XGDBench: A Benchmarking Platform for Graph Stores in Exascale Clouds

Miyuru Dayarathna
Department of Computer Science
Tokyo Institute of Technology
Tokyo, Japan
dayarathna.m.aa@m.titech.ac.jp

Toyotaro Suzumura
Department of Computer Science
Tokyo Institute of Technology/IBM Research - Tokyo
Tokyo, Japan
suzumura@cs.titech.ac.jp

Abstract—In recent years many online graph database service providers have started their operations in public clouds due to the growing demand for graph data storage and analysis. While this trend continues to rise there is little support available for benchmarking graph database systems in cloud environments. In this paper we introduce XGDBench which is a graph database benchmarking platform for cloud computing systems. XGDBench has been designed to operate in current cloud service infrastructures as well as on future exascale clouds. We extend the famous Yahoo! Cloud Serving Benchmark to the domain of graph database benchmarking. The benchmark platform is written in X10 which is a PGAS language intended for programming future exascale systems. We describe the architecture of the XGDBench focusing on its importance for exascale clouds. We did a cluster analysis to compare the realistic nature of XGDBench data generator and saw that the community structures of synthetic graphs produced by XGDBench outperforms RMAT. We also conducted performance evaluation of four famous graph data stores AllegroGraph, Fuseli, Neo4j, an OrientDB using XGDBench on Tsubame 2.0 HPC cloud environment.

Keywords—Database systems; Benchmark testing; Network theory (graphs); System performance; Performance analysis;

I. INTRODUCTION

Recently, graphs have become an important workload in cloud systems due to the rapid increase of applications that produce data in the form of graphs. Semantic web applications, geographic information systems, bioinformatics applications [12], cheminformatics applications [14] are some examples for such application areas. Historically, graph data have been modeled in relations and was stored in relational databases. Complex analysis which involved graph traversal has been implemented in application logic. However, it has been shown that graph data storage and analysis in the form of graphs is more effective since it introduces optimized performance and query specification productivity. Many commercial and open source graph databases have appeared in recent times to cater this need. Furthermore, multiple cloud services which offer data storage in the form of graphs have started their operations in recent times. NuvolaBase [29], Dydra [13], CloudGraph [6], etc. are some examples for such services that operate on SaaS model.

The main difference of the graph databases from the relational databases is that graph databases use network data model. In a graph database an entity is represented by a vertex and its associated set of attributes. Furthermore, the relationships between the entities are represented as edges and their attributes. Graph databases have a close relationship to RDF (Resource Description Framework) stores since RDF stores use triples which could be used to represent entities and the relationships between them. Hence RDF stores are also categorized as graph databases.

There have been benchmarks for graph databases such as HPC Scalable Graph Analysis Benchmark [2] which focus on some core network analysis features of Graph databases. However, a key point missing in these benchmarks is that the application scenarios modeled by the prior benchmarks does not realistically model real application scenarios. Many of current graph applications involve graphs with vertex and edge attributes (also known as colorful graphs). Furthermore, there are a plethora of benchmarks developed by Semantic Web community such as LUBM [17], SP2Bench [32], DBpedia [24] that could be also used to benchmark graph databases. However these benchmarks are not smoothly scalable in the sense that they do not follow a statistical model.

Furthermore, we are heading towards the Exascale computing era which is predicted to appear in 2018-2020 time frame [11]. Exascale technology will revolutionize future cloud computing paradigm by creating power efficient cloud computing systems that offer huge performance per watt values. However, completely new programming techniques and models need to be used to program such Exascale clouds. Use of PGAS (Partitioned Global Address Space) languages for programming such systems is one approach in achieving the Exascale.

X10 [5] is a new programming language of Asynchronous PGAS family, a potential candidate for programming such Exascale clouds. However, not much work has been conducted to observe the applicability of such programming models for cloud computing systems. By creating XGDBench we also aim to fulfill this important requirement.

