# Evaluating Use of Data Flow Systems for Large Graph Analysis

Andy Yoo and Ian Kaplan

Lawrence Livermore National Laboratory
Livermore, CA 94551

## Abstract

*Large graph analysis has become increasingly important and is widely used in many applications such as web mining, social network analysis, biology, and information retrieval. The usually high computational complexity of the commonly-used graph algorithms and large volume of data frequently encountered in these applications, however, make scalable graph analysis a non-trivial task. Recently, more and more of these graph algorithms are implemented as dataflow applications, where many tasks perform assigned operations in parallel independent of other tasks. These applications are run on large-scale computing platforms to combine the advantages of the data parallelism enabled by dataflow model and the high computing power and large storage capacity offered by increasingly affordable high-end computers. In this paper, we evaluate the potentials of many-tasks concept in a form of dataflow system for large graph analysis applications by studying the performance of complicated graph algorithms on an actual dataflow machine. We have found that a dataflow system can achieve orders of magnitude performance improvement over state-of-art database systems and serve as a viable scalable graph analysis engine.*



MTAGS '09 November 16th, 2009, Portland, Oregon, USA
(c) 2009 ACM ACM 978-1-60558-714-1/09/11... $10.00

## 1 Introduction

Large graph analysis has become increasingly important and is widely used in many applications such as web mining, social network analysis, biology, and information retrieval. Typically, graphs analyzed consist of typed vertices such as `person` and `organization` and typed edges that represent the relationships between the vertices such as `works_for` and `visit`. Vertices and edges also have attributes associated with them (e.g., *name* and *date_of_visit*).

These graphs, which are usually formed by fusing fragmental information obtained from many different sources like web documents, news articles, and public records, have very complex structures. The graphs also tend to continue to grow in size, and it is not uncommon for scientists and knowledge engineers to deal with graphs with billions of vertices and edges in practice. The large size and complexity of the graphs, unfortunately, make the graph analysis an inherently difficult problem. Furthermore, many of the queries performed on real-world graphs are very complicated with high-order complexity and generate a large volume of intermediate results [7, 19, 16, 14, 12, 13, 3]. This makes analyzing large graphs in a scalable and efficient manner even more challenging.

Graph data has been commonly stored in conventional relational databases mainly due to their availability and ease of use. However, using relational databases to run graph analysis applications has significant impact on query performance, since the complicated graph queries usually are implemented using a set of expensive SQL operations. Their in-

ability to scale becomes evident when the graph queries are executed on very large graph data [7]. In addition, the SQL query performance is hard to optimize, because how an SQL code is translated into underlying primitives is largely opaque to users and varies widely from one SQL compiler to another. These limitations of the relational databases as a graph analysis engine warrant investigation of alternative systems for the scalable graph analysis.

Many-task computing (MTC) [17] is an ideal alternative computing paradigm that enables scalable large graph analysis. MTC denotes high-performance computations comprising multiple distinct activities that are carried out by a set of tasks, coupled via file system operations. There are many variations of MTC depending on the characteristics of the tasks. The tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. The tasks may be static or dynamic, homogeneous or heterogeneous, loosely coupled or tightly coupled. The aggregate number of tasks, quantity of computing, and volumes of data may be extremely large.

Recently, more and more of the graph algorithms are implemented as MTC applications and run on large-scale computing platforms. These efforts are exemplified by Google's PageRank algorithm implemented in Map/Reduce programming framework [15], where the computation is done by a large number of loosely-coupled Map and Reduce tasks.

The Map/Reduce programming model, however, has significant limitations. First, the programming model is too primitive for many complex graph algorithms (although its simplicity is its strengths for embarrassingly-parallel applications). Second, the volume of intermediate results that have to be read and processed by Map or Reduce tasks for the graph analysis applications usually increases rapidly and therefore quickly becomes a performance bottleneck that hampers scalable graph analysis. We need more general and powerful tool.

We believe dataflow model [4, 5, 6, 8] is a promising alternative that can offer flexible and scalable solutions for large graph analysis problems. A dataflow system follows MTC paradigm, since reading, processing, and writing data on typical dataflow systems is performed by many built-in and user-defined tasks. In this sense, the Map/Reduce is also considered as a (primitive form of) dataflow system.

In this paper, we investigate the use of dataflow systems as a tool to run complex queries on large graphs. We have developed a suite of benchmarks that encompass a wide range of complex graph queries. Then, we have evaluated a state-of-art parallel dataflow system by running the graph benchmark on a very large real-world graph. The performance of the system is measured and compared to that of advanced relational database systems. We have found that a dataflow system can achieve orders of magnitude performance improvement over these database systems and serve as a viable scalable graph analysis engine.

The paper is organized as follows. Section 2 describes a dataflow system evaluated in this research. We discuss the programming environment of the target dataflow system in Section 3. Section 4 describes benchmarks used in our experiments. The results from performance study are discussed in Section 5, followed by concluding remarks in Section 6.

## 2 Testbed Dataflow System

Dataflow is a simple and powerful model that is believed to offer a basis for a scalable architecture to replace the relational databases for graph analysis. In dataflow model, there is no notion of a single locus of control. The dataflow model describes computation in terms of locally controlled events where each event corresponds to the firing of an actor [10].

These actors are implemented as various tasks that run in parallel independent of each other, enabling data parallelism. This inherent data parallelism facilitates the development of parallel data management system that can process different portions of data simultaneously in the most scalable and efficient manner.

In this research, we ran experiments on an actual dataflow system called *data analytics supercomputer* (DAS) [9], which is a dataflow system based on the *active disk* architecture [18]. The DAS system can be built and operate virtually on any commodity clusters.

DAS system is comprised of a set of servers.

Many of these servers are usual components used by resource management systems for efficient and reliable operations on the target system. A key component here is the *data storage servers* that run on those cluster nodes reserved to store and manage the data. Like Map/Reduce that operates on a distributed file system (e.g., Hadoop's Map/Reduce on Hadoop distributed file system), the DAS system stores data in a distributed file system, which is implemented by the data storage servers. Furthermore, the DAS system provides users with a global name space for their data files, enabled by the data storage servers. Another key component of the DAS system is *language server*, which essentially translates user queries to executables that run on data storage nodes.

The DAS testbed at Lawrence Livermore National Laboratory (LLNL) consists of 40 Sun Fire X2100 servers, out of which 20 nodes are reserved for data storage. Each node in the testbed has a dual-core AMD Opteron processor running on 2.6 GHz with 4 GB of memory and 400 GB of disk. The details of the design and usage of the system is further articulated in [1, 2].

# 3 DAS System Execution Environment

In this section, we describe the execution environment of the DAS system, including the dataflow language to program user queries and the flow of work units.

Users write their queries using a dataflow description language called *Enterprise Control Language* (ECL) on DAS system, in contrast to SQL for queries on relational databases. A user of the DAS system should view his query in terms of the flow of data through various data manipulation constructs and decompose the query into a series of operations that can be implemented using given constructs, which are eventually executed by a set of parallel independent tasks. Then these constructs are linked to form a dataflow diagram, which is then translated into a query in ECL.

ECL is simply a language used to define dataflow diagrams. The language is non-procedural, as there is no means to control the flow of data. Hence, the order of ECL statements in a query is irrelevant to the order they are executed. ECL codes consist of set of declarations. These typically include the declarations of the records used in the queries and the definition of actors that handle data. Figure 1 shows a sample ECL code that finds a set of unique vertex IDs in given graph.

```
edge_rec := record
    integer src;
    integer dst;
end;

edgesr := dataset('edges', edge_rec);
edgesd := distribute(edgesr, hash(src));
edgess := sort(edgesd, src, local);
edges := dedup(edgess, src, local);
output(adjacent);
```

Figure 1: A sample ECL code that finds a set of unique vertex IDs in given graph.

Once a user composes a query in ECL, she submits it to the DAS system for execution via SOAP-based client tool. The submitted query is then sent to a special server called *ECL server*, where the submitted ECL query is translated to C++ codes. The C++ codes are then compiled to a shared object, which is then sent to the data storage servers. Each data storage server links the received shared object with DAS data manipulation library to create an executable and starts execution. Results of the query are returned to user via the client tool.

The core of the DAS system with which it boosts query performance lies in the data manipulation library. The library, which implements all data manipulation constructs in ECL, is highly optimized for scalable and efficient manipulation of data. First, all the constructs in the library are designed to process incoming data in a pipelined fashion, increasing the data parallelism. Second, the library is designed to maximize the usage of memory to achieve the best possible performance. Finally, as the query performance is dominated by only a handful of data operations, the system is optimized for the fast execution of the ECL constructs for these critical operations, specifically `Sort` and *Join* operations. We believe that the optimized data

manipulation library, in conjunction with the flexibility offered by dataflow model, enables the DAS system to process very large graph data in a scalable and efficient way.

# 4 Graph Query Benchmark

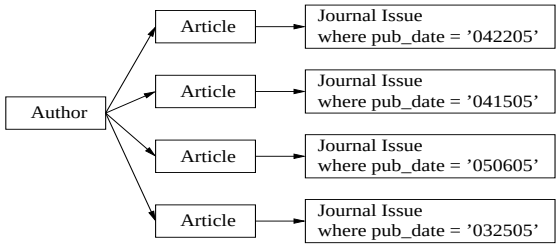
We developed a suite of graph queries for the target graph (PubMed) to represent a wide spectrum of complex queries [7, 21] as benchmarks in this study. These benchmarks are salient since they represent those queries that information analysts are likely to issue. Furthermore, they serve as an excellent tool to measure the scalability of a system since they tend to generate a large volume of intermediate results to push a system's capability to its limit.

The benchmarks consist of two types of graph queries: *subgraph pattern matching* and *graph search*. Given a graph and a template graph (also known as a pattern graph), a subgraph pattern matching attempts to find all the instances of subgraphs in the graph that match the given template. Graph search query is used to find paths between two vertices in the graph.

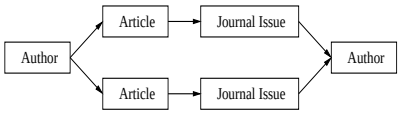
The first query (Query 1) is a highly constrained form of pattern query. This query attempts to find all the authors who published articles in four specific dates. This query is first broken into four smaller patterns, each of which represents the instances of an author-to-date path. The four sets of instances are joined later to find authors who are found in the all four groups.

The second query (Query 2) shown in Figure 2.b finds authors who have published two articles in the same journal. Similarly to the Query 1, all the instances of authors who have published any articles in any journals are detected first and then a self-join is performed to find a set of authors who are found in the both sets of instances.

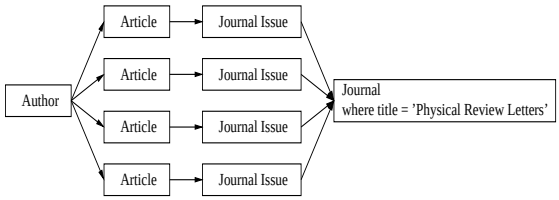
Figure 2.c depicts a pattern for the authors who have published four articles in a journal called *Physical Review Letters* (Query 3). As the first step, a path query for all the authors who have published articles in the particular journal is performed. Successive joins of intermediate results will eventually return a set of authors who published more than four papers in the journal.



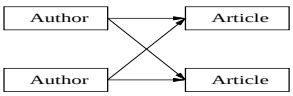

(a) Query 1



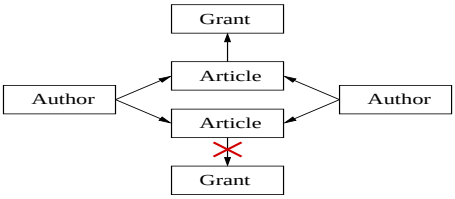

(b) Query 2




(c) Query 3




(d) Query 4




(e) Query 5

Figure 2: A suite of subgraph pattern queries used in performance study.

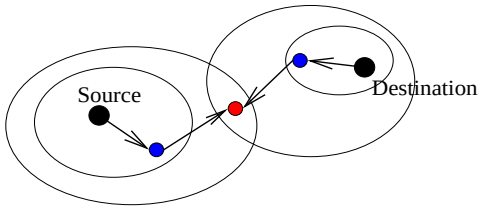



Figