Handling Big Data Using a Data-Aware HDFS and Evolutionary Clustering Technique

Mustafa Hajeer, Member, IEEE, and Dipankar Dasgupta, *Fellow, IEEE*

Abstract— The increased use of cyber-enabled systems and Internet-of-Things (IoT) led to a massive amount of data with different structures. Most big data solutions are built on top of the Hadoop eco-system or use its distributed file system (HDFS). However, studies have shown inefficiency in such systems when dealing with today's data. Some research overcame these problems for specific types of graph data, but today's data are more than one type of data. Such efficiency issues lead to large-scale problems, including larger space required in data centers, and waste in resources (like power consumption), that in turn lead to environmental problems (such as more carbon emission) [1], as per scholars. We propose a data-aware module for the Hadoop eco-system. We also propose a distributed encoding technique for Genetic Algorithms. Our framework allows Hadoop to manage the distribution of data and its placement based on cluster analysis of the data itself. We are able to handle a broad range of data types as well as optimize query time and resource usage. We performed our experiments on multiple datasets generated via LUBM.

Index Terms— Clustering methods, Distributed Computing, Information Management, Optimization, Scalability

1 Introduction

Building a science out of data faces many challenges. One major problem is that today's data is big, dynamic, and heterogeneous, collected from multiple sources and frequently has no standard structure.

The majority of modern data analytics, management tools and services are designed to use Hadoop Distributed File System (HDFS) as a data warehouse; sometimes these analytic tools use services provided by the Hadoop ecosystem for processing. From a price/performance standpoint, Hadoop stands well.

The flexibility Hadoop provides to scale on data management problems is the reason why users perform inefficiently as per [1]. As per Huang et al. (a) the way users add machines to overcome computation issues made them focus less on how their codes use resources, and (b) many HDFS users are convinced that it is designed for batch processing. Hence, it's okay to have the codes running for a long time in the background without even thinking about the resources these processes are using.

In Bajda-Pawlikowski et al. work *Hadapt* [2], the authors gave an example of such inefficiency, and overcame it for structured data by a factor of 50. However, the enterprise data explosion is mostly semi, multi and unstructured according to Michael Walker in the survey he referenced in his blog [3]. The International Data Corporation (IDC) estimates that the volume of digital data will grow 40 to 50 percent per year [4]. By 2020, IDC [4] predicts the number will have reached 40 Zettabytes (ZB). By 2020, the world will generate 50 times the amount of data and 75 times the number of data containers [3]. There is an intense need for the current data analytic tools to scale on big data and process it efficiently to utilize the resources.

Rohloff et al. in 2011 [5] explained how to store graph data in Hadoop using a representation of triples. They also showed how to perform sub-graph pattern matching in a scalable fashion on graphs of data. Even though the focus was Semantic Web graphs, the techniques presented in the paper are generalizable to other types of graphs. The system SHARD was a result of that paper. Its techniques support Hadoop with the capacity to scale sub-graph pattern matching.

In 2011, the work of Huang et al. on Scalable SPARQL querying of large RDF graphs [1] showed an efficiency problem in the techniques presented by Rohloff et al. [5]. Huang et al. [1] introduced a factor of 1340 times less efficient in Rohloff et al. [5] than other alternative techniques for processing sub-graph pattern matching queries within a Hadoop-based system.

In some situations, Big Data solutions do not use HDFS as a storage. However, they use the same methodology of horizontal scalability. We proposed and experimented with solutions that work on the core HDFS and can be generalizable in those cases. Examples of such tools that use HDFS as storage are Apache Spark [6] and Mesos [7]. An example of a system that supports Hadoop through Yarn resource negotiator and HDFS as a data source is HAMR [8]. Hence, optimizing HDFS with the proposed data-aware HDFS framework will lead to optimizing a large number of current big data solutions.

Spark [6] and Storm [9] are the colorful new Big Data toys. Apache Storm [9] is another big data solution that uses yarn to run real-time analysis on unbounded streams of data. Storm is building on what Hadoop did for batch processing. Some efforts have been made on optimizing Storm. An example of scheduling optimization is found in

• *Mustafa Hajeer Ph.D. at computer science department – The University of Memphis, Memphis, TN 38152 USA (e-mail: mhhajeer@gmail.com). Mustafa also works at Intel Data Center Group.*
• *Dipankar Dasgupta is a Professor of Computer Science at The University of Memphis, a Director of Center for Information Assurance as well as a Director of Intelligent Security Systems Research Laboratory, Memphis, TN 38152 USA (e-mail: ddasgupt@memphis.edu).*

[10], and extensions to Storm were proposed in [11]. Our focus in this study is on optimizing HDFS as a data warehouse (where data already resides on HDFS). However, in some cases, Storm [9] processes streams of data and stores the output results for further processing on HDFS, where our approach facilitates the data-analytic packages around Hadoop.

As concerns about semi/multi/unstructured data growth and the ability to process such data efficiently became a lead concern, our primary goal aimed at improving the Hadoop distributed file system (HDFS) efficiency to handle modern data and to improve utilization of HW resources. Even though techniques in SHARD [5] and Scalable SPARQL [1] are generalizable, there are some limitations on how to deal with modern data. Furthermore, there are other limitations on how to handle the stream of dynamic changes that occur in the data. One may add to these limitations the scalability and storage usage of the clustering and placement algorithm that was used in such work, as discussed in Hajeer et al. [12].

We transformed the data and stored it in graph-based scalable stores to give it a sense of structure and to be able to stream changes, constructing vertex-to-vertex triples for data points, then adding cluster affiliation data to these triples to form quadruples as described in the architecture section. These techniques allowed us to (1) collect data from multiple sources and convert them into quads with a sense of structure for different data, (2) stream changes dynamically and push to the graph database, and (3) prepare the data for application to a new version of Hajeer et al. [12]. A novel encoding of chromosomes was used to handle the modern data clustering problem along with novel crossover, mutation and evaluation techniques to deliver the needs of the new distributed encoding technique. Later, we distributed the sub-graphs over HDFS based on the cluster affiliations to produce optimized data to query and process.

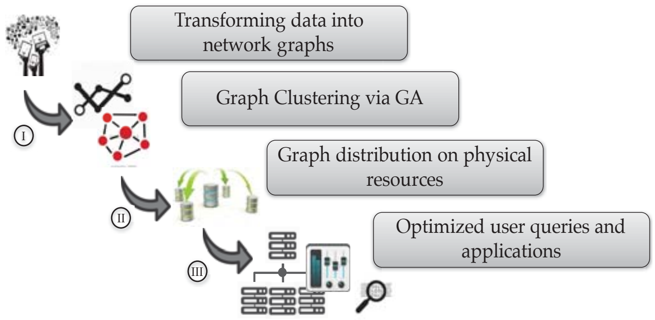

Transforming data into network graphs
Graph Clustering via GA
Graph distribution on physical resources
Optimized user queries and applications
I
II
III

Fig. 1. Computational steps of the proposed framework.

Fig. 1 illustrates the contribution and modules on the proposed framework as follows: (1) after collecting the data or gathering old datasets, this module converts the data into the desired network graphs; (2) finding patterns in the graphs, the module distributes the data into the right data blocks; (3) distributes the blocks into the right machine accordingly; and (4) an optimized DHFS serves as a data source for services to execute queries and provide a platform to apply graph algorithms efficiently as well as reduce resource usage.

To summarize the above, the proposed framework improves the ability of HDFS to handle modern data by building data awareness modules that detect, distribute, and manage data over the scalable file system. Thus, the framework results in optimization and efficient resource usage of the Hadoop eco-system and other tools and services that use HDFS as a distributed storage.

Promising solutions in next generation analytics and lambda architecture have been presented in recent studies. Song et al. [13] reviewed the recent research in data types, storage models, analysis methods and application to network Big Data. They also summarized the challenges and development of big data to predict current and future trends. Song et al. [13] showed how streaming and real time data has been accompanied by the rise in online streaming services, and they also showed how a system based on SQL called DBStream [14] relies on surveys for continuous data analysis.

Studies [15], [16], [17] and [18] have focused on real time analysis, including efforts on lambda architecture, where the authors in [18] have introduced an architecture for time-critical Big Data systems. They showed how current Big Data infrastructures lack the requirement to work with time-critical applications and only focus on the general-purpose applications. Basanta-Val et al. [18] proposal addressed the issue from the perspective of the real-time systems community. Their architecture considered the time-critical (TC) analytic as a group of TC off-line batch processing and TC on-line stream processing. Basanta-Val et al. [18] transformed the general purpose Big Data stack into a TC Big Data stack by imposing the requirement and challenges of TC applications on the general-purpose stack.

T-Hoarder [19] is a framework that collects tweets along with the associated spatiotemporal data; it also displays summarized and analytical information about the Twitter activity with respect to a certain topic or event in a webpage. Studies in next-generation analytics and lambda architecture along with Apache Kudu [20] and a set of studies in [21] proved to be fast and more efficient in processing of OLAP workloads and showed a strong performance in running time-critical workloads. It is worth the effort, however, to study the impact of intelligent data placement on such methods, especially with new technologies that can reduce the tradeoff of data processing, such as Intel's first public demo of persistent memory in SAPPHIRE2017 [22]. Intel showed a new type of persistent memory with huge space compared to DRAM and less latency compared to SSDs.

2 Scope of Work

The current workloads running on systems (where inefficiency exists) lead to requiring more space in data centers, and some severe environmental consequences from the increased carbon emissions due to the extra power consumption [1]. This can influence enterprises because of the additional power consumption and low performance for the same hardware resources. We need systems to scale efficiently.

