Trinity (GraphEngine)

FAMIM TALUKDER

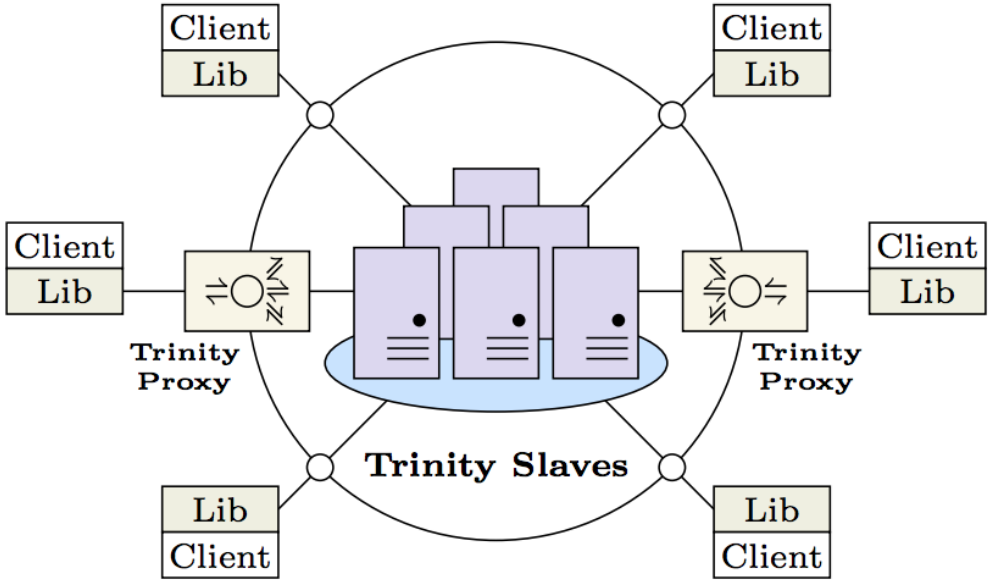
Background

- Developed by Microsoft Research Asia – in 2013
- Renamed to GraphEngine (GE)
- Satisfy two requirements
  - Online query processing – low latency
    - Ex. Link prediction in social networks
  - Offline graph analytics – high throughput
    - Ex. PageRank in the WWW
- Belief: High speed network is more available + DRAM prices going down → in-memory solutions will be cheap!

	Graph Database	Query Processing	Graph Analytics	Scale-out System
Neo4j [4]	Yes	Yes	Yes	No
HyperGraphDB [22]	Yes	Yes	No	No
GraphChi [25]	No	No	Yes	No
PEGASUS [23]	No	No	Yes	Yes
MapReduce [15]	No	No	Yes	Yes
Pregel [28]	No	No	Yes	Yes
GraphLab [1]	No	No	Yes	Yes