Considering the aforementioned issues associated with the current benchmarks and the trend in HPC clouds, we propose a different benchmarking platform that is suited for benchmarking graph databases in Exascale systems. Our benchmark is based on Multiplicative Attribute Graph model (MAG) [26] which is a synthetic graph model for attribute graphs. Our focus on this paper is not only introducing the design of

The original version is available from IEEE Xplore: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6427516&contentType=Conference+Publications&sortBy%3Dasc_p_Sequence%26filter%3DAND%28p_IS_Number%3A6427477%29%26pageNumber%3D2
XGDBench, but also to demonstrate its applicability for future exascale system benchmarking. The contributions of this paper can be listed as follows:

- **Graph database benchmarking platform** - We describe the architecture of XGDBench which is a benchmark targeted for graph databases.
- **Benchmarking Exascale Clouds** - Our benchmarking platform is aimed for benchmarking graph databases that operate in future Exascale clouds.
- **Workload characterization of graph databases** - We evaluate performance of four famous graph database servers AllegroGraph [20], Fuseki [1], Neo4j [28], and OrientDB [34] on Tsubame [15] HPC cloud environment using XGDBench.

The paper is organized as follows. We describe the related work for XGDBench in Section II. Next, we provide an overview to Exascale graph database benchmarking in Section III. Methodology of XGDBench is described in Section IV. We describe requirements for a graph database benchmark in Section V. We provide implementation details and the evaluation conducted on XGDBench in Sections VI and VII respectively. A discussion and the limitations of current XGDBench prototype are provided in Section VIII. We conclude in Section IX.

II. RELATED WORK

Relational databases, the most popular model of DBMSs have standard benchmark suites such as TPC [27] benchmark suite. However, the emerging field of Graph DBMSs still lacks such support. We review current-state-of-the-art efforts in graph database benchmarking and explain how XGDBench enhances over them.

HPC Scalable Graph Analysis Benchmark is composed of four separated operations on a graph that follows a power-law distribution [2]. However, this benchmark does not evaluate some features that are inherent to Graph Databases such as object labeling, attribute management, etc. [2] a feature that will dominate future graph database landscape that we address in XGDBench. Dominguez-Sal et al., made a survey of graph database performance on the HPC Scalable Graph Analysis Benchmark [10]. They evaluated the performance of four graph databases/stores Neo4j, Jena, HypergraphDB, and DEX. They also provide a discussion on graph database benchmarking with the goal of allowing practitioners to have a set of basic guidelines for graph database benchmarking [9]. However, a limitation of their work is that it only implements HPC Benchmark and does not consider attribute graphs.

Ciglan et al. described a benchmark for graph traversal operations on graph databases [23]. They extend the discussion on the design of graph database benchmarks focusing on traversal operations in a memory constrained environment where whole graph cannot be loaded and processed in memory. Similar to them, XGDBench implements graph traversal as one of the workload items. However, unlike them our study is purely based on graph database servers and we do not focus on benchmarking embedded graph databases. It is because XGDBench is a benchmarking framework rather than a benchmark specification.

Vicknair et al. made a performance comparison of Neo4j graph database and MySQL [36]. But their study did not focus specifically cloud environments.

There have been works on creating benchmarks for data stores that store graph information such as Lehigh University Benchmark (LUBM) [17], Berlin [3], DBpedia [24], SP2Bench [32]. Yet, none of these benchmarks employ statistical graph generator model which allows very large scale, realistic synthetic graphs.

Rohloff et al. conducted an evaluation of triple-store technologies for large data stores [30]. Triple-stores also have been used as graph database management systems in various occasions. We use AllegroGraph a famous triple-store (which is also popular as a graph database) for evaluation of XGDBench due to this reason. However, Rohloff et al.'s work was conducted using the LUBM and their study focused on evaluating triple store technologies. Another similar work on benchmarking RDF stores has been conducted by Thakker et al. [35]. However, they used the University Ontology Benchmark (UOBM) [22] for this purpose.

Graph 500 is a new benchmark suite intended for benchmarking data intensive supercomputing applications [25]. The intended application scenario is a compact application that has multiple analysis techniques accessing a single data structure representing a weighted, undirected graph. Current implementation of Graph 500 does not consider different application scenarios such as graph databases. Its focus is on benchmarking super computer performance. Graph500 has been implemented in C++ using MPI. Different from Graph500, we implemented XGDBench framework using X10 which is aimed for providing productive programming on future Exascale systems.