Graph theory is a well-studied discipline. Dr. Roy Marsten noted in his blog [23] that Graph Theory was a key approach in understanding and leveraging big data. Dr. Marsten focused on how Google started the graph analysis trend in the modern era using links between documents on the Web to understand their semantic context. As a result, "Google produced a Web search engine that massively outperformed its established competitors and saw it jump so far ahead that 'Google' became a verb" [23]. Lots of data can be transformed into graphs. Conversely, many problems can be transformed into graph problems. Using graph theories and algorithms, most of these problems can be solved efficiently.

We consider SHARD within the work of Rohloff et al. [5] as a win for today's data, since we are proposing a way to convert various kinds of data into quadruple graphs, as we show in the architecture section. There are ways to scale graph data using distributed eco-systems, such as Hadoop, as we discuss in the graph databases section. Furthermore, a larger success occurred when Huang et al. [1] adopted the scholarship of Rohloff et al. [5] and optimized it to deal with RDF data and to overcome limitations concerning techniques in Rohloff et al.

Previous efforts have been achieved for graph queries optimization; SHARD, [5] for example, hash-partitioned the data. However, hashing led to subject-to-object joins limitations in RDF graphs due to the need for moving intermediate data over the network. Another example is Huang et al. [1], where the objects connected to a subject were processed to fall into the same blocks for one or two hops between subject and object (one to two edges travel distance between subject and objects). However, space limitations due to increase in data size were present. Also, there is a limitation of applying such an algorithm to a highly-connected graph. Other works like Sempala [24], or HIVE, PigSPARQL [25] & [26], MapMerge [27] and MAPSIN [28] overcame scalability to some extent. However, such work uses a different storage than triples and relies on the advantages of MR, HIVE and Impala where partitioning can be optimized using our framework.

Frameworks like Sempala, PigSPARQL, MapMerge and MAPSIN use different techniques to store RDF graphs. These frameworks transform all predicates into columns and create tables' schemas accordingly, then transform triples into traditional database records. Such frameworks have a limitation of updating all data when new predicates become available along with new data, and furthermore, updating schemes! Thus, getting far away from the idea of RDF and graph-based databases, where the updates come with new predicates, is easier.

2.1 Graph Databases

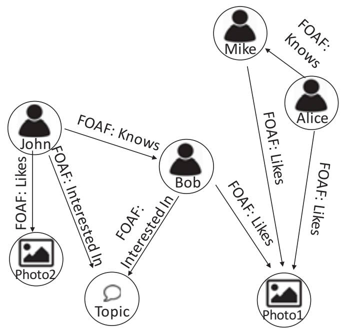
The current data and modern applications have led to limitations in storing and processing using traditional databases, particularly using the relational model. The attention toward graph databases has increased, and the topic that almost died in the early nineties [29] got attention again. The importance of such databases came along with the fact that information in modern data relies on the relations, equally or even more than the information of the entities sometimes [30]. Projects in different fields gave attention to such databases (e.g. Biology [31], semantic web [32], web mining [33] and chemistry [34]).

As per Silberschatz et al. [35], the most general sense of a data model (database model) is a collection of conceptual tools used to model real-world entities and the relations among these entities. As per [35] the three components of this model, from a database point-of -view, are: (1) the set of data structure types, (2) the set of integrity rules and (3) the operators and interface rules.

As per Renzo, "Graph database models can be characterized as those where data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors" [30].

Many graph databases have been developed in research and industry fields. Some differences were found in [30] such as AllegroGraph, DEX, HypergraphDB, InfiniteGraph, Neo4J and Sones. Some of them were even designed to deal with triple data format as described in Huang et al. work [1].

A triple or RDF store is a graph-based database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object. Triple stores represent information as triples and retrieve it via a query language; yet, there are some key differences from relational databases, mainly that a triple store is optimized for triples.



Subject	Predicate	Object
Alice	knows	Mike
Alice	Likes	Photo1
Mike	Likes	Photo1
Bob	Likes	Photo1
Bob	Knows	John
Bob	Interested In	Topic
John	Interested In	Topic
John	Likes	Photo2

Fig. 2. RDF triple store and its representation as a graph.

Triples in triple store are illustrated in Fig. 2. There has been some progress in research made towards clustered RDF database systems. Clustered RDF databases that are currently available, such as SHARD [5], YARS2 [36], Jena and Jena Elephas [37] and Virtuoso [38], generally hash partition triples across multiple computing nodes and parallelize access to these nodes at query time.

2.2 Community Detection and Multi-Objective Evolutionary Algorithms

In graph theory, clusters are often defined algorithmically when certain measures of density and sparseness are optimized by an algorithm, the result of which is the partitioning of a network into communities [39]. In many cases, finding optimality of these measures is NP-hard. Usually, approximate but faster algorithms are used for tackling

NP-hard problems. One of the practical meta-heuristic algorithms for approximate solving of NP-hard tasks is evolutionary algorithms (EA) described in [40]. The application of GA (genetic algorithms) to community detection was described in [41], [42], [43], [44], [45], and [46], and the application of EA was described in [47] and [48]. Chang et al. [49] described an application of ant colony optimization to community detection.

Modularity [50] is one of the most popularly used metrics for concluding the quality of non-overlapping graph clustering, particularly in the network analysis community [42], [43], [44], [45]. The problem of discovering a clustering with maximal modularity is NP-Complete [51]. As a result, many polynomial time heuristic algorithms have been developed [52], [53] [54] [55].

One popular community detection algorithm is the Girvan-Newman algorithm [56], where edges having maximal betweenness centrality are consecutively removed from the network until no edges remain. Modularity may be defined as in equation (1) [57], where n_c is a total number of communities, m is the number of edges in the graph, l_i is the total number of edges within community i, d_i is the sum of degrees of all nodes in i.

$$Q = \sum_{i=1}^{n_c} \left[\frac{l_i}{m} - \left(\frac{d_i}{2m} \right)^2 \right] \quad (1)$$

As per Yi et al. [58], "In Genetic Algorithms (GA), a population of chromosomes, which encode candidate solutions/individuals to an optimization problem, evolves toward better solutions. After the solution is genetically represented in the chromosome format and the fitness functions are defined, GA proceeds to initialize a population of solutions randomly/deterministically. Then, GA aims to improve it through repetitive applications of several genetic operators such as selection, crossover, and mutation. Finally, local search and boundary search operators are applied to fine tune the results."

Li and Song [46] described an extended compact genetic algorithm. We chose our previous works using GA of [12], [59] and [60] for various reasons, mainly: (1) due to the huge search space, (2) NP-completeness of maximizing modularity [51], (3) being able to scale on large graphs, as well as (4) being able to reflect dynamic changes coming from different data sources. However, some changes on GA from [12] were necessary to adapt to our proposed data-aware HDFS framework. In the architecture section, we describe our distributed genetic algorithm where we developed, using open sources for multi-objective optimization and genetic algorithms, Jmetal [61] and Apache Jena Elephas [37] to store and manage RDF data. Later, we validate it on well-known datasets and experiment with it on the big RDF graph generated by the Lehigh University Benchmark (LUBM).

2.3 Service deployments over HDFS

As mentioned before, HDFS serves as a distributed data source for modern big-data solutions, such as Apache Spark [6], Mesos [7], HAMR [8] and hundreds of others. Such solutions have many deployments, mostly over HDFS or over a service that runs on HDFS (see Fig. 3).

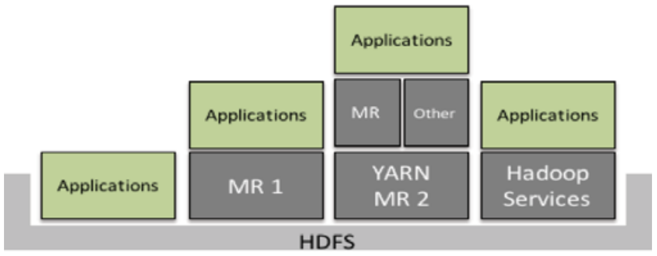


Fig. 3. Services and applications deployments on HDFS.

3 HDFS PERFORMANCE & EFFICIENCY PROBLEM

The utilization of the Hadoop eco-system to process enterprise data and build applications on top of it is dependent upon the enterprise use-cases and the data. Since IT BI teams (business intelligence) in businesses and enterprises configure such systems to meet their goals and roadmaps, they focus on the data and use-cases.

Most enterprise data are collected for specific use cases. Later, these data reside on storages waiting for the BI team to make use of them, thus resulting in data collected from multiple sources having multiple structures.

As per Huang et al. [1] and Rohloff et al. [5], the implementation of Hadoop and the services that are designed to run on HDFS lack optimization for graphs. Some of the causes for HDFS inefficiency include the following as per [1]: (1) the default hash partitioning provided by Hadoop may lead related data to end up far away physically over the set of computing resources, effectually resulting in a massive amount of data transfer between resources to finish graph operations. Thus, combining related data is a win as per [1]; (2) Hadoop considers the same importance for all data blocks and partitions, so maintaining the locality of inter-cluster neighbors and keeping them physically close-by improves efficiency, and (3) HDFS is not optimized for graph data.

Huang et al. [1] showed an efficiency problem with the said technique in Rohloff et al. [5] within a Hadoop-based system. However, the manner in which Huang et al. [1] and Rohloff et al. [5] worked around the problem can be generalized. Since they focus on one particular file type and one simple clustering algorithm, we believe that this technique has some drawbacks when we deal with big and dynamic un/semi/multi-structured data. In a previous study, Hajeer et al. [12], we confirmed such limitations. The study showed how to use genetic algorithms to cluster such data. We used this technique in Hajeer et al. [12], Pizzuti [43] and [42] along with a list of other work, such as [48], [46], [62], and [63], after building a transformation method to convert desired data into graph data. The results of extending the work in [12] were used to generalize Huang et al. [1] and Rohloff et al. [5].