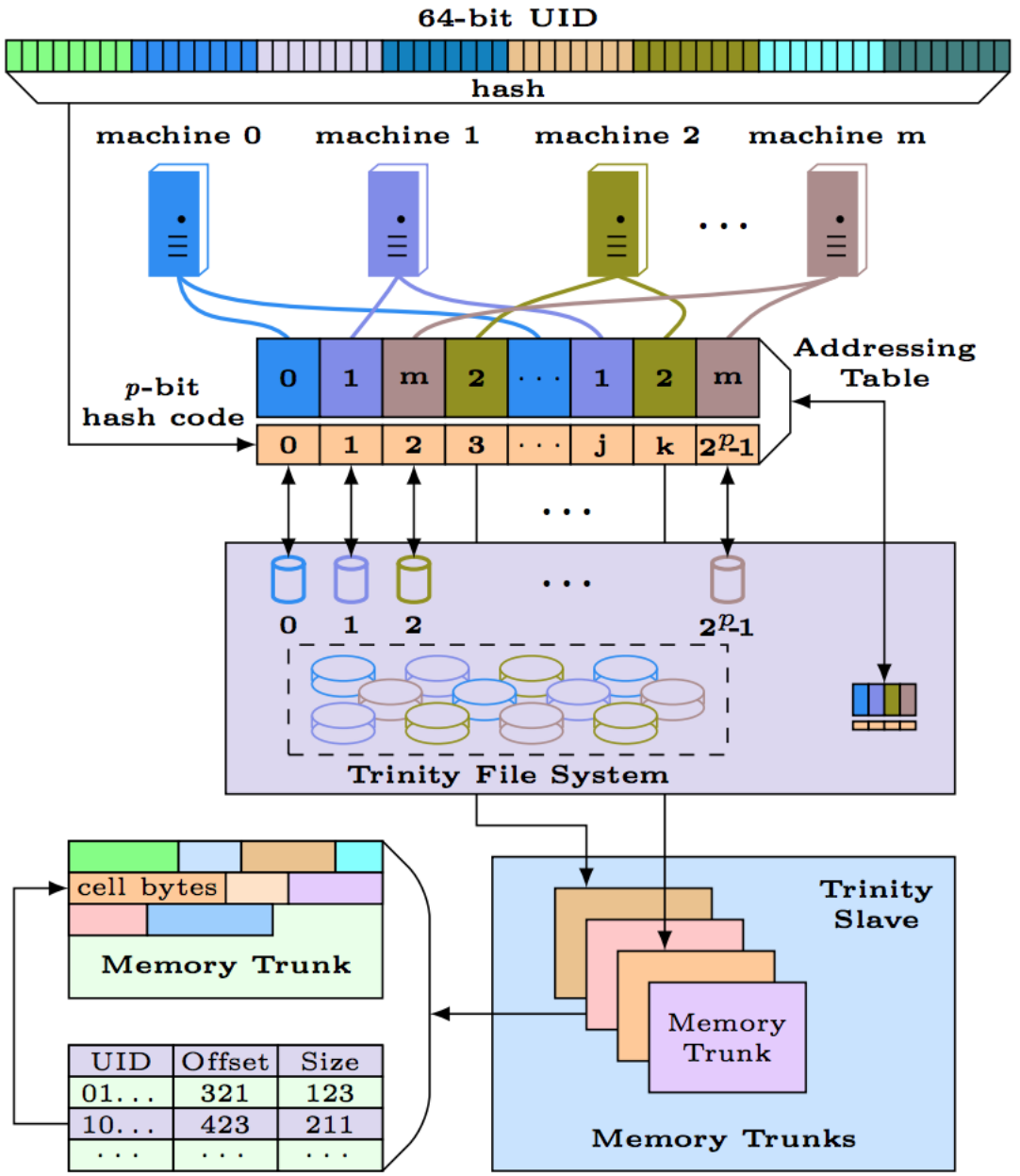
Overview

- Storage Infrastructure
- Computation Framework
- Multiple components communicate over a network
  - Slave
    - Store graph
    - Perform computations
  - Proxies
    - Handles messages
  - Clients
    - Enable user interaction
- Trinity Specification Language (TSL)
  - Bridges graph model and data storage