In contrast to Graph500, XGDBench framework has been implemented in X10 which provides productive programming future Exascale systems compared to use of MPI.

III. BENCHMARKING GRAPH STORES IN EXASCALE

XGDBench is an extension of famous Yahoo! Cloud Serving Benchmark (YCSB) [7]. In this section we provide a brief introduction to YCSB. Also we describe some popular synthetic graph models currently used for benchmarking graph processing systems. We explain the MAG model which we use in our work and describe the reason why it is a better graph generator for graph database benchmarking. We also provide brief background to X10 programming language.

A. Yahoo! Cloud Serving Benchmark

YCSB [7] framework was released by Yahoo! Inc. in 2010, with the motivation of creating a standard benchmark and benchmarking framework to assist evaluation of cloud data serving systems. One of the key goals of YCSB is its extensibility. The framework is composed of a workload generator client and a package of standard workloads that cover interesting parts of the performance space. The workload generator
of YCSB supports definition of new workload types which motivated us for following YCSB’s approach for implementing XGDBench.

B. Synthetic Graph Generators

As mentioned by Chakrabarti et al. [4] graph models can be classified into five categories such as, Random graph models (e.g., Erdős Rényi), Preferential attachment models (e.g., Barabasi-Albert model), Optimization-based models (e.g., Highly Optimized Tolerance model), Tensor-based models (R-MAT), and Internet-specific models (e.g., Inet). Each generator model has its own pros/cons and the best generator model depends on the application area. We summarize R-MAT model which has also been used as a data generator for Graph500. R-MAT (Recursive MATrix) generates graphs by recursively traversing the adjacency matrix of a graph without any edges in recursive manner. Graphs generated by R-MAT depends on five parameters scale(n),a,b,c, and d. The parameters a,b,c, and d are floating point values whose sum is 1. The number of vertices in an R-MAT graph is set to \(2^n\).

C. X10 - A brief Overview

X10 is an open source programming language that is aimed for providing a robust programming model that can withstand the architectural challenges posed by multi-core systems, hardware accelerators, clusters, and supercomputers [19][5]. The main role of X10 is to simplify the programming model in a way that it leads to increase in programmer productivity for future systems such as Extreme-scale computing systems [11]. X10 is a strongly typed, object-oriented language which emphasizes static type-checking and static expression of program invariants. The latest major release of X10 is X10 2.2.0 of which the applications are developed by source-to-source compilation to either C++ (Native X10) or Java (Managed X10). We used managed X10 when developing XGDBench because we get support from X10 for smooth extension of YCSB framework. Furthermore, the distributed data structures (e.g., DistArray) present in X10 allows for distributed storage of large graphs that could not be stored in single place. Moreover, the object-oriented, easy programmability features of X10 allows for writing extensions for XGDBench for future Exascale graph stores with less effort. While more details of X10 language could not be provided due to space limitations, we refer the reader to X10 website [19] for further details.

D. An overview of Multiplicative Attribute Graphs

Multiplicative Attribute Graph (MAG) is an approach for modeling the structure of networks which have node attributes [26]. MAG naturally models the interactions between the network structure and the node attributes. Compared to Kronecker graphs (used in graph 500) that follow R-MAT model, MAG model creates realistic attribute graphs which are much suited for benchmarking graph databases due to the aforementioned reason. It has been proven that MAG generates graphs with both analytically tractable and statistically interesting properties [26].