In Hadoop, the main idea is to bring the computations to the data; for example, MapReduce, the Map part, can quickly bring the computations to the container that has part of the data as its resource. On the other hand, the reduce phase collects data from the mappers' outputs, and in most cases, these data parts have to travel over the network to the right containers that run some specific reducers. Such mappers' output usually gets collected from multiple mappers to each reducer. Programmers have to control such problems to some extent. That is why even at Cloudera's training for developers they tend to give a guideline for programmers to use mapper side joins rather than reducers. Cloudera's training also encourages developers to use Hive [64] query planner when possible to take care of such joins. Hence, we strongly believe that the Hadoop cluster itself should have a sense of data awareness, where data from mappers' outputs for the same key should be in the same, or at least a nearby, machine as much as possible.

Since applications and jobs are different from one use-case to another, it is impossible to cover all cases and future cases. One solution to optimize such jobs is to cluster the data as the proposed framework does. Not all algorithms are covered in such cases, but most graph algorithms specifically rely on the data that are related and connected (e.g. graph traversing).

4 Data-aware HDFS Indices

4.1 Graph Transformation

We discussed in the introduction the sources and problems of modern data. And we mentioned that data could come from different sources $S = (S_1, S_2, ..., S_Z)$ where S_N is source Z and $Z \geq 1$. These various sources generate or contain data with different structures, sometimes for the same entities but different data and different structures. $D=(DS_1,DS_2...,DS_Z)$, where DS_Z is the structure of data coming from source Z and contained to the infinite superset DS (Structure) $DS_Z \subset D$, $|DS|=\infty$.

Data with the structure $\in D$ were transformed into a graph G (V, E) as an undirected graph and with the number of vertices $|V|=m$ and the number of edges $|E|=n$. This transformation is further explained in the Overall Architecture section.

4.2 Graph Clustering

We referred to a graph of vertices V and edges E as G (V, E), as a directed graph. Also, the number of vertices $|V|=m$, number of edges $|E|=n$ and the clustering $C= (C_1, C_2, C_3..., CJ)$ as a partition of V as disjoint sets. We call C a clustering of G containing J clusters. The number of clusters j has a minimum of $j=1$ when C contains only one subset $C_1 = V$, and a maximum of $j=m$ when every cluster C_j contains only one vertex. We identify the cluster C_i as a subgraph of G. The graph $G[C_j]:=(C_j,E(C_j))$, where $E(C_j)=\{\{VW\} = \{E:VW \in :C_j\}$. Then $E(C) = \bigcup_{J=1}^{j} E(C_j)$ is the set of intra-cluster edges and $E/E(C)$ is the set of inter-cluster edges. The number of intra-cluster edges denoted by $m(C)$ and $\overline{m}(C)$ is the number of inter-cluster edges.

In our clustering algorithm, we used modularity as a fitness measure in Hajeer et al. [12]. Modularity Q is then defined as the fraction of edges that fall within group 1 or 2, minus the expected number of edges within groups 1 and 2 for a random graph with the same node degree distribution as the given network. Hence, the actual number of edges between v and w minus expected number of edges between them is $A_{vw}-(k_v\ k_w)/2m$. Modularity can be expressed in Equation (2) [57]:

$$Q = \frac{1}{2m}\sum_{vw}[A_{vw} - \frac{K_v * K_w}{2m}]\frac{S_v S_w + 1}{2} \tag{2}$$

Notice that Equation (2) partitions the network only for two groups. To identify multiple communities in a graph; the formula has to be generalized as (3) [57]:

$$Q = \sum_{vw}\left[\frac{A_{vw}}{2m} - \frac{K_v * K_w}{(2m)(2m)}\right]\delta(C_v, C_w) = \sum_{i=1}^{C}(e_{ij} - a_i^2) \tag{3}$$

Where e_{ij} is the fraction of edges with one-end vertices in community i and the other in community j as (4) [57]:

$$e_{ij} = \sum_{vw}\frac{A_{vw}}{2m}1_{v\in c_i}1_{w\in c_j} \tag{4}$$

And a_i is the fraction of ends of edges that are attached to vertices in community i as in Equation (5) [57]:

$$a_i = \frac{k_i}{2m} = \sum_i e_{ij} \tag{5}$$

4.3 Graph Distribution and Assumptions

As described previously, three major challenges faced HDFS optimization; two of them were about how Hadoop hashes and distributes the data. Our assumption and experiments showed that (1) storing intra-cluster data together on the same machine and (2) storing close inter-clusters data on close-by machines were a huge step toward optimizing HDFS. Let $M= (M_1,M_2,....,M_I)$ a set of machines that represent Hadoop computing resources, where $I\in [0, \infty)$ belongs to the finite natural numbers set. And M_i, M_{i+1}, M_{i+n} are machines where physical network distance between M_i and M_{i+1} are closer than M_i and M_{i+n}. The cluster C_j with all its vertices and edges should be placed in the same graph partition or at least on the same machine M_i. When there is no place left on M_i it should be placed at least to M_{i+1} and so on; the closer the machine, the better the results. For Cj and C_{j+1} where there are more inter-cluster edges than C_j and C_{j+n} then C_j and C_{j+1} should be placed into the smallest possible m, M_i and M_{i+m} (closest machine physically) where $0 \leq m \leq I$.

5 Overall Architecture

Our clustering framework (DEGA-Gen) is a part of the proposed data-awareness module running on top of the distributed data storage as shown in Fig. 4.

The framework interacts with HDFS and its available services to provide updated clusters as data flows in HDFS. Our goal is to achieve optimization by placing related data together and reducing overhead on data movement between hosts. Data transfer mostly happens in aggregation processes or joins.

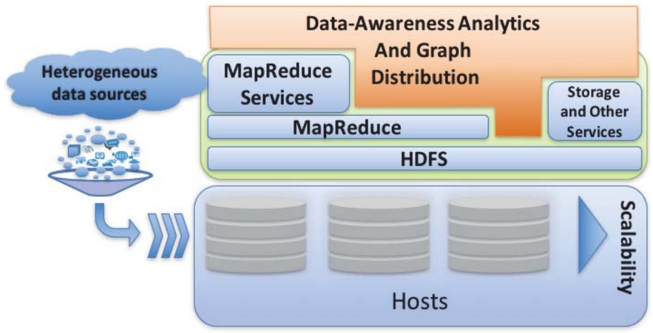

Data-Awareness Analytics And Graph Distribution
Heterogeneous data sources
MapReduce Services
Storage and Other Services
MapReduce
HDFS
Scalability
Hosts

Fig. 4. Data awareness module and Distributed evolutionary clustering algorithm as part of Hadoop.

5.1 Building Distributed RDF Graphs

The first step performed by the data-aware HDFS framework is converting datasets into a distributed RDF graph. This process was done using the open sources Apache Jena and Apache Jena Elephas. The proposed data-aware HDFS turns the datasets into quadruples rather than triples; reasons for this process are explained in the Genetic-Based Clustering. Unlike the widespread use of quadruple representation, we used the extra field in the quads to represent cluster affiliation of the triple rather than the graph membership of the quad; we called the field (Chromosome ID). This process leverages the usage of quad stores to enhance the sub-process (encoding and representation for distributed clustering).

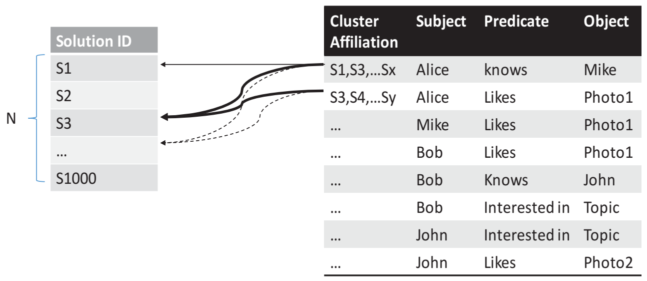
Not all types of data can be transformed into graphs. However, Big Data is about the use-case and the data in some cases. This transformation is direct (SHARD), and relational database records (for example) can be represented as an RDF graph where each attribute of each record is a relation between two nodes, one node represents the value of the key field and the other node is the value of the attribute field. In other cases, (like text data), relations between data points can be defined based on the use case. Take natural language processing, for example, one can define relations of (word comes before) and (word comes after) for each word to build a prediction model.

5.2 Genetic-Based Clustering

5.2.1 Encoding & Representation

We used the encoding from [65] to overcome the big encoding issues found in previous studies and listed in [65]. Such encoding derives from the definition of clusters.

However, even with such encoding in [65], solutions can still have a very long representation as the data scales up. Eventually, the GA client will run out of memory handling solutions itself as the data scales up. Another technique we used to reduce the overhead of manipulating solutions was to store it as extra information along with graph triples on HDFS, by converting data points from <Node> <Predicate><Node> triples, as in Rohloff et al. [5], into <Chromosome_part><Node><predicate> <Node> quadruples. We referred to <chromosome_part> as a list of solution_IDs that this particular node belongs to in the population. This encoding leads to a population of a fixed size list of Integers on the GA client side called solution_IDs. This technique allows the client to scale the clustering GA on larger size datasets that the HDFS can hold.



Cluster Affiliation	Subject	Predicate	Object
S1,S3,...Sx	Alice	knows	Mike
S3,S4,...Sy	Alice	Likes	Photo1
...	Mike	Likes	Photo1
...	Bob	Likes	Photo1
...	Bob	Knows	John
...	Bob	Interested in	Topic
...	John	Interested in	Topic
...	John	Likes	Photo2

Fig. 5. RDF-triples to RDF-quadruples (solution encoding).

The idea was to treat solutions as data and to inherit all scalability properties that apply to the graph. Thus, the population of a size X on the client side has a constant size(X) regardless of the data size. We referred to this novel technique as Distributed chromosomes, and as a concept, it is about the distribution of genes from the solutions along with the data. Fig. 5 explains how the graph data were stored in RDF format and how we performed the integration of solution encoding on RDF data.