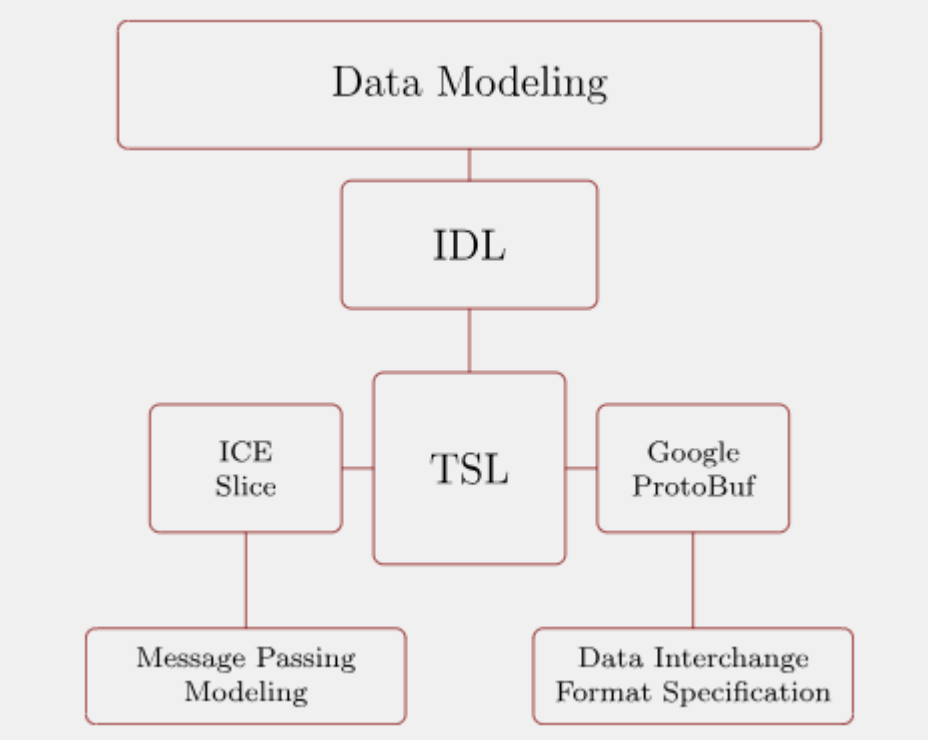
Storage

- Each machine has multiple memory trunks
  - Trunk level parallelism – no locking overhead
  - Single hash table is suboptimal
- Key-Value pair hashing
- Each machine keeps a copy of the addressing table
- Backed up onto Trinity File System (TFS)
- Allows machines to dynamically join/leave memory cloud



Trinity Specification Language (TSL)

- Declarative language
- TSL ties everything together
  - Data modeling
  - Message passing
  - Data interchange
- TSL is compiled into .Net



TSL Basics

- Similar to C/C++ or C#
- Can define protocol
  - Syn
  - Asyn
  - HTTP
- Can store objects
  - Intuitive
  - Inefficient – large overhead
- Instead treat data as blobs

```
[CellType: NodeCell]
cell struct Movie
{
    string Name;
    [EdgeType: SimpleEdge, ReferencedCell: Actor]
    List<long> Actors;
}
[CellType: NodeCell]
cell struct Actor
{
    string Name;
    [EdgeType: SimpleEdge, ReferencedCell: Movie]
    List<long> Movies;
}
```

Sample Network

- Supports edges of different types

Table 1. Characters

Name	Gender	Married	Spouse	Cast
Rachel Green	Female	true	Ross Geller	Jennifer Aniston
Monica Geller	Female	true	Chandler Bing	Courteney Cox
Phoebe Buffay	Female	true	Mike Hannigan	Lisa Kudrow
Joey Tribbiani	Male	false	N/A	Matt Le Blanc
Chandler Bing	Male	true	Monica Geller	Matthew Perry
Ross Geller	Male	true	Rachel Green	David Schwimmer

Sample Network

- Supports edges of different types
- TSL script to model the data
- Two running modes
  - Client mode
  - Embedded modes

```
cell struct Character
{
    String Name;
    byte Gender;
    bool Married;
    long Spouse;
    long Performer;
}

cell struct Performer
{
    String Name;
    int Age;
    List<long> Characters;
}
```

