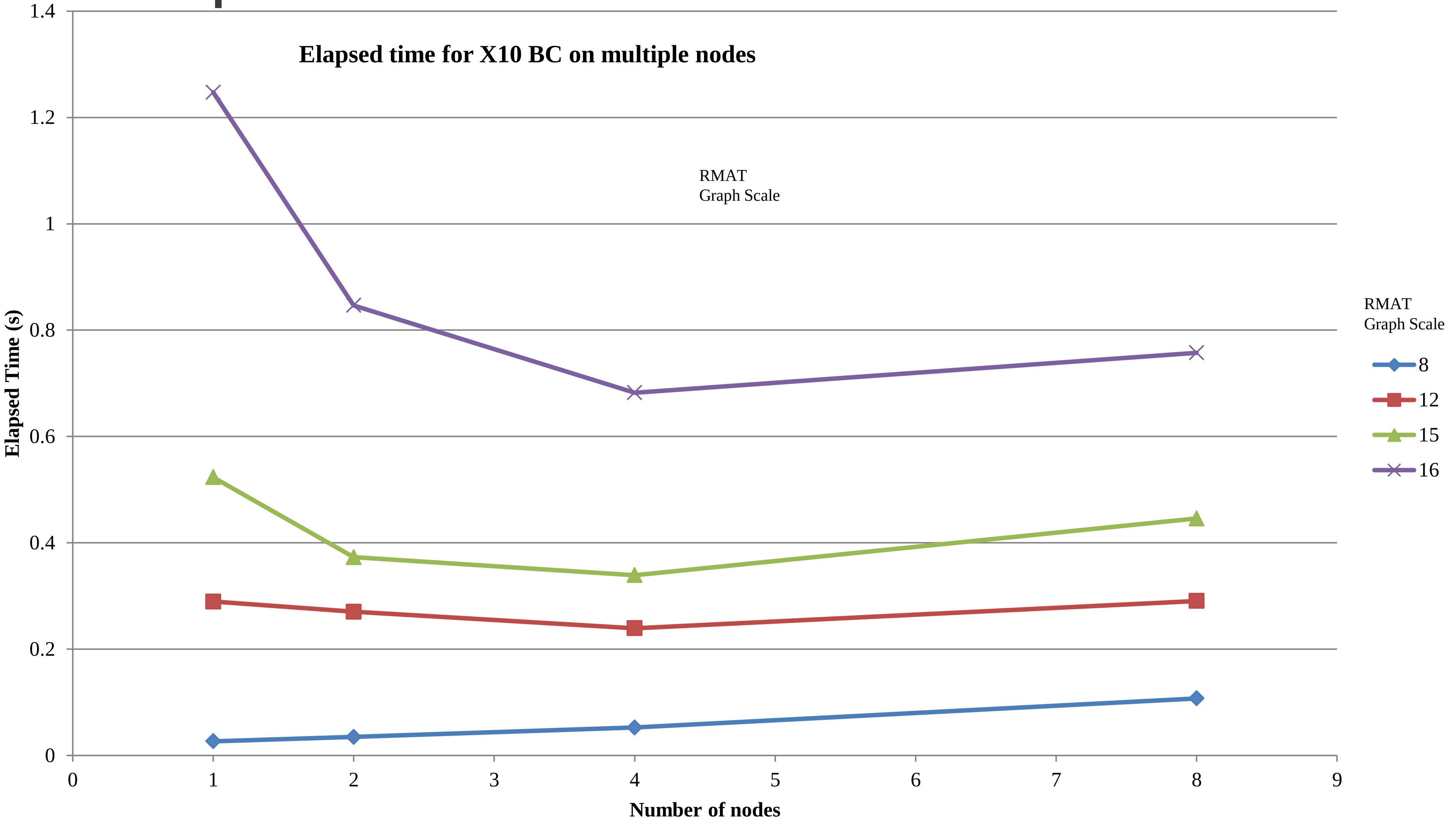
- A benchmark implementation of Betweenness Centrality
- Available from X10 source distribution from

  http://x10.svn.sourceforge.net/viewvc/x10/benchmarks/trunk/BC/

Evaluation: Elapsed time of X10 BC on multiple nodes



Evaluation: Elapsed time of X10 BC on multiple nodes



Evaluation : Degree Distribution calculation on KAIST Twitter dataset

- Contains 41.7million user profiles represented as follower/followee relationship
- Contains 1.47 billion edges
- The dataset of 11GB (on GPFS) was scattered into 5454 files each of 2MB in size
- Results (Three times average)
  - Data loading : 40 minutes
  - Get vertex count : 81 seconds
  - Get edge count : 93 seconds
  - In/out degree calculation: 1hour and 12 minutes

Conclusion

- Objective of this paper : Introduce the design and some initial experiment results of ScaleGraph
- Concrete abstractions for representing graph data on distributed environments while providing simple API for X10 application developer community
- Distinguishing feature : Graph is distributed across places
  - Difficult to load.
  - Solved by graph scattering

Current status and Future Work

- Five Developers (2 part-time)
- 14,000 lines of X10 code
- Currently working on
  - Improving scalability of Algorithms. Experiments are done on Tsubame 2.0
    - Degree, BC, Spectral Clustering, PageRank, Random Walk With Restart)
  - Improving scalability of Data representation
    - CyclicPlainGraph
  - Implement other graph algorithms
    - Graph pattern matching, graph property calculation algorithms
  - Getting ready for Release 1.0 soon. Also planning for release 2.0.
- In Future
  - Support for other complex graph algorithms and analysis techniques
  - Usage of heterogenious hardware

Acknowledgement

- This research was partly supported by the Japan Science and Technology Agency (JST) Core Research of Evolutionary Science and Technology (CREST)