We used Apache Jena and Jena Alephas and modified these open sources to match our needs. Convert_to_quads_Chromo class was developed to convert RDF graph Triples to Quads as in Fig. 5. This class contained Mapper, reducer, combiner, and appropriate writables as well as input and output classes formatted to deal with RDF data. It takes each triple from each block of data and converts it into a quad with a random gene (part of solutions) that it belongs to then stores it back into HDFS.

5.2.2 Objective Functions

In our clustering algorithm, we maximize modularity as an objective in Hajeer et al. [65]. As per [57], Modularity Q is defined then as the fraction of edges that fall within group 1 or 2, minus the expected number of edges within groups 1 and 2 for a random graph with the same node degree distribution as the given network. Hence, the actual number of edges between v and w minus expected number of edges between them is A_{vw}-$(k_v\ k_w)/2m$. Please refer to Equations 1-5 in the Data-aware HDFS Indices section. Note that modularity maximization is not the only objective. Another objective is to minimize the solution length. Considering intra-cluster edges as inter-cluster edges results in some longer solutions with no difference in modularity. Hence, those solutions need to be given a smaller fitness but not totally ignored (a combination with other solutions may lead to a better clustering).

Since our evaluation on the datasets used considers predicates and relations to work both ways (an undirected graph), we used modularity in Equation (3). For use cases where the defined relationships result in a directed graph, there is an extended modularity that was proposed for a directed graph that can be utilized. On the other hand, the clustering purpose is to find a better placement for graph data. Hence, considering directed graphs as undirected graphs while clustering does not necessarily force the user to have the same assumption while querying the data.

5.2.3 Details Steps of Genetic Clustering

Population Initialization

Population initialization is the process of creating a collection of diverse solutions. As described in the Encoding & Representation sub-section, we transform the triples in RDF data into quads, adding the ability to hold a gene (part of the solution) for each vertex, where this gene is a list of random solution IDs to which a particular vertex belongs. For example, if a Quadruple *D* has *S1* and *S5* as genes, then that is translated as solutions *S1* and *S5;* both consider the edge *D* as an inter-cluster edge connecting two separate clusters.

In $\forall T_i \in T$ there is a set of solutions $\cup S_n \subseteq S$, where *T* is $\cup$ of triples *t* that constructs the graph *G*, and *S* is the set of solutions *s* in the population. It is critical to keep in mind that the maximum size of such a list is the integer size of the population. The initialization process for populations is shown in Fig. 5. The GA client only holds a two-dimensional array of solution-IDs and Modularity Fitness (Integers and floats), allowing the client to start the selection process and initiate the distributed GA operators. Working on a fixed, small size two-dimensional array, where the real genes are stored in the data blocks in a distributed manner taking advantage of HDFS, proved to provide more scalability.

Solutions Evaluation

The evaluation was done using the objective functions described in the objectives section. Each solution is evaluated by computing modularity on the analogous graph, a graph where edges in the solution are marked as inter-cluster edges. We identified the clusters by removing the marked edges and considering the disconnected graph components as communities. Then, we computed the modularity considering the marked edges again as inter-cluster edges.

The process of computing the modularity on a large graph is both resource and time consuming, so we decided to improve it using distributed tasks to be run on the quadruple store created with extra data for solutions. Using HDFS and distributing the dataset over multiple machines, we were able to batch process each set of solutions (generation) at once.

Given a Population_ID list S that contain ID's of solutions to be evaluated

```
Map (Key index, Value Quad):
  ForEach solutionID in S:
      If Quad.GetGenes in solution:
          Quad.marked = True
          Emit (solutionID , Quad)
      Else:
          Quad.marked = False
```

Fig. 6. Distributed evaluation of solutions (map task).

After the client side of the algorithm injects current population solutions data into the quadruples stored in HDFS, it sends the list of solution IDs (list of integers) to be evaluated. Fig. 6 illustrates the evaluation Map tasks.

The map function is called for each Quad in the graph chunk that represents part of the graph. Jena Elephas is used with modified input and output class to use Chromosome quads rather than default graph quads. Each container on the HDFS cluster performs a map operation on the graph chunks it has been assigned. After mapping all the chunks into pairs of <Keys, Values> representing solution IDs and Quads that are part of the corresponding solution, the shuffling task takes place. All values for the same key are grouped together as <Key, List of Values> that represent each solution and the list of marked and unmarked Quads (Graphs where inter-cluster edges are marked). The final stage consists of the reduce tasks that are described in Fig. 7.

Given a solution S and a set of Quads marked based on S, as mappers' outputs and reducers' input for a graph G with N Quads

```
Reduce (Solution S, EdgesQuads [E1,E2,E3,….EN]):
  ForEach Quad E in QuadssList:
      If E.marked = True:
          MarkedQuads.append(E)
      Else:
          UnMarkedQuads.append(E)
  Endfor
  Communities = FindComponents(MarkedQuads, UnMarkedQuads)
  Modularity = 0
  ForEach Community C in Communities:
      DegreeFraction = $\frac{(C.InnerEdges * 2 + C.OutterEdges)}{(2*N)}$
      Modularity += (C. InnerEdges /N)-(DegreeFraction)^2
  Endfor
  Emit (S, Modularity)
```

Fig. 7. Distributed evaluation of solutions (reduce task).

The FindComponents function was implemented using a modified linear finding component algorithm to store also the number of intra-cluster edges and the number of inter-cluster edges for each community. When reduce tasks finish, the results of reducers are written to HDFS, and the results contain each solution with its modularity. The results consist of a fixed size two-dimensional array of integer solution IDs and a fitness for each solution. The evolutionary algorithm reads this file and continues working on an evaluated generation ready for selection, crossover and mutation processes. In the last generation, an extra piece of information controlled by a boolean configuration variable is written to HDFS as well; this piece contains the clustering affiliation for each node. The reason they are only written in the last generation is to lower the write overhead on HDFS while affiliations are not needed any time before it.

Crossover, Mutation and Selection

Since we stored the chromosomes in a distributed manner, we needed to modify the GA operators used in Jmetal open source to be able to run them on the corresponding quadruples that represent the graph. This procedure was done by developing distributed crossover and distributed mutation modules, which in return created jobs of crossover and mutations to be performed on the corresponding population.

After evaluating the population, the selection process starts based on each solution ID and its fitness. Tournament selection is the selection used, and the reason is to avoid converging to locally optimal solutions, which are a lot based on our encoding technique. By ranking the population and choosing solutions from each class, a set of parents along with the new offspring IDs were constructed. Fig. 8 is a high-level diagram showing the steps in which the algorithm creates GA operator's tasks.

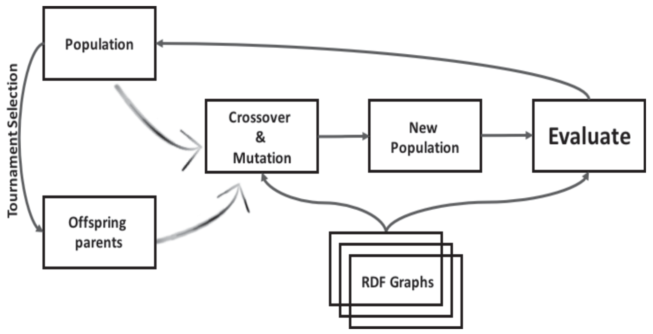
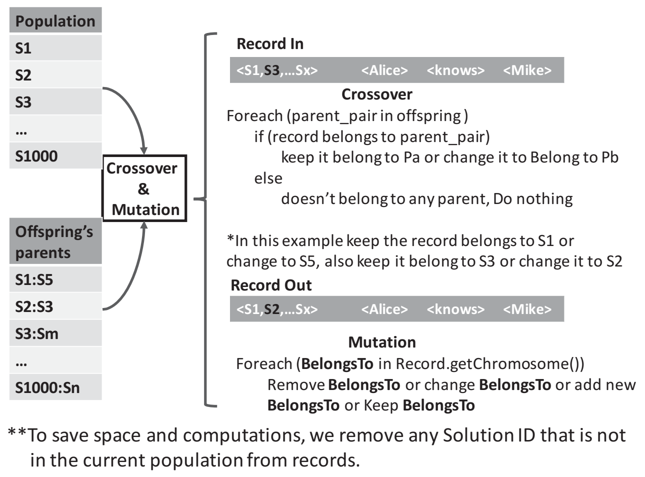


Fig. 8. Distributed genetic algorithm for RDF clustering.

Encoding and storing solutions in a distributed manner delivered the advantage of small and fixed size populations on a client side. However, GA operators in the open source Jmetal needed to be modified as well as NSGAII, which was used in our case. The original NSGAII creates a population's and offspring's solutions and evaluates it one solution at a time; such a situation creates an overhead of tasks on HDFS. Rather, we modified NSGAII to DNSGAII (Distributed NSGAII) by creating a set of solutions then performing evaluation and GA operators at once in one MR2 task. Fig. 9 illustrates the task of distributed crossover and distributed mutation.



```
Crossover
Foreach (parent_pair in offspring )
    if (record belongs to parent_pair)
        keep it belong to Pa or change it to Belong to Pb
    else
        doesn't belong to any parent, Do nothing
```

*In this example keep the record belongs to S1 or change to S5, also keep it belong to S3 or change it to S2

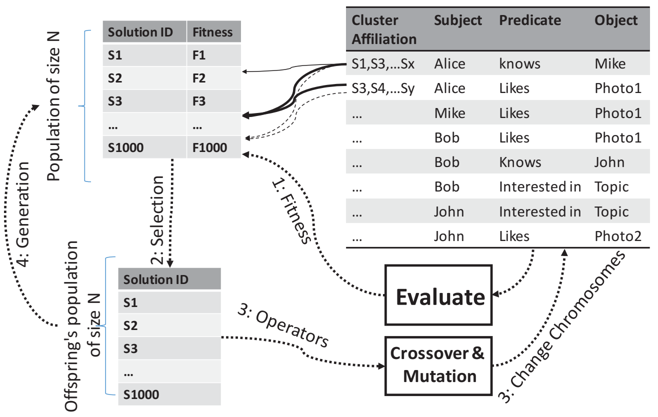
```
Mutation
Foreach (BelongsTo in Record.getChromosome())
    Remove BelongsTo or change BelongsTo or add new
    BelongsTo or Keep BelongsTo
```

**To save space and computations, we remove any Solution ID that is not in the current population from records.

Fig. 9. Distributed crossover & mutation in genetic search.

The distributed crossover and mutation task takes the population as inputs along with the selection results, then for each quad in the data changes the partial chromosomes accordingly. The task removes any solution ID (gene) that does not belong to the current population to save space and computations. Then, as shown in Fig. 8, the new offspring population is sent to evaluation. Here we have to note that solutions that belong to a previous generation will not be evaluated since they already have fitnesses. This copy technique of fitnesses saved an enormous amount of computations when we dealt with big data for a long series of generations.

The processes of representation, population initialization, evaluation, selection and offspring evaluation to population are illustrated in Fig. 10. The numbers represent the processes and tasks order. Since we were dealing with dynamic data as one of the Big-Data five V limitations (Velocity, Variety, Veracity, Value and Volume), the algorithm gets suspended when it converges to the same solution for a sequence of generations then continues working as new data arrives to start from the last generation reached.



Solution ID	Fitness
S1	F1
S2	F2
S3	F3
...	...
S1000	F1000

Cluster Affiliation	Subject	Predicate	Object
S1,S3,...Sx	Alice	knows	Mike
S3,S4,...Sy	Alice	Likes	Photo1
...	Mike	Likes	Photo1
...	Bob	Likes	Photo1
...	Bob	Knows	John
...	Bob	Interested in	Topic
...	John	Interested in	Topic
...	John	Likes	Photo2

Solution ID
S1
S2
S3
...
S1000

Fig. 10. Distributed genetic algorithm clustering process flow.

Suspension of the algorithm ensures that clustering will apply to the new data while old data have the best affiliation found, so there is no need for the clustering process to start from the beginning.

5.3 Partitioning and Placement

After clustering the RDF data, the last step was to repartition the data and place graph quadruples accordingly. The goal in this step was to place quadruples that belong to the same cluster and have a high degree of connectivity into the same partition to ensure locality of intra-cluster quadruples. Another goal was to place highly connected inter-clusters into a close partition physically, to map the inter-cluster distance onto the physical distance of partitions. Fig. 11 illustrates the desired allocation of quadruples, assuming that the horizontal distance in the figure represents the physical distance between the computing nodes (the distance of network routing).

We account for the distance of HDFS nodes by how many routing hops between them (networks, routers, switches…). We set up HDFS over machines connected using multiple networks to create a distance in routing. The placement script placed quadruples as in Fig. 11.

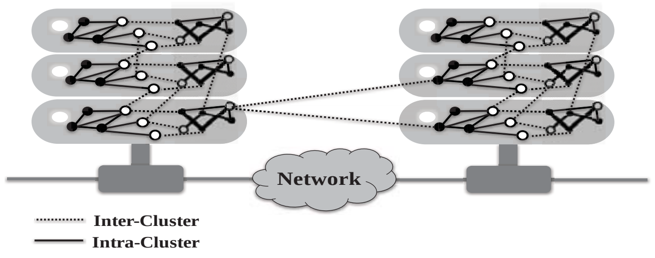


Fig. 11. Quadruples placement in different HDFS nodes.

Partitions are created based on the number of machines; each machine has its own partition. A MapReduce job scans the quadruples and places quadruples related to one random cluster in one partition then emits the placed quadruples, leading to where all connected clusters' IDs are stored for the next scan. The second scan places the quadruples for the closest inter-clusters in the same partition and emits from the original dataset. Further, the second closest inter-clusters are placed into the next closest partition. When there are no more interconnected clusters left, another random cluster is chosen from the dataset until no more data is available. Fig. 11, for example, illustrates how clusters with three inter-cluster connectivity are placed in the same HDFS node, and with two inter-cluster connectivity are placed in the next HDFS node, further clusters are placed in further nodes.

6 Experiments and Results

We divided the experiment into two sections: the first section is the design and the testing of the clustering algorithm on the graph store to test the clustering results, and the second section is about the tests and comparisons of the effect of optimization framework on HDFS. All graphs and trend models are processed using Tableau [66].

6.1 Graph Conversion and Clustering

We validated the correctness of our clustering algorithm and made sure it produced valid and comparable results. We chose some well-known small datasets carefully and made sure they were the same datasets used in previous studies for comparison. These sets are:

Zachary Karate Club: Graph contains 34 vertices and 78 edges. Nodes represent members of university karate club, and connections between them represent communication patterns. It was collected in 1997 [67].

Bottlenose Dolphins: Network made of 62 dolphins and their interactions, collected over a period of 7 years, 1994.

US Political books: Network of 105 nodes and 441 edges. Network represents books about US politics, frequently bought together.

American College football: Network of 115 nodes representing the teams, and 613 edges connecting them.

TABLE 1 shows the results of the algorithm validation and compares it to some of the popular algorithms. Our algorithm achieved a maximized modularity in some cases and close modularity in the rest. Some algorithms were omitted because of a very high modularity; such results are impossible for hard clustering as per Daniel Aloise, Sonia Cafieri et al. [68] since they found and proved the optimal modularity for each one of these datasets.

TABLE 1 Modularity Maximization Comparison

Dataset	GN	CNM	L Max	GATHB	MOG A-Net	**Our Method**
Karate	0.4	0.380	0.419	0.4	0.416	**0.416**
Dolphins	0.52	0.495	0.523	0.52	0.505	**0.528**
Football	0.6	0.577	0.61	0.55	0.515	**0.539**
Books	0.51	0.502	0.526	0.52	0.518	**0.523**

The results in TABLE 1 and [68] prove that our approach provides results that converge to optimal solutions, and the quality of the results (compared to other popular algorithms) are better in most cases. Some cases showed a slightly lower modularity.

We found that selection plays a role in how the solutions converge. Furthermore, for certain datasets, binary selection converges to higher modularity in a smaller amount of generations, whereas, random selection can provide a higher rate of jumps from local optimal fitnesses.

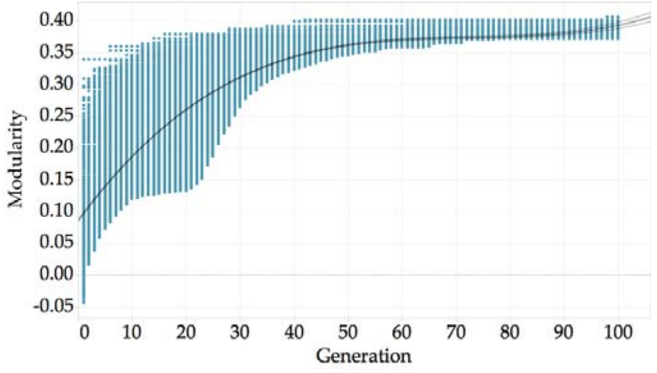


Fig. 12. Population fitness vs. generation scatter plot (karate club network).

The evaluations after each population were reported to analyze the convergence of solutions over generations. We generated graphs and computed trend models by dumping the population array and using the scatter plot to create a visual representation of the outcomes for each generation. We found a correlation between the distribution of modularities and the number of generations to extract such modularities. Fig. 12 shows the distribution of modularity vs. Generation.

To scale our approach to Big Data we used LUBM to generate RDF graph data and deploy on a cluster with the following properties, as in TABLE 2. The configurations we used yielded 87 containers. Each container has access to all 48 disks, two CPU cores and 4 GB of memory.

We used six computing nodes compared to 10 and 20 nodes in similar studies. We looked at the number of nodes only to validate the effect of network-communications and distance in the network. However, in the presence of YARN and the notion of containers, it makes more sense to compare resources than the number of nodes. Our computing cluster and our configuration yield to 86 containers, each with 4 GB of memory and 2 Threads (one Core) of CPU (Max of 2.93 GHz and Min of 2.00 GHz), a total of 344 GB of memory and 172 CPU Threads (86 Cores), compared to a total of 80 GB of memory and 40 CPU cores and 40 Disks divided equally between 20 computing nodes per Huang et al. [1].

TABLE 2 HADOOP CLUSTER AND CONFIGURATIONS

Machine		Threads	Memory	Disks
Master	Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz	72	64	10
Node1	Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz	56	64	10
Node2	Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz	40	64	10
Node3	Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz	40	64	10
Node4	Intel(R) Xeon(R) CPU X5570 @ 2.93GHz	16	96	2
Node5	Intel(R) Xeon(R) CPU E5520 @ 2.27GHz	16	48	6

The Lehigh University Benchmark (LUBM) is a university domain ontology for synthetic OWL and RDF data scalable to an arbitrary size and fourteen queries representing a variation of properties. LUBM is the most widely used benchmark of the Semantic Web community.

We generated multiple datasets with different sizes to compare the behavior of our algorithm. We removed predicates of value “type” or similar before clustering, which is a previously adopted way to increase the quality of clusters and simplify the complexity of the graph [1]. We analyzed the execution time of initializing a population of solutions; the population size is 1000 solutions per generation. TABLE 3 presents the execution time for graph conversion into quadruples and the initialization of the first population. The penalty of having a large amount of time to prepare the data is a tradeoff with the amount of processing and querying that is done on the data and can be further optimized with new technologies [22]. It is important to note that our framework, best used when the data will be processed heavily or queried continuously in the future, is a good idea to prepare the system for fast response and reduced hardware overhead.

TABLE 3 POPULATION INITIALIZATION & ALGORITHM RUN TIME (LUBM DATASETS)

Number of triples	Initialize Population (S)	Algorithm Run Time (Minutes)
8,970,048	13.556	21.7
20,637,270	19.621	31.6
30,285,222	28.611	47
221,140,408	207.314	261(~4.3 hours)

We further analyzed LUBM data. Although there is an enormous amount of triples, initialization of the first population with random inter-cluster edge affiliations does not take a long period of time. Fig. 13 shows the population convergence over time to a maximized modularity for a dataset of 30M quads.

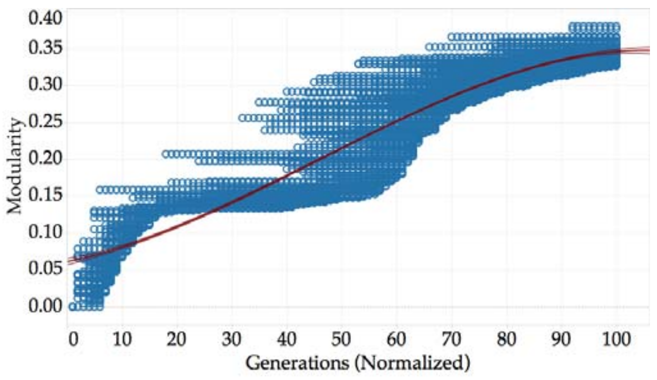


Fig. 13. Convergence to maximum modularity (LUBM 30 million Triples).

The trend model for LUBM 30 million triples dataset is illustrated in Fig. 14.

Trend Lines Model

A polynomial trend model of degree 3 is computed for Modularity given Generation. The model may be significant at p <= 0.05.

Equation:	Modularity = -2.13096e-07*Generation^3 + 2.51096e-05*Generation^2 + 0.00253695*Generation + 0.0567858
Model degrees of freedom:	4
R-Squared:	0.820795
p-value (significance):	< 0.0001

Fig. 14. Trend model description (30 million Triples).

The encoding technique we introduced created a solution space with many locally optimal solutions. Hence, using a mutation rate of 100 percent and multiple-point crossovers prevented falling in locally optimal solutions. Even though the mutation rate was set to 100 percent, considering intra-cluster edges as inter-cluster edges results in no difference in modularity for a given solution. On the other hand, due to the proposed encoding, the chances that the solution is affected by the mutation is less than a 100% (~72% of the solutions were affected on the data used). The encoding proposed is less influenced by mutation than traditional encoding. Hence a higher mutation rate is necessary to affect solutions.

Fig. 15 describes the modularities and its count in all generations (each bin is a range from the bin x-value until the next bin x-value). Since most of the intra-cluster edges do not affect the number of communities produced, having such edges in solutions do not affect the total fitness and explains these high modularity counts for non-maximal modularity. Yet, having these quadruples in the solution did not affect the algorithm’s ability to jump out of such cases.

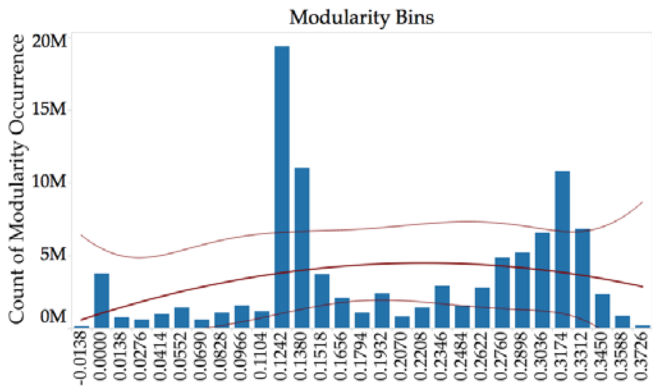


Fig. 15. Count of modularity for all generations as modularity bins (LUBM 30 million triples).

On a larger scale, we used 221M Quads initialized by converting LUBM triples. Fig. 16 shows the distribution and the correlation between the fitness measure and its frequency across all generations.

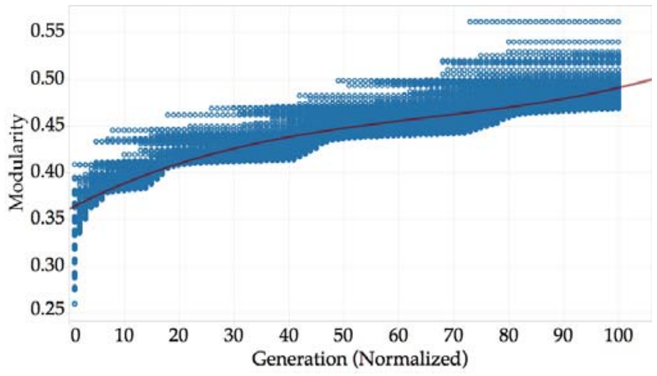


Fig. 16. Convergence to maximum modularity over generations (LUBM 221 million triples).

Fig. 17 shows the minimum, maximum, average and Median for each generation modularity fitness. A high mutation rate kept a space for some diverse generations.

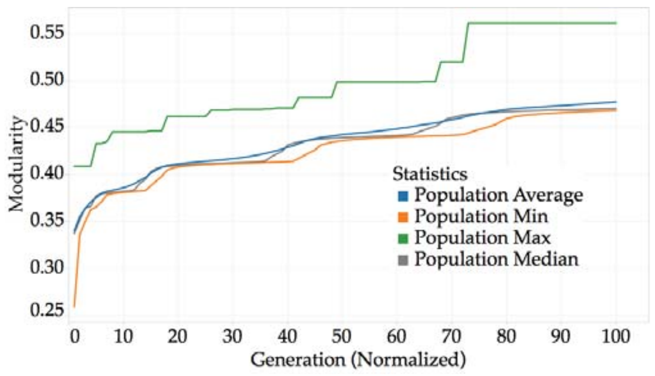


Fig. 17. Convergence to maximum modularity over generations (LUBM 221 million triples).

6.2 System Performance Experiments

In this section, we measure the performance of our system against multiple RDF stores including SHARD after clustering and placement. We used Cloudera Impala to create a table on the data. We focused mainly on time and resource usage. The system configurations used are the same as in TABLE 2. We refer to our framework as cluster partitioning.

In our experiments, we generate a dataset using LUBM. The generated dataset's size was from 37 to 142 GB in N-Triples format and contained from 200 million to 600 million triples (see TABLE 5). TABLE 6 shows each query's complexity as the number of joins in each query in regards to the benchmark.

6.2.1 Clustering and Load Time

The results of loading 270 million RDF triples into a twenty-machine HDFS cluster as per Huang et al. [1], are shown in TABLE 4. For a ten-machine cluster, as per Alexsander et al. [24], the loading time was 40 minutes. Our results are shown in TABLE 5.

Because of the differences in resources, for benchmarking queries in the Query Performance Comparison section, we normalize results to be able to compare query response time. There is a noticeable workload in terms of time to prepare the Cluster-based partitioned RDF against hash-partitioning the data. However, the effect on optimization during query time and storage is a trade off, as we describe in the Query Performance Comparison section. It is critical to point out that the number of triples has a larger effect than the storage size in GB; since different compression can solve the storage size issue but not the amount of information needed to be processed. It is also very important to note that clustering results are stored by our proposed architecture, so when new data and changes become available, the algorithm does not need to start over. In other words, adding changes to data, to some extent, happens in close to real time by adding the new triples to the physical server that has the data-cluster it connects to.

TABLE 4 Loading time as per Huang et al. [1] for 270 Million triples

System	Load Time
RDF-3X	2.5 H
SHARD	6.5 H
Hash Partitioning	0.5 H
Graph Partitioning	4.2 H

TABLE 5 Cluster Partitioning Load Time

Number of Triples	Size (GB)	Load Time (~)
221,278,374	37.1	3.8 Min Conversion 4.3 H clustering and placement
427,016,108	82.7	6.5 Min Conversion 7.2 H clustering and placement
613,190,853	142.3	8.1 Min Conversion 9.4 H clustering and placement

6.2.2 Query Performance Comparison

To compare to other studies, we need first to understand the complexity of each query of the 14 queries. As shown in TABLE 6 [1], each query has a count of subject-to-subject joins (S-S) and subject-to-object joins (S-O).

Hash partitioning maps the same keys (subjects) to the same blocks. Thus, optimization issues are less likely to happen in subject-to-subject than subject-to-object joins. subject-to-object joins are the ones that cause most of the optimization issues in hash partitioning the data since it does not ensure the values of the same key (related data points) to be in the same data blocks.

TABLE 6 LUBM QUERY COMPLEXITY [1]

Query	S-S Joins	S-O Joins	Total
Q1	1	0	1
Q2	3	3	6
Q3	1	0	1
Q4	4	0	4
Q5	1	0	1
Q6	0	0	0
Q7	1	2	3
Q8	3	1	4
Q9	3	3	6
Q10	1	0	1
Q11	1	1	2
Q12	2	1	3
Q13	1	0	1
Q14	0	0	0

Huang et al. [1] showed that the queries could be characterized into two groups. Queries 1, 3, 4, 5, 7, 8, 10, 11 and 12 run fast on single-machine RDF stores like RDF-3X; queries 2, 6, 9, 13 and 14 run slower on single machine RDF stores. For fast queries, data size does not matter as much since it is reduced before scans are required [1]. Hence, paralyzing such stores on multiple machines does not have many advantages and will only add a network delay for queries to start [1]. On the other hand, the slow queries lack scalability on single machine RDF stores.