Sample Network

- Supports edges of different types
- TSL script to model the data
- Two running modes
  - Client mode
  - Embedded modes
- Create 12 entity cells
  - 1 for each character
  - 1 for each performer

```
// Characters
Character Rachel = new Character(Name: "Rachel Green", Gender: 0, Married: true);
Character Monica = new Character(Name: "Monica Geller", Gender: 0, Married: true);
Character Phoebe = new Character(Name: "Phoebe Buffay", Gender: 0, Married: true);
Character Joey = new Character(Name: "Joey Tribbiani", Gender: 1, Married: false);
Character Chandler = new Character(Name: "Chandler Bing", Gender: 1, Married: true);
Character Ross = new Character(Name: "Ross Geller", Gender: 1, Married: true);

// Cast
Performer Jennifer = new Performer(Name: "Jennifer Aniston", Age: 43,
Characters: new List<long>());
Performer Courteney = new Performer(Name: "Courteney Cox", Age: 48,
Characters: new List<long>());
Performer Lisa = new Performer(Name: "Lisa Kudrow", Age: 49,
Characters: new List<long>());
Performer Matt = new Performer(Name: "Matt Le Blanc", Age: 45,
Characters: new List<long>());
Performer Matthew = new Performer(Name: "Matthew Perry", Age: 43,
Characters: new List<long>());
Performer David = new Performer(Name: "David Schwimmer", Age: 45,
Characters: new List<long>());
```

Sample Network

- Supports edges of different types
- TSL script to model the data
- Two running modes
  - Client mode
  - Embedded modes
- Create 12 entity cells
  - 1 for each character
  - 1 for each performer
- Directed and undirected relationship
- Hyperedge cell
- 12 Entity cells – 3 relationships

```
Rachel.Performer = Jennifer.CellID;
Jennifer.Characters.Add(Rachel.CellID);
```

```
Monica.Spouse = Chandler.CellID;
Chandler.Spouse = Monica.CellID;
```

```
cell struct Friendship
{
    List<long> friends;
}
```

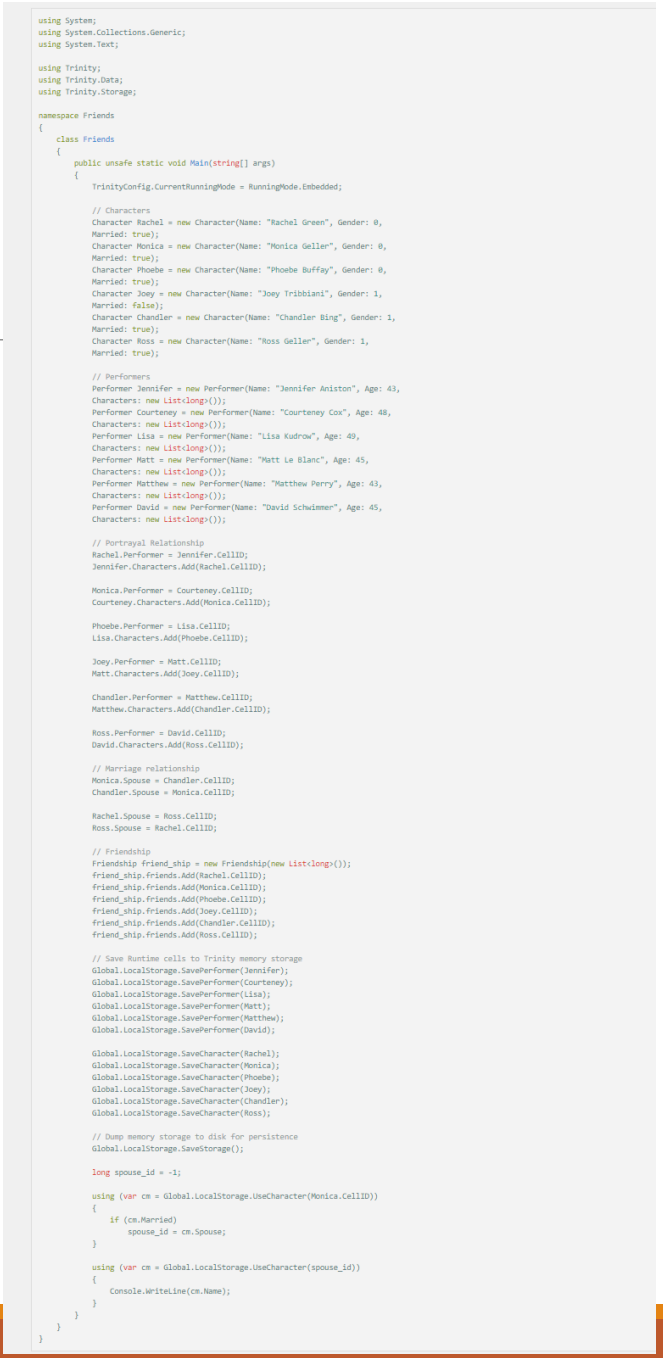
```
Friendship friend_ship = new Friendship();
friend_ship.friends.Add(Rachel.CellID);
friend_ship.friends.Add(Monica.CellID);
friend_ship.friends.Add(Phoebe.CellID);
friend_ship.friends.Add(Joey.CellID);
friend_ship.friends.Add(Chandler.CellID);
friend_ship.friends.Add(Ross.CellID);
```

Trinity Memory Storage

- Save the network on the Trinity's main memory
- Implementation of the friends network in Trinity using the TSL
- Also has a graph generator

```
Global.LocalStorage.SavePerformer(Jennifer);
Global.LocalStorage.SaveCharacter(Rachel);
```

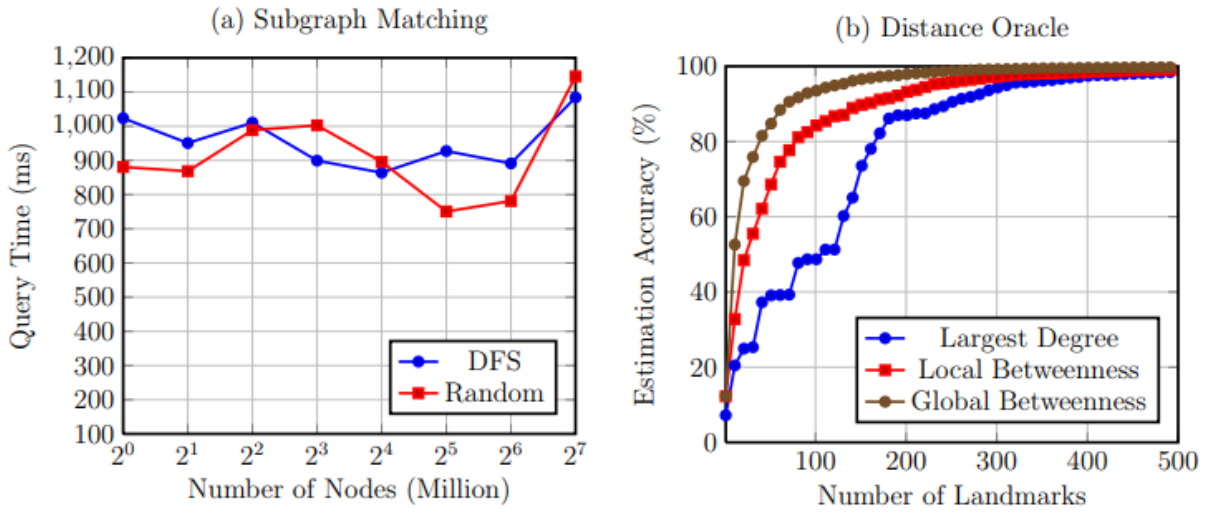
- Is available on GitHub
- Extension on VisualStudio



```
using System;
using System.Collections.Generic;
using System.Text;

using Trinity;
using Trinity.Data;
using Trinity.Storage;

namespace Friends
{
    class Friends
    {
        public unsafe static void Main(string[] args)
        {
            TrinityConfig.CurrentRunningMode = RunningMode.Embedded;

            // Characters
            Character Rachel = new Character(Name: "Rachel Green", Gender: 0,
            Married: true);
            Character Monica = new Character(Name: "Monica Geller", Gender: 0,
            Married: true);
            Character Phoebe = new Character(Name: "Phoebe Buffay", Gender: 0,
            Married: true);
            Character Joey = new Character(Name: "Joey Tribbiani", Gender: 1,
            Married: false);
            Character Chandler = new Character(Name: "Chandler Bing", Gender: 1,
            Married: true);
            Character Ross = new Character(Name: "Ross Geller", Gender: 1,
            Married: true);

            // Performers
            Performer Jennifer = new Performer(Name: "Jennifer Aniston", Age: 43,
            Characters: new List<long>());
            Performer Courteney = new Performer(Name: "Courteney Cox", Age: 48,
            Characters: new List<long>());
            Performer Lisa = new Performer(Name: "Lisa Kudrow", Age: 49,
            Characters: new List<long>());
            Performer Matt = new Performer(Name: "Matt Le Blanc", Age: 45,
            Characters: new List<long>());
            Performer Matthew = new Performer(Name: "Matthew Perry", Age: 43,
            Characters: new List<long>());
            Performer David = new Performer(Name: "David Schwimmer", Age: 45,
            Characters: new List<long>());

            // Portrayal Relationship
            Rachel.Performer = Jennifer.CellID;
            Jennifer.Characters.Add(Rachel.CellID);

            Monica.Performer = Courteney.CellID;
            Courteney.Characters.Add(Monica.CellID);

            Phoebe.Performer = Lisa.CellID;
            Lisa.Characters.Add(Phoebe.CellID);

            Joey.Performer = Matt.CellID;
            Matt.Characters.Add(Joey.CellID);

            Chandler.Performer = Matthew.CellID;
            Matthew.Characters.Add(Chandler.CellID);

            Ross.Performer = David.CellID;
            David.Characters.Add(Ross.CellID);

            // Marriage relationship
            Monica.Spouse = Chandler.CellID;
            Chandler.Spouse = Monica.CellID;

            Rachel.Spouse = Ross.CellID;
            Ross.Spouse = Rachel.CellID;

            // Friendship
            Friendship friend_ship = new Friendship(new List<long>());
            friend_ship.friends.Add(Rachel.CellID);
            friend_ship.friends.Add(Monica.CellID);
            friend_ship.friends.Add(Phoebe.CellID);
            friend_ship.friends.Add(Joey.CellID);
            friend_ship.friends.Add(Chandler.CellID);
            friend_ship.friends.Add(Ross.CellID);

            // Save Runtime cells to Trinity memory storage
            Global.LocalStorage.SavePerformer(Jennifer);
            Global.LocalStorage.SavePerformer(Courteney);
            Global.LocalStorage.SavePerformer(Lisa);
            Global.LocalStorage.SavePerformer(Matt);
            Global.LocalStorage.SavePerformer(Matthew);
            Global.LocalStorage.SavePerformer(David);

            Global.LocalStorage.SaveCharacter(Rachel);
            Global.LocalStorage.SaveCharacter(Monica);
            Global.LocalStorage.SaveCharacter(Phoebe);
            Global.LocalStorage.SaveCharacter(Joey);
            Global.LocalStorage.SaveCharacter(Chandler);
            Global.LocalStorage.SaveCharacter(Ross);

            // Dump memory storage to disk for persistence
            Global.LocalStorage.SaveStorage();

            long spouse_id = -1;

            using (var cm = Global.LocalStorage.UseCharacter(Monica.CellID))
            {
                if (cm.Married)
                    spouse_id = cm.Spouse;
            }

            using (var cm = Global.LocalStorage.UseCharacter(spouse_id))
            {
                Console.WriteLine(cm.Name);
            }
        }
    }
}
```

Graph Computation - Query

- Facebook: find all Davids within 3 hops
- Efficient memory-based graph exploration
  - Result:
    - Synthetic Facebook like network
    - 800 million nodes and 104 billion edges
    - 130 average edges per nodes
    - Solves this problem in 100 milliseconds
    - Sends asynchronous requests recursively to remote machines → efficient memory access + optimized network communication
- Fast random access + parallel computing
  - Average query size is 10 nodes → average time is 1 second



Graph Computation – Offline

- Vertex based computation model
- Super-step
  - Receive messages from a fixed set of vertices (usually its neighbors)
  - Send messages to another set
  - Modify values
- Trinity can adopt any computation model
- Not constrained by any model

Results

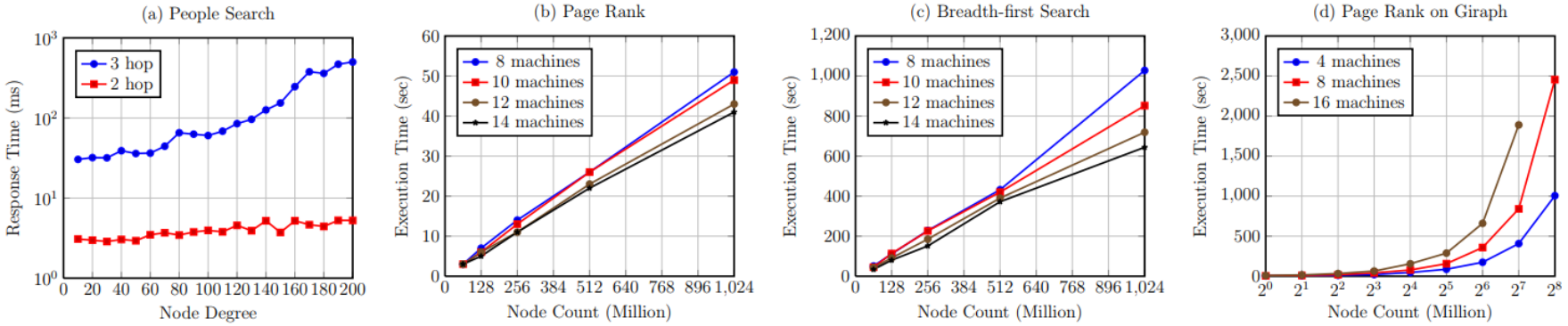


Figure 12: Trinity Performance Experiments

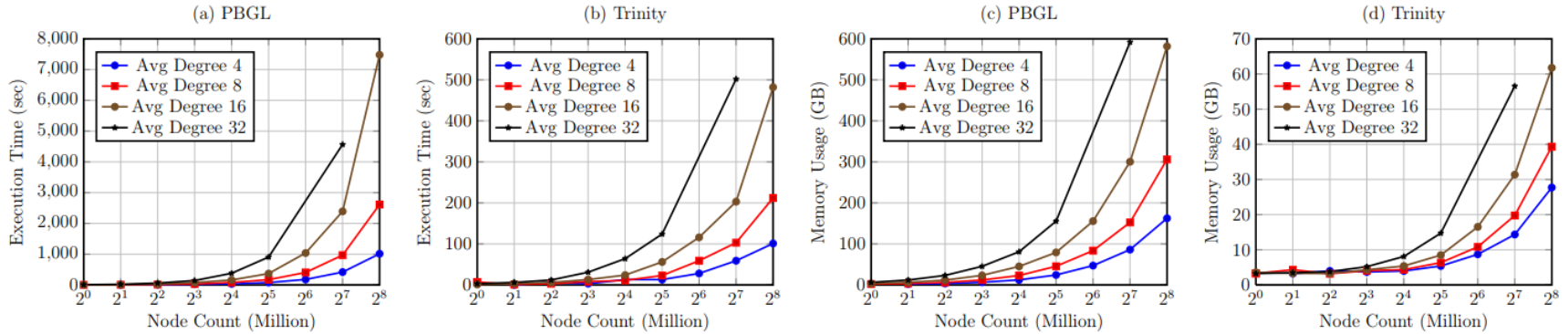


Figure 13: BFS in PBGL and Trinity

References

- https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Trinity.pdf
- https://www.graphengine.io/