Using Property Graphs for Rich Metadata Management in HPC Systems

Dong Dai[1], Robert B. Ross[2], Philip Carns[2], Dries Kimpe[2], and Yong Chen[1]

[1]Computer Science Department, Texas Tech University, {dong.dai, yong.chen}@ttu.edu
[2]Mathematics and Computer Science Division, Argonne National Laboratory, {rross, pcarns, dkimpe}@mcs.anl.gov

***Abstract*—HPC platforms are capable of generating huge amounts of metadata about different entities including jobs, users, and files. *Simple metadata*, which describe the attributes of these entities (e.g., file size, name, and permissions mode), has been well recorded and used in current systems. However, only a limited amount of *rich metadata*, which records not only the attributes of entities but also relationships between them, are captured in current HPC systems. Rich metadata may include information from many sources, including users and applications, and must be integrated into a unified framework. Collecting, integrating, processing, and querying such a large volume of metadata pose considerable challenges for HPC systems. In this paper, we propose a rich metadata management approach that unifies metadata into one generic property graph. We argue that this approach supports not only simple metadata operations such as directory traversal and permission validation but also rich metadata operations such as provenance query and security auditing. The property graph approach provides an extensible method to store diverse metadata and presents an opportunity to leverage rapidly evolving graph storage and processing techniques.**

I. Introduction

Rich metadata describes detailed information about different entities such as users, applications, files, and their relationships. This information extends simple metadata, which contains only predefined attributes about individual entities (e.g., file name, permissions) and basic relationships (e.g., ownership, POSIX namespace), to a more detailed level. Rich metadata can contain arbitrary user-defined attributes and flexible relationships. A typical example of such rich metadata is provenance, which records a complete history of data elements, including the processes that generated it; the user who started the processes; and even the environment variables, parameters, and configuration files used during execution [1, 2]. This level of metadata detail enables a variety of new data management capabilities [3, 4]. For example, the access history of users reading/writing data files can be used to monitor users in shared supercomputer facilities; read/write history from processes can be used to trace back suspicious executions; and provenance capture can be used to regenerate consistent environments for reproducible scientific results.

Existing storage systems capture simple metadata to organize files, control file access, and record access times. Systems including Spyglass [5] and Magellan [6] have also been proposed as extensions to store and utilize this metadata. However, while capturing rich metadata offers numerous advantages, current HPC platforms still lack basic facilities to collect, store, process, and query such metadata. At least three challenges must be addressed.

- *Storage System Pressure*. A leadership supercomputer might include millions of processes, millions of compute cores, and billions of files. Recording data from all of these entities could produce a large volume of data and interfere with I/O performance.
- *Efficient Processing/Querying*. Once the metadata has been collected, processing and querying it efficiently may still be difficult. Use cases such as permission checking require real-time results, and more complex management activities could require extensive processing time. This situation calls for a flexible, distributed processing capability.
- *Metadata Integration*. Rich metadata can be diverse. It can contain predefined attributes and simple relationships as well as user-defined attributes and relationships. Moreover, it may be collected from different components such as file systems, operating systems, or schedulers. These data sources and data formats must be integrated in order to avoid duplication of functionality across management tools.

Previous work has sought to unify metadata, but these approaches relied on either relational databases [7] or key-value storage systems [8] to store the metadata. In this paper, we propose unifying all metadata into one *property graph* in order to integrate rich metadata from different sources. All storage management services and applications can store their rich metadata by using graph storage APIs and can access different categories of metadata by using graph query APIs. The benefits are twofold. First, this directly solves the integration issue by using a single representation: all services and applications will use the same interface to store or process metadata in a single service where we can apply complex optimizations to improve the performance further. Second, the graph model provides a flexible method for processing and reasoning about metadata. Complex queries are easier to express as a graph traversal instead of as a table "join" in relational databases. Also, by abstracting metadata into a graph, we can utilize

rapidly-evolving graph techniques to provide better access speed, flexible query languages, and distributed processing.

This paper is organized as follows. In Section II we introduce the proposed graph model for rich metadata. In Section III we explore the attributes of such graphs by building an example graph using the metadata collected from the Darshan trace of a leadership-class supercomputer (Intrepid). In Section IV we present our strategy for implementing several critical data management functionalities based on the graph model. In Section V we briefly introduce the relevant techniques from the graph storage and processing community and discuss the challenges on current graph infrastructure. In Section VI we summarize our conclusions and briefly propose ideas for future work.