A schematic representation of general MAG algorithms is shown in Figure 1. The Figure 1 shows two vertices \(v\), and \(u\) each having a vector of \(n\) categorical attributes and each attribute have a cardinality \(d_i\) for \(i = 1, 2, ..., n\). There are also \(n\) matrices denoted by \(\Theta_i\), \(\Theta_i \in d_i \times d_i\) for \(i = 1, 2, ..., n\). Each entry of \(\Theta_i\) is the affinity of a real value between 0 and 1. Values \(\alpha, \beta, \gamma\) are floating point values between 0 and 1. Given these conditions, probability of an edge \((u,v)\), \(P(u,v)\), is defined as the multiplication of affinities of affinities corresponding to individual attributes as shown in Figure 1.

\[
\begin{align*}
\mathbf{v} &= [0, 0, 1, 0] \\
\mathbf{u} &= [0, 1, 1, 0]
\end{align*}
\]

\[
\mathbf{P}_{[v,u]} = \alpha_0 \times \beta_1 \times \gamma_2 \times \alpha_3
\]

Fig. 1: Multiplicative Attribute Graphs (MAG) model

However, the MAG algorithm used in XGDBench generator is a simplified version of the model shown in Figure 1. The simplification is done by considering undirected version of the model by making each \(\Theta_i\) symmetric. The node attribute values are made binary (e.g., The attributes may represent yes/no answers received for a question asked by each member of a social network). This makes the \(\Theta_i\) to be a \(2 \times 2\) matrix. Also it is assumed that all the affinity matrices are equal (i.e., \(\Theta_i = \Theta\)). Our implementation of the MAG algorithm is shown in Algorithm 1. It accepts number of vertices of the generated graph \(nVertices\), number of attributes per vertex \(nAttributes\), a threshold value for random initialization of attributes \(attribThresh\), an edge affinity threshold value that determines whether there is an edge between two vertices, and an affinity matrix \(\text{theta}\) corresponding to the \(\Theta\) mentioned above. The randZeroOrOne() function in Algorithm 1 constructs a zero initialized matrix of size \(nVertices \times nAttributes\). Then the matrix is populated with 1s using randomly generated values if they exceed the \text{attribThresh}. This matrix is returned to Algorithm 1’s \text{nodeAttribs} variable. The key feature why MAG model is much suited for graph database benchmarking is that the probability of an edge between pairs of vertices is controlled by the product of individual attribute-attribute affinities. Most of the current graph databases are made to store not only vertices and their relationships, but also attributes of these vertices and relationships. Therefore, MAG is a more natural synthetic graph model suitable for benchmarking graph databases.

IV. METHODOLOGY OF XGDBENCH

Almost every software benchmark has been developed around a real world application scenario of the software
Algorithm 1 mag(nVertices, nAttributes, attribThresh, pThresh, theta)

1: nodeAttribs ← randZeroOrOne(nVertices, nAttributes, attribThreshold)
2: result ← ones(nVertices, nVertices)
3: for i ← 0 to nVertices do
4:     for j ← 0 to nVertices do
5:         for k ← 0 to nAttributes do
6:             if nAtt[i,k] = nAtt[j,k] then
7:                 if nAtt[i,k] = 0 then
8:                     result[i,j] = result[i,j] * theta[0]
9:                 else
10:                    result[i,j] = result[i,j] * theta[3]
11:                 end if
12:             else
13:                 if nAtt[i,k] = 0 and nAtt[j,k] = 1 then
14:                     result[i,j] = result[i,j] * theta[1]
15:                 else if nAtt[i,k] = 1 and nAtt[j,k] = 0 then
17:                 end if
18:             end if
19:         end for
20:     end for
21: end for
22: for i ← 0 to nVertices do
23:     for j ← 0 to nVertices do
24:         if result[i,j] > pThresh then
25:             result[i,j] = 1
26:         else
27:             result[i,j] = 0
28:         end if
29:     end for
30: end for
31: return (result)

A. Attribute Read/Update

Graph databases in Exascale clouds will have to handle massive graphs online and they will partially load the graph into memory. The workloads will include both read/update operations. However, in most of the future exascale applications the Read operations will dominate the workload [8]. Therefore, we included read-heavy (e.g., a workload with 0.95 probability of read operation and 0.05 probability of write operation [7]) and read-only (having only read operations) workloads with XGDBench.

Graphs need to be updated online. In a typical OSN a node represents a user and an edge represents friendship/relationship. Properties of nodes/edge include messages, photos etc. The friendship graph of OSNs change at a slower rate compared to their properties. Therefore, performance of attribute update operation is critical compared to node/edge update.

Moreover, the Benchmarking platform needs to be scalable to store data in-memory for update operations. This will eliminate unexpected delays involved in reading large data from secondary storage.

B. Graph Traversal

Compared to other databases, graph databases have unique property of having data encoded in its graph structure that could be obtained by traversing them. Therefore, the benchmark should have support for evaluating the performance of graph traversal operations. While there are a variety of graph traversal techniques, we decided to use an algorithm that will be most frequently executed against the graph database. This is because it is important to check the performance of frequently used operations than operations that ran infrequently which does not have requirements for real-time execution. We selected a scenario of listing friends of friends which is one of the frequently used traversal operations in OSNs. This includes execution of BFS (breadth-first search) from a particular vertex for detecting the connected component of a graph.

Based on the aforementioned requirements we define the following set of basic operations on a graph database (Shown in Table I). We believe that these basic operations are sufficient for defining many workloads that are frequently present in graph databases.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Read a vertex and its properties</td>
</tr>
<tr>
<td>Insert</td>
<td>Inserts a new vertex</td>
</tr>
<tr>
<td>Update</td>
<td>Update all the attributes of a vertex</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete a vertex from the DB</td>
</tr>
<tr>
<td>Scan</td>
<td>Load the list of neighbors of a vertex</td>
</tr>
<tr>
<td>Traverse</td>
<td>Traverses the Graph from a given vertex using BFS. This represents finding friends of a person in social networks.</td>
</tr>
</tbody>
</table>

V. REQUIREMENTS OF XGDBENCH

In this section we describe the performance aspects that are specifically targeted by XGDBench.
VI. IMPLEMENTATION

Implementation details of XGDBench is shown in Figure 2. XGDBench Client is the software application that is used to execute the benchmark's workloads. It consists of a graph data generator for generating the data to be loaded to the database. The workload generator is implemented using Multiplicative Attribute Graphs (MAG) model [26] as described in the previous section. The workload executor drives multiple client threads. A sequential series of operations are executed by the client threads. Graph database interface layer translates simple requests from client threads into calls against the graph database.

XGDBench has two phases of execution called loading phase and transaction phase. The loading phase generates an attribute graph by using the MAG algorithm shown in Algorithm 1. The transaction phase of XGDBench calls a method in CoreWorkload called doTransaction(), which invokes the basic operations such as database read, update, insert, scan. We have implemented the workloads that satisfies the requirements stated in Section V on XGDBench and those are listed in Table I. We use these workloads on 4 graph database servers and report performance in next section. Note that we currently do not implement the traverse operation related workloads.

The update operations on the graph data preserves the power-law distribution that is present in the original graph created by MAG because the update operations are conducted only on attributes that are not related to calculation of probability of an edge. Furthermore, we make sure the insert operations of the vertices done during the workload executions preserve the power-law structure.

VII. EVALUATION

In the first half of this section we evaluate XGDBench's data generator model's (MAG) properties. First, we compare the degree distribution of MAG with a real world social network graph. Next, we compare MAG with the popular R-MAT model focusing on the properties of the graphs generated by them. In the second half we evaluate the performance characteristics drawn out by XGDBench on four graph DBs.

A. Properties of the MAG Data Generator

We used degree distribution and graph community structure of MAG model to evaluate its properties as follows,

1) Power-law Distribution: Degree distributions of many real world graphs that arise in many applications such as the web and social networks satisfy power-law distribution [16]. Two variables x and y are related to each other by a power-law when,

\[ y(x) = Ax^{-\gamma} \]  

(1)

where A and \( \gamma \) are positive constants [4]. We plotted the degree distribution of a graph with 1000 vertices produced by XGDBench generator (shown in Figure 3(a)). Note that the "inout degree" in Figure 3 means the total number of edges connected to a vertex without considering the edges' directions. We observed that it creates a degree distribution similar to a power-law distribution. We also plotted degree distribution of an online social network called Epinions social network (described in [21]) to compare and confirm that MAG creates power-law degree distributions similar to real world graphs.

2) Categorical prominence within Communities: The data generation module of XGDBench need to model real life graphs that reside on graph databases. As mentioned before, graph databases are designed to store not just plain graphs (with only vertices and edges), but more colorful graphs with vertex and edge attributes. The graph generator should generate such realistic attribute graphs in order for XGDBench to generate realistic workload scenarios. We compare famous non-attribute graph generator model R-MAT followed by famous benchmarks such as Graph 500 with our MAG data generator to observe which model best fits for graph database benchmarking scenarios.

We implemented an R-MAT version of XGDBench by replacing the data generator algorithm with R-MAT algorithm. Furthermore, after generating the R-MAT synthetic graph we randomly populate the vertex attributes to mimic the attribute graphs produced by MAG model. We used five graphs from each model with R-MAT scale (n) 10 to 14 for this purpose.

---

![Diagram](image.png)

**Fig. 2. Architecture of XGDBench Client**
R-MAT graph was generated with parameters \( a=0.6, b=0.15, c=0.15, d=0.1 \) (these values closely resemble the Graph500 R-MAT generator’s initiator parameters). For MAG, we used a probability threshold of 0.01 and attribute threshold of 0.25. Each graph had 4 attributes per vertex. Details of the generated graphs are listed on Table III.

Next, we conducted a community cluster analysis on each graph. We used Cytoscape [33] which is a platform for complex network analysis and visualization for this purpose. The vertices in the top 3 resulted clusters were further clustered based on the vertex attributes. Next, we take the percentages of vertex counts in each sub cluster and rank them based on their percentage values. We defined cluster prominence metric \( C_p \) as the percentage difference between the largest sub cluster and the second largest sub cluster and report on the Table III.

From the results we observed that the graphs generated by MAG model had sub clusters with higher prominence indicating that the communities created by MAG represented the phenomenon of social affinity that is present in real social networks.

### B. Performance Evaluation of Graph Databases

We conducted a performance evaluation experiments of OrientDB (v1.0rc9), Allegrograph (v4.6), Neo4j (community v1.6.1), and Fuseki (v0.2.1) using XGDBench. The experiments were done on Tsubame 2.0 [15] cloud computing environment (A single node’s specifications are given on Table IV). We used two nodes in each experiment. The graph database server was set up in one node and XGDBench was run on the next node. We set the JVM heap for Neo4j to 4GB and in the case of OrientDB and Fuseki to 2GB. Since Allegrograph is LISP based server we need not to set JVM heap size. For XGDBench we set up 8GB heap for X10 runtime. We used X10 runtime 2.2.2 which is build with fully optimized settings. XGDBench generated graphs with 1024 vertices during these evaluations.

### VIII. Discussion and Limitations

From the results we observed that XGDBench’s graph generator model is much suited for evaluating performance of graph databases. Furthermore, the attribute graphs produced by MAG follow the power-law distributions which enables realistic benchmarking scenarios. Form the performance evaluation of the graph DBs we observed that OrientDB performed well during the data loading phase and with all the workloads.

Most of the current graph databases (including the 4 databases used in this study) are not distributed which is a limitation in this study. Furthermore, we believe that 1024 vertices is not enough to thoroughly characterize performance of graph databases. However, we had to use 1024 vertices because some database servers used in this evaluation performed poorly which made our experiments restricted to 1024 vertices.
IX. CONCLUSION AND FURTHER WORK

In this paper we described XGDBench, a graph database benchmarking framework for Exascale clouds. The data generator of XGDBench is based on MAG Model which enables realistic modeling of attribute graphs. XGDBench has been implemented using X10 which enables XGDBench to be extended easily to support benchmarking future Exascale graph databases. We evaluated the applicability of MAG model for graph database benchmarking and also conducted a performance evaluation. From the cluster analysis we observed that MAG model creates much realistic attribute graphs compared to the popular RMAX model which is used in several graph based benchmarks. In future we hope to conduct thorough evaluation of graph databases using XGDBench. We will also implement traverse operation based workloads. Also we hope to investigate the reasons for why graph databases perform poorly and find path ways to improve their performance.

X. ACKNOWLEDGMENTS

This research was supported by the Japan Science and Technology Agency’s CREST project titled “Development of System Software Technologies for post-Peta Scale High Performance Computing”.

REFERENCES