Unlike a data-aware HDFS framework, some previous efforts have been achieved for slow queries, for example hash-partitioning the data on SHARD. However, hashing led to subject-to-object joins' limitations due to the need of moving intermediate data over the network. Another example is Huang et al. where the objects connected to a subject were processed to fall into the same blocks for one or two hops between subject and object. However, space limitations due to an increase in data size were present; also, there is a limitation of applying such an algorithm to a highly-connected graph [1]. We used clustering in our framework rather than hops. So, triples do not need to be replicated as many do in different partitions, and no increase in the data size occurred (other than HDFS replication for reliability). Other works like Sempala [24], or using HIVE, PigSPARQL [25] & [26], MapMerge [27] and MAPSIN [28] overcame scalability but used different storage than triples.

Huang et al. [1] covered four different frameworks on LUBM 270 million triples, evident in Fig. 18.

As observed in Huang's et al. [1] results, changing the data in each sub-graph partition to include vertices that are two hops away from each subject in each triple increases efficiency in regard to execution time. However, it increases the data size dramatically as well as adding an extra step of processing duplicates after the query finishes. Huang et al. [1] explained it as a tradeoff between size and fast response time.

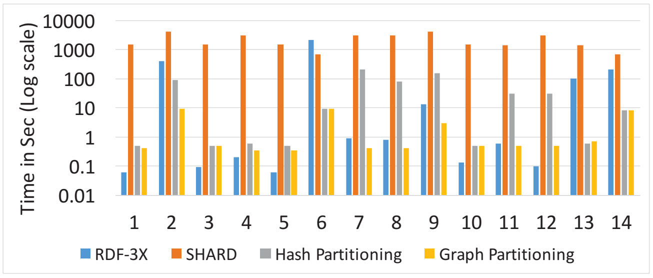


Fig. 18. Huang's query time (LUBM 270 million triples).

The two hops guarantee that the technique enabled the optimization of queries 2, 7, 8, 9, 11 and 12. On the other hand, it slowed down the rest of the queries compared to hash-partitioned data, since there was an extra required step to remove duplicates created by the replication algorithm they used. Hops-guarantee had no optimization to the rest of the queries. Queries 1, 3, 4, 5, 6, 10, 13 and 14 were equal or faster in the hash-partitioned graph than Hops-guarantee since they only had subject-to-subject joins and no duplicate removal steps were needed.

Comparing our cluster-based partitioning and SHARD allows one to directly see the benefit of our proposed graph partitioning technique on the naive triple placement on HDFS. To compare our results, we first took into consideration the difference in HDFS resources. It is very hard to match the two different Hadoop ecosystems since we do not have a clear idea about what are the exact configurations or what other services are running on those machines that might slow down the queries. However, one solution is to perform SHARD on our Hadoop eco-system on the same number of triples, then find the ratio of the difference. In other words, if Query Q1 on native HDFS took X time on cluster (A) and 2X on cluster (B) with the same native HDFS, then cluster (A) has 2 times speed up for Q1. So, optimizing Q1 to be Y speed up using a specific algorithm means Q1 time will be X/Y on cluster A and 2X/Y on cluster B for that specific algorithm. Note that optimization of a specific algorithm is measured in terms of Y times efficient than native HDFS regardless of the cluster used. Fig. 19 shows the comparative results (mean) for our data-aware HDFS framework (Cluster-Based partitioning) after running each query 20 times with an average variance of 0.06.

Results reported an increase of execution time efficiency for queries 2, 3, 6, 9, 13, and 14. Close efficiency occurred in queries 1 and 10. A less efficiency occurred in 4, 5, 7, 8, 11 and 12. The reason behind the improvement in subject-to-subject joins is the extra step of removing duplicates in graph-partitioning that Huang et al. used [1]. On the other hand, the efficiency increase related to subject-to-object joins was caused by the least amount of data scanned and moved by cluster-based partitioning, since cluster-based partitioning ensured the presence of most objects related to subjects in the same partition without the duplicates in another (less scans). Such results also reported less storage space than graph partitioning reported in [1].

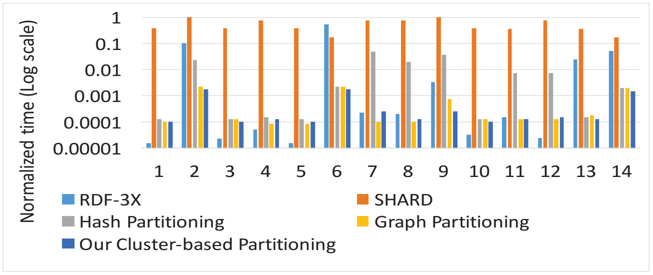


Fig. 19. Data-aware HDFS, query execution time (LUBM 270 million triples).

We further analyzed the network traffic on each HDFS computing node using Linux performance monitoring tools. SAR commands were used to monitor network adapters on each machine during each query. The results were aggregated from the report of each HDFS node to one file. Our framework effect on network traffic optimization for LUBM queries on a 221 million triple RDF graph is as follows: a great optimization factor on mostly S-O joins, a reduction by a factor of ~200X for Q7 hash partitioning of SHARD and cluster based triple placement. Furthermore, ~160X for Q8 and ~150X for Q9 optimization also were reported on network traffic.

Even though one of our goals is to deal with dynamic data, we looked at frameworks like Sempala, PigSPARQL, MapMerge and Sempala-Using-Hive, regardless of their limitations in dealing with dynamic updates. Fig. 20 illustrates results for 221 million RDF record.

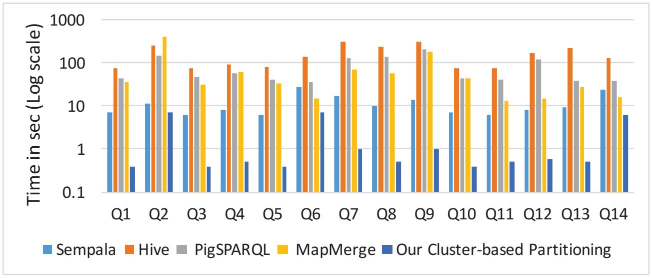


Fig. 20. Data-aware HDFS, query execution time (LUBM 221M triples).

7 Conclusion

In this article, we presented a data-aware HDFS and the services running on top of HDFS that optimize state-of-the-art RDF stores. We proposed a cluster-based data partitioning to manipulate the physical locality of the data to match the graph locality as well as the causality in HDFS processes. This allowed parallel processing of queries for data on HDFS that required less resource usage. Our framework was able to perform faster than some attempts and slightly slower than other attempts for scalable RDF data stores. However, with less resource usage. Studies in next-generation analytics and lambda architecture [15], [16], [17] and [18], along with Apache Kudu [20] and a set of studies in [21] proved to be fast and more efficient in processing of OLAP workloads and showed a strong performance in running time-critical workloads. It is worth the effort, however, to study the impact of intelligent data placement on such methods. For future work, we plan to further improve the distributed encoding and the genetic operators to reduce computation overhead. We also plan to experiment with dynamic updates for a larger velocity of data flow and to utilize tools and frameworks of the lambda architecture and next-generation analytics presented in the recent studies.

References

[1] J. Huang, D. J. Abadi and K. Ren, "Scalable SPARQL querying of large RDF graphs," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1123-113, 2011.

[2] K. Bajda-Pawlikowski, D. J. Abadi, A. Silberschatz and E. Paulson, "Efficient processing of data warehousing queries in a split execution environment," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.

[3] M. Walker, "Data Science Central," 19 Dec 2012. [Online]. Available: http://www.datasciencecentral.com/profiles/blogs/structured-vs-unstructured-data-the-rise-of-data-anarchy. [Accessed 16 Oct 2015].

[4] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, no. 2012, pp. 1--16.

[5] K. Rohloff and R. E. Schantz, "Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store," in *Proceedings of the fourth international workshop on Data-intensive distributed computing*, 2011.

[6] T. A. S. Foundation, "Apache Spark," The Apache Software Foundation, [Online]. Available: http://spark.apache.org. [Accessed Jan 2016].

[7] T. A. S. Foundation, "Apache Mesos," The Apache Software Foundation., [Online]. Available: http://mesos.apache.org. [Accessed 19 Jan 2016].

[8] E. I. Inc., "HAMR - Faster than the speed of data," ET International, Inc., [Online]. Available: http://www.hamrtech.com/index.html. [Accessed 19 Jan 2016].

[9] Apache Storm, "Apache STORM," Apache Software Foundation , [Online]. Available: http://storm.apache.org. [Accessed 16 9 2016].

[10] L. Aniello, R. Baldoni and L. Querzoni, "Adaptive online scheduling in storm," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013.

[11] P. Basanta-Val, N. Fernandez-Garcia, A. Wellings and N.

Audsley, "Improving the predictability of distributed stream processors," *Future Generation Computer Systems,* vol. 52, pp. 22--36, 2015.

[12] M. Hajeer, D. Dasgupta, A. Semenov and J. Veijalainen, "Distributed evolutionary approach to data clustering and modeling," in *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium*, 2014.

[13] H. Song, P. Basanta-Val, A. Steed, M. Jo and Z. Lv, "Next-generation big data analytics: State of the art, challenges, and future research topics," *IEEE Transactions on Industrial Informatics,* p. In Press, 2017.

[14] N. Agnihotri and A. K. Sharma, "Proposed algorithms for effective real time stream analysis in big data," in *Image Information Processing (ICIIP), 2015 Third International Conference on*, 2015.

[15] L. Aniello, R. Baldoni and L. Querzoni, "Adaptive online scheduling in storm," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013.

[16] P. Basanta-Val, N. Fernandez-Garcia, A. J. Wellings and N. C. Audsley, "Improving the predictability of distributed stream processors," *Future Generation Computer Systems,* vol. 52, pp. 22--36, 2015.

[17] P. Basanta-Val and M. Garcia-Valls, "A distributed real-time java-centric architecture for industrial systems," *IEEE Transactions on Industrial Informatics,* vol. 10, no. 1, pp. 27--34, 2014.

[18] P. Basanta-Val, N. C. Audsley, A. J. a. G. I. Wellings and N. Fernandez-Garcia, "Architecting Time-Critical Big-Data Systems," *IEEE Transactions on Big Data,* vol. 2, no. 4, pp. 310--324, 2016.

[19] M. Congosto, P. Basanta-Val and L. Sanchez-Fernandez, "T-Hoarder: A framework to process Twitter data streams," *Journal of Network and Computer Applications,* vol. 83, pp. 28--39, 2017.

[20] T. A. S. Foundation, "Introducing Apache Kudu," The Apache Software Foundation, 2017. [Online]. Available: https://kudu.apache.org/docs/. [Accessed 5 April 2017].

[21] N. Marz and J. Warren, Big Data: Principles and best practices of scalable realtime data systems, Manning Publications Co., 2015.

[22] M. Ferron-Jones, "It Peer Network," 16 May 2017. [Online]. Available: https://itpeernetwork.intel.com/new-breakthrough-persistent-memory-first-public-demo/. [Accessed 1 June 2017].

[23] E. Roy Marsten, "Is graph theory key to Understand Big Data," March 2014. [Online]. Available: http://www.wired.com/insights/2014/03/graph-theory-key-understanding-big-data/. [Accessed Oct 2015].

[24] A. Schatzle, M. Przyjaciel-Zablocki, A. Neu and G. Lausen, "Sempala: Interactive SPARQL query processing on hadoop," *The Semantic Web--ISWC 2014,* pp. 164--179, 2014.

[25] A. Schatzle, M. Przyjaciel-Zablocki, T. Hornung and G. Lausen, "PigSPARQL: a SPARQL query processing baseline for big data," in *Proceedings of the 2013th International Conference on Posters Demonstrations Track-Volume 1035*, 2013.

[26] A. Schatzle, M. Przyjaciel-Zablocki and G. Lausen, "PigSPARQL: Mapping SPARQL to Pig Latin," in *Proceedings of the International Workshop on Semantic Web Information Management*, 2011.

[27] M. Przyjaciel-Zablocki, A. Schatzle, E. Skaley, T. Hornung and G. Lausen, "Map-side merge joins for scalable SPARQL BGP processing," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, 2013.

[28] A. Schatzle, M. Przyjaciel-Zablocki, C. Dorner, T. Hornung and G. Lausen, "Cascading map-side joins over HBase for scalable join processing," *SSWS+ HPCSW,* p. 59, 2012.

[29] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR),* vol. 40, no. 1, p. 1, 2008.

[30] R. Angles, "A comparison of current graph database models," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, 2012.

[31] B. A. Eckman and P. G. Brown, "Graph data management for molecular and cell biology," *IBM journal of research and development,* vol. 50, no. 6, pp. 545-560, 2006.

[32] J. Hayes and C. Gutierrez, "Bipartite graphs as intermediate model for RDF," in *The Semantic Web--ISWC 2004*, Springer, 2004, pp. 47-61.

[33] A. Schenker, Graph-theoretic techniques for web content mining, World Scientific, 2005, p. 62.

[34] A. Nayak and I. Stojmenovic, Handbook of applied algorithms: Solving scientific, engineering, and practical problems, John Wiley & Sons, 2007.

[35] A. Silberschatz, H. F. Korth and S. Sudarshan, "Data models," *ACM Computing Surveys (CSUR),* vol. 28, no. 1, pp. 105-108, 1996.

[36] A. Harth, J. Umbrich, A. Hogan and S. Decker, "Yars2: A federated repository for querying graph structured data from the web," Springer, 2007, pp. 211--224.

[37] Apache, "Apache Jena Elephas," Apache, [Online]. Available: https://jena.apache.org/documentation/hadoop/. [Accessed 10 Feb 2016].

[38] O. Erling and I. Mikhailov, "Towards web scale RDF," *Proc. SSWS,* 2008.

[39] S. Fortunato, "Community detection in graphs," *Physics Reports,* vol. 486, no. 3, pp. 75-174, 2010.

[40] Goldberg, D. Edward and others, Genetic algorithms in search, optimization and machine learning, vol. 412, Addison-wesley Reading Menlo Park, 1989.

[41] M. Tasgin, A. Herdagdelen and H. Bingol, "Community detection in complex networks using genetic algorithms," *arXiv preprint arXiv:0711.0491,* 2007.

[42] C. Pizzuti, GA-Net: A genetic algorithm for community detection in social networks, Springer, 2008, pp. 1081-1090.

[43] C. Pizzuti, A multi-objective genetic algorithm for community detection in networks, IEEE, International Conference on Tools With Artificial Intelligence. ICTAI, 2009, pp. 379-386.

[44] M. Gong, L. Ma, Q. Zhang and L. Jiao, "Community detection in networks by using multiobjective evolutionary algorithm with decomposition," *Physica A: Statistical Mechanics and its Applications,* 2012.

[45] R. Shang, J. Bai, L. Jiao and C. Jin, "Community detection based on modularity and an improved genetic algorithm," *Physica A:*

Statistical Mechanics and its Applications, vol. 392, no. 5, pp. 1215-1231, 2013.

[46] J. Li and Y. Song, "Community detection in complex networks using extended compact genetic algorithm," *Soft Computing,* vol. 17, no. 6, pp. 925-937, 2013.

[47] M. Gong, X. Chen, L. Ma, Q. Zhang and L. Jiao, "Identification of multi-resolution network structures with multi-objective immune algorithm," *Applied Soft Computing,* vol. 13, no. 4, pp. 1705-1717, 2013.

[48] G. K. Kumar and V. K. Jayaraman, "Clustering of Complex Networks and Community Detection Using Group Search Optimization," *arXiv preprint arXiv:1307.1372,* 2013.

[49] C. Honghao, F. Zuren and R. Zhigang, Community detection using Ant Colony Optimization, IEEE CEC, 2013, pp. 3072-3078.

[50] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E,* vol. 69, no. 2, p. 026113, 2004.

[51] U. Brandes, D. Delling, M. Gaertler, R. G{\"o}rke, M. Hoefer, Z. Nikoloski and D. Wagner, "On modularity clustering," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 20, no. 2, pp. 172--188, 2008.

[52] M. E. Newman, "Fast algorithm for detecting community structure in networks," *Physical review E,* vol. 69, no. 6, p. 066133, 2004.

[53] A. Clauset, M. E. Newman and C. Moore, "Finding community structure in very large networks," *Physical review E,* vol. 70, no. 6, p. 066111, 2004.

[54] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences,* vol. 103, no. 23, pp. 8577--8582, 2006.

[55] Y. Zhang, J. Wang, Y. Wang and L. Zhou, "Parallel community detection on large networks with propinquity dynamics," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[56] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *ConferenceProceedings of the National Academy of Sciences,* vol. 99, no. 12, pp. 7821-7826, 2002.

[57] Wikipedia, "Modularity (Networks)," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Modularity_(networks). [Accessed 10 March 2016].

[58] Y. Gu, S.-L. Shenq, Q. Wu and D. Dasgupta, "On a multi-objective evolutionary algorithm for optimizing end-to-end performance of scientific workflows in distributed environments," in *Proceedings of the 45th Annual Simulation Symposium*, 2012.

[59] M. H. Hajeer, D. Dasgupta and K.-I. Lin, "Distributed Evolutionary Algorithm for Clustering Multi-Characteristic Social Networks," in *Proceedings of the International Conference on Data Mining (DMIN)*, 2014.

[60] A. Semenov, J. Veijalainen, M. Hajeer and D. Dasgupta, "Political Communities in Russian Portion of LiveJournal," in *In the Proceedings of International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, USA, 2014.

[61] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software,* vol. 42, no. 10, pp. 760--771, 2011.

[62] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E,* vol. 69, no. 2, p. 026113, 2004.

[63] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports,* vol. 533, no. 4, pp. 95-142, 2013.

[64] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment,* vol. 2, no. 2, pp. 1626-1629, 2009.

[65] M. Hajeer, D. Dasgupta, A. Semenov and J. Veijalainen, "Distributed evolutionary approach to data clustering and modeling," in *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, Orlando, 2014.

[66] T. SOFTWARE, "Tableau," TABLEAU, [Online]. Available: https://www.tableau.com. [Accessed 1 Oct 2017].

[67] W. Zachary, "An Information Flow Model for Conflict and Fission in Small Groups," *Journal of Anthropological Research,* vol. 33, no. 4, pp. 452-473, 1977.

[68] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron and L. Liberti, "Column generation algorithms for exact modularity maximization in networks," *Physical Review E,* vol. 82, p. 046112, 2010.

Mustafa Hajeer is a computer science Ph.D. recent graduate from The University of Memphis. He has his B.Sc. from Yarmouk University and his M.Sc. from The University of Memphis. Mustafa worked as a programmer at Alkena Group in Jordan and as a computer science instructor at Alqusor Academy. He also worked as a research assistant at the FedEx Institute of Technology within the Institute of Intelligent Systems at The University of Memphis. His most recent internship was done at the Intel Corporation at the Data Center Group and now is working at the same group at Intel. His research involves BigData, Data analysis, and machine learning.

Dipankar Dasgupta is a Professor of Computer Science at the University of Memphis. His research interests are broadly in the area of scientific computing, design, and development of intelligent cyber security solutions inspired by biological processes. He is one of the founding fathers of the field of artificial immune systems in which he has established himself. Dr. Dasgupta is at the forefront of research in applying bio-inspired approaches to cyber defense, having served as a program co-chair at the National Cyber Leap Year Summit organized at the request of the White House Office of Science and Technology Directorate (2009). Dr. Dasgupta has more than 240 publications with 10000+ citations and having h-index of 52 as per Google Scholar. Dipankar Dasgupta is an IEEE Fellow.