II. Graph-Based Metadata Model

Arguably, researchers already consider metadata as a graph. The traditional directory-based file management constructs a tree structure to manage files, with additional metadata stored in *inodes* [9]. This tree is a graph. The provenance standard (*Open Provenance Model* [10]) captures the provenance of objects by an annotated causality graph, which is a directed acyclic graph enriched with annotations capturing further information.

We generalize these graphs in HPC scenarios and propose the metadata graph model. The metadata graph is derived from the *property graph model* [11], which includes vertices that represent entities in the system, edges that show their relationships, and properties that annotate both vertices and edges and can store arbitrary information that users want. Based on the entities in an HPC environment, we introduce our strategy for mapping arbitrary rich metadata into this property graph model.

A. Entity to Vertex

An HPC platform comprises three basic entities: users, the running applications, and the data files. We define these as three basic types of vertices, as follows.

- *Data Object*: represents the basic data unit in a storage system. In a parallel file system, a data object would represent a file or a directory.
- *Execution*: represents the execution of an application. There are three levels of executions: *Job*, submitted by the user; *Processes*, scheduled from one job; and *Threads*, running inside one process. Different use cases require different levels of execution detail and generate graphs with different sizes. For simplicity, we call all these entities as *Execution* entities.
- *User*: represents an end user of a system.

In addition to these basic entities, users can define their own entities. For example, in a work-flow system, users can create *work-flow* entities and connect them with *Executions* to trace work-flow executions. Also, the system administrators can create *user group* entities to include different *Users* and assign privileges for them. The metadata graph allows users to extend existed entities to build their own entities. The only limitation is that these user-defined entities must connect with existing entities to keep every element in the graph accessible by traveling through the graph.

B. Relationship to Edge

Relationships between entities are shown in Table I. Each cell shows relationships from the row identifier to the column identifier. Each relationship will be mapped to a directed edge in the metadata graph. In Table I, *run* indicates that the user starts an execution; *exe* means the execution is based on a corresponding executable file; and *read/write* indicates the I/O operations from executions to data objects. Since all the relationships are directed, traveling back from *dest* nodes to *src* nodes will be difficult. Therefore, in the current model we define corresponding reversed relationships for each relationship, in order to accelerate the reversed traversal (the *wasXXBy* relationships).

TABLE I
Default Relationships Definition.

	User	**Execution**	**Data Object**
User		*run*	
Execution	*wasRunBy*	*belongs, contains*	*exe, read, write*
Data Object		*exedBy, wasReadBy, wasWrittenBy*	*belongs, contains*

Table I lists several *belongs/contains* relationships. In the Execution entity case, the relationship means that one job contains multiple processes, which in turn belongs to this job. In the Data Objects case, the relationship means that one directory may contain multiple files or directories.

Also, users can create their own relationships based on any two existing entities. For example, two user entities can have a new relationships called *login-together* if they log into the system roughly at the same time.

C. Property

Rich metadata also contains annotations on entities and their relationships. In the metadata graph model, we store these as properties, which are key-value pairs attached on vertices and edges. Users can create their own properties on vertices and edges, but their keys must be unique in each user's namespace. By isolating properties by user, we avoid global contention among different users. Examples of properties include user name, privilege, execution parameters, file permission, creation time, data source agent, data quality score, and execution environment variables.

III. Metadata Graph Prototype

Collecting rich metadata generally requires modification to HPC services in order to collect data from jobs, processes, and read/write operations [2, 12, 13]. To help understand the attributes of a rich metadata graph in an HPC context, we exploit Darshan trace logs as a source of rich metadata for prototyping purposes [14].

A. Mapping Strategy

The Darshan utility is an MPI library that can be linked to applications in order to generate I/O behavior logs during execution [15]. Each Darshan log file represents a distinct job. The log entries of a job contain the ID of the user who started this job, the executable file that the job was based on, the parameters of this execution, some environment variables, and, most important, the file access statistics of each process (ranks) inside this job (MPI program). Note that the collected Darshan traces available on-line have been anonymized, storing only the hashed values of file names, path, user names, and job names.

We map Darshan logs to the metadata graph defined in Section II. Each unique user ID indicates a User entity, each Darshan log file represents a Job, all the ranks inside a job correspond to the Processes, and both the executables and data files are abstracted as Data Object entities. Darshan does not capture directory structure, since it stores only the hashed value of file paths. We therefore synthetically create the simplest directory structures: data files visited by each execution are considered under the same directory, and all these directories accessed by one user are placed under one directory for each user (this structure is possible only in the graph model). Based on this mapping, we were able to import a year's Darshan traces (*2013*) from the Intrepid machine into an example graph [16]. This collection of Darshan logs characterizes the I/O activity of approximately 42% of all core-hours consumed on Intrepid over the course of a year.

B. Graph Size

TABLE II
DARSHAN GRAPH SIZE AND COMPARISONS.[a]

Number	Basic	With I/O Ranks	With Full Ranks	With Directory
Vertices	34.6 M	41.7 M	147.8 M	147.9 M
Edges	126.5 M	133.6 M	239.8 M	366.3 M

	Road Graph USA [17]	Twitter [18]	Facebook [19]	Web Page (2002) [20]
Vertices	24 M	645 M	1.28 B	2.1 B
Edges	29 M	81.4 B	256 B	15 B

[a] M = million; B = billion; T = trillion

The first property of a metadata graph is its size. The top half of Table II shows the graph size of our example metadata graph in terms of vertices and edges for different levels of detail. The first column considers the job as the Execution entity and eliminates all process (rank) information. All I/O behavior is associated with the Job entity. The second column (*With I/O Ranks*) records the ranks that have I/O operations. In many cases, this indicates the rank 0 process or the aggregators in two-phase I/O. The third column (*With Full Ranks*) records all the ranks as Processes entities whether they performed I/O or not. The last column shows the graph size with the complete synthetic directory structures and full ranks. This table clearly shows that increasing the level of detail will dramatically increase the graph size. The metadata graph model can be tuned for different use cases to strike a balance between performance overhead and metadata capability.

In the bottom half of Table II, we show several example large-scale graphs from different fields for comparison, including a road map graph (e.g., USA map), social network (e.g., Twitter, Facebook), and Internet web pages (estimation of 2002). Comparing these, we note that although the metadata graph is large, graphs of this size are already manageable in many existing systems.

C. Graph Structure

In addition to the graph size, the graph structure is a critical property of metadata graphs for storage and processing. Before discussing the graph structure, we briefly list in Table III different entities of the example metadata graph.

TABLE III
STATISTICS OF METADATA GRAPH.

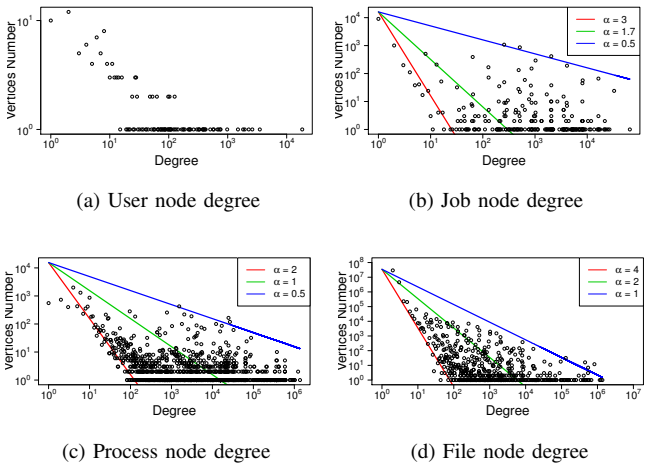
	Users	Jobs	Proc.	Ranks	Files
Num	177	47,592	10,085,931	113,278,038	34,608,033

This table shows that different entities in HPC environment may have totally different sizes; hence, the edges between them also vary considerably. In this section, we consider them separately. Based on the mapping strategy, the degree of a user node indicates how many jobs each user submitted; the degree of a job node shows how many processes it contains; and the degree of each process node shows how many files are read or written by it. The file node degree shows all data accesses including execution, reading, and writing.

Figure 1 shows the node degree distributions of four basic entities: User, Job, Process, and File (Data Object) in the metadata graphs. In these figures, the x-axis denotes the degree, and the y-axis shows the number of vertices that have that degree. Both the x-axis and y-axis are $\log_{10}$ values. All four entity types have the common attribute that most entities have very small degrees and a small number of entities have much larger degrees. Take *User* node as an example. A total of 177 users submitted jobs in 2013. Among them, around 10% of the users actually submitted more than 80% of all the jobs. We observe a similar phenomenon in the other three entities.

Many graphs have this attribute. They can be described by using the *skewed* power-law degree distribution [21], which means that most vertices have relatively few neighbors, while a few vertices have many more neighbors. We use the function $\mathbf{P}(d) \propto d^{-\alpha}$ to describe the probability that a vertex has degree d in such graphs. Here, α is a constant parameter of the distribution known as the *exponent* or *scaling parameter*.

From Fig. 1, we speculate that some of the entities fit the power-law distribution. We therefore plot three lines of the power-law distribution with different α values in each figure (we eliminate *User* because there are only 177 user samples, which is too few to visualize the distribution). We observe that the *File* nodes come closer to the power-law degree distribution as Fig. 1(d) shows. This observation is intuitive because of locality; most files are seldom accessed, but a small



(a) User node degree

(b) Job node degree

(c) Process node degree

(d) File node degree

Fig. 1. Node degree for different entities.

subset of files are visited frequently. The *Process* and *Job* degree distributions do not fit the power-law distribution quite as well. For *Process* (Fig. 1(c)), there are not enough low-degree processes nodes; and for *Job* (Fig. 1(b)), the distribution scatters more randomly than for the other two.

We plan to investigate whether additional use cases fit the power-law distribution and whether adding or deleting nodes or edges will change this attribute based on the statistical analytical strategy introduced by Clauset et al. [22]. Preliminary results still strongly indicate that the power-law distribution is an accurate estimation for these basic entities in metadata graphs. These results play an important role for our future design of graph partition and storage strategy. Moreover, they serve as a basis for creating synthetic graphs for evaluation, since collecting large-scale rich metadata in running HPC systems is an open challenge.

IV. Use Cases For the Metadata Graph

Unifying rich metadata into one graph turns many appealing data management functionalities into graph traversal operations or graph queries. In this section, we show how to map use cases from real-world scenarios to graph operations.

A. User Audit

Data auditing is critical in large computing facilities where different users share the same cluster. It requires detailed user-to-file access history for future security checks. In the metadata graph, we already collect the *run* relationships between Users and Executions, and the *read/write* relationships between Executions and Data Objects will record the file access history (all those relationships will contain properties such as timestamps). In this configuration, we can (for example) find all the files that were accessed by a specific user in time frame $[t_s, t_e]$ by using graph operations as follows: (1) locate the given user from the metadata graph; (2) travel through *run* edges from this User node to Execution nodes; (3) filter executions based on the given time frame; and (4) travel through the *read* edges from the remained executions to the final files.

B. Hierarchical Data Traversal

Hierarchical data organization is used to present a logical layout of data sets to users. The simplest example of hierarchical data traversal is traditional directory namespace traveling. In the metadata graph model, we abstract both directories and files as Data Object entities. The *belongs* and *contains* relationships between different Data Objects represent the relationships between files and directories. Given an absolute path, locating a file translates into traversing *contains* edges from a Data Object node and filtering edges as needed.

Metadata graphs also offer the potential to leverage existing techniques for scalable data structure traversal. Traditional POSIX directories often do not perform well for HPC systems where millions of files may be stored under one directory. Previous work such as Giga+ [23] has been proposed to improve performance by distributing directory metadata across multiple servers. In a graph model, this becomes a graph partition problem, which is already studied extensively in the literature [24, 25, 26].

In addition to traditional POSIX-style files and directories, semantic data management would be another hierarchical traversal use case. Scientists could manage their data in a semantic way, such as arranging all the inputs and outputs of a single simulation execution together. Traditionally, doing so requires careful file naming and directory placement. In a metadata graph, we can simply create new entity named Simulation and connect it with the Data Object. Data can be organized across multiple dimensions simultaneously by using this approach.

C. Provenance Support

Provenance has a wide range of use cases, such as data sharing, reproducibility, and work-flow management. As a superset of provenance, the metadata graph model naturally supports these use cases. In this subsection, we borrow the problem from the first Provenance Challenge as an example [27].

In this challenge, an example work-flow was provided as the basis. A workable provenance system should be able to represent the work-flow and all the relevant provenance and, most important, be able to answer predefined queries. Based on our proposed graph model, it is straightforward to abstract the work-flow as series of executions run by the same user, and each execution reads files (Data Objects) and generates outputs for applications in the next phase. Based on this work-flow, the provenance system needs to answer queries such as *Find the execution whose model is AlignWarp and inputs have annotation* ['center':'UChicago']. This can be expressed in a metadata graph as follows: (1) query all the Execution vertices, which have *exe* out-edge pointing to a Data Object named 'AlignWarp'; (2) start from all those Execution vertices and filter out the executions whose property "center" is not "UChicago."

A notable advantage of the metadata graph when compared with a traditional provenance system is that it allows users to cross-reference different categories of metadata in an unified way. If the provenance query needs other metadata (e.g., file size, permission mode, or user group information), processing that metadata in a unified graph will be more efficient and straightforward.

V. Graph Facilities and Challenges

Graph databases and distributed processing frameworks are two basic facilities to support our metadata graph model. Although a large number of these facilities exist, challenges remain because of the specific requirements of storing, processing, and querying metadata graphs of this scale.

A. Graph Databases

The graph databases are designed to cover the requirements of complex graph-based relationships, which are not well-suited to traditional relational databases. The past few years has seen an increasing number of graph database implementations, including AllegroGraph, DEX, G-Store, HyperGraphDB, InfinitGraphDB, and Neo4j etc. [28, 29, 30, 31, 32, 33] They can be categorized based on different metrics. Based on the storage device, there are in-memory databases and disk-based databases. Based on the supported graph data structure, there are simple graphs databases, hypergraphs databases, and property graphs databases.[1] Based on distributed deployment, there are single server databases, high availability databases, and distributed databases.

A metadata graph would require support for property graphs and distributed disk-based storage (since an HPC storage system graph is likely to be too large to fit in memory). Several implementations satisfy such requirements, such as Titan, DEX, and Neo4j. Performance is an important consideration as well. In a distributed environment, updates across servers will significantly reduce performance, so we need to consider the structure of the metadata graph and provide an optimized storage layout. Another performance challenge is graph traversal. In fact, traveling through a metadata graph usually includes applying filters and computations on properties during traveling, as in the provenance example shown in Section IV-C. Since these properties are too big to be fully cached in memory, we expect to load them from persistent devices. An intelligent cache strategy could help mitigate random seek costs in this scenario.

B. Graph Processing

In addition to graph databases, graph processing frameworks can be used to perform computation or queries on graphs in a distributed way. Typical examples of these frameworks include Giraph [35], which was designed and implemented based on the Pregel computing model [36]; GraphX [37], which was based on the Spark computing framework [38]; GraphLab [39], and X-Stream [40]. These processing frameworks are a complement to the querying and searching capability provided by graph databases. For example, we can run *community discovery* algorithms on a metadata graph to find the "closely" related data files. The results can be used to optimize physical placements for better I/O performance. These algorithms may iteratively traverse a graph over an extended period of time.

However, most these distributed graph processing frameworks were designed to work on unstructured graphs, which are usually simply stored as a plain file in adjacency list formats in a general storage back-end. There is a gap from deploying graph algorithms in these plain graph formats to running these algorithms on a graph database, which usually has a specific, optimized storage layout for querying and searching.

[1]Here, the simple graph indicates a graph defined as a set of nodes connected by weighted edges. Hypergraphs extends simple graphs by allowing an edge to relate an arbitrary number of nodes [34]. A property graph indicates a graph where nodes and edges contain properties. This property graph is the basis of our proposed metadata graph model.

VI. Conclusion and Future Work

In this paper, we proposed an idea of unifying rich metadata in a HPC platform into a property graph model, which supports a wide range of metadata management functionalities in a simple but efficient way. By prototyping such a metadata graph from Darshan I/O traces of a real-world leading supercomputer, we explored the attributes of such graphs. Based on the prototype, we introduce existing graph facilities and discuss their remained challenges for storing and processing the metadata graph. We discuss the benefits of unifying HPC metadata into a graph, and we present the feasibility of implementing such a graph in current HPC platforms. Future work will include implementing such a platform with optimized graph facilities in order to provide a practical metadata solution for exascale data management systems.

Acknowledgment

This material is based upon work supported by the U.S. Department of Defense; by the U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357; and by the National Science Foundation under grant CNS-1338078 and CNS-1162488.

References

[1] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and Where: A Characterization of Data Provenance," in *Database Theory ICDT 2001*. Springer, 2001, pp. 316–330.

[2] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware Storage Systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*. USENIX Association, 2006.

[3] Y. L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.

[4] C. T. Silva, J. Freire, and S. P. Callahan, "Provenance for Visualizations: Reproducibility and Beyond," *Computing in Science & Engineering*, vol. 9, no. 5, pp. 82–89, 2007.

[5] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems," in *FAST*, vol. 9, 2009, pp. 153–166.

[6] A. Leung, I. Adams, and E. L. Miller, "Magellan: A Searchable Metadata Architecture for Large-Scale File Systems," *University of California, Santa Cruz, Tech. Rep. UCSC-SSRC-09-07*, 2009.

[7] S. N. Jones, C. R. Strong, A. Parker-Wood, A. Holloway, and D. D. Long, "Easing the Burdens of HPC File Management," in *Proceedings of the sixth workshop on Parallel Data Storage*. ACM, 2011, pp. 25–30.

[8] J. C. Mogul, "Representing Information about Files," Ph.D. dissertation, Citeseer, 1986.

[9] A. S. Tanenbaum and A. Tannenbaum, *Modern Operating Systems*. Prentice Hall, Englewood Cliffs, NJ, 1992, vol. 2.

[10] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The Open Provenance Model Core Specification (v1. 1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[11] "Property Graph," http://www.w3.org/community/propertygraphs/.

[12] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor, "Layering in Provenance Systems," in *Proceedings of the 2009 USENIX ATC*, 2009.

[13] U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer, "Issues in Automatic Provenance Collection," in *Provenance and Annotation of Data*. Springer, 2006, pp. 171–183.

[14] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 Characterization of Petascale I/O Workloads," in *Cluster Computing and Workshops, CLUSTER'09. IEEE International Conference on*, pp. 1–10.

[15] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and Improving Computational Science Storage Access through Continuous Characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, p. 8, 2011.

[16] "Darshan data." ftp://ftp.mcs.anl.gov/pub/darshan/data/.

[17] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Mathematical Soc., 2009, vol. 74.

[18] "Twitter Statistics," http://www.statisticbrain.com/twitter-statistics/.

[19] A. Ching, "Giraph: Production-Grade Graph Processing Infrastructure for Trillion Edge Graphs," in *ATPESC*, ser. ATPESC '14, 2014.

[20] J.-L. Guillaume, M. Latapy *et al.*, "The Web Graph: an Overview," in *Actes d'ALGOTEL'02*, 2002.

[21] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[22] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law Distributions in Empirical Data," *SIAM Review*, vol. 51, no. 4, pp. 661–703, 2009.

[23] S. Patil and G. A. Gibson, "Scale and Concurrency of GIGA+: File System Directories with Millions of Files." in *FAST*, vol. 11, 2011, pp. 13–13.

[24] M. Kim and K. S. Candan, "SBV-Cut: Vertex-cut based Graph Partitioning Using Structural Balance Vertices," *Data & Knowledge Engineering*, vol. 72, 2012.

[25] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs." in *OSDI*, vol. 12, no. 1.

[26] A. Abou-Rjeili and G. Karypis, "Multilevel Algorithms for Partitioning Power-Law Graphs," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[27] "Provenance Challenges," http://twiki.ipaw.info/bin/view/Challenge/.

[28] "AllegroGraph," http://franz.com/agraph/allegrograph/.

[29] "DEX," http://www.sparsity-technologies.com/.

[30] R. Steinhaus, D. Olteanu, and T. Furche, "G-Store: A Storage Manager for Graph Data," Ph.D. dissertation, Citeseer, 2010.

[31] B. Iordanov, "HyperGraphDB: A Generalized Graph Database," in *Web-Age Information Management*. Springer, 2010, pp. 25–36.

[32] "InfiniteGraph," http://www.objectivity.com.

[33] J. Webber, "A Programmatic Introduction to Neo4j," in *Proceedings of the 3rd annual conference on Systems, Programming, and Applications: Software for Humanity*. ACM, 2012, pp. 217–218.

[34] C. Berge and E. Minieka, *Graphs and Hypergraphs*. North-Holland Publishing Company, Amsterdam, 1973, vol. 7.

[35] "Giraph," http://giraph.apache.org/.

[36] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a System for Large-Scale Graph Processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[37] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: A Resilient Distributed Graph System on Spark," in *First International Workshop on Graph Data Management Experiences and Systems*, 2013, p. 2.

[38] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[39] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Graphlab: A New Framework for Parallel Machine Learning," *arXiv preprint arXiv:1006.4990*, 2010.

[40] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-Centric Graph Processing using Streaming Partitions," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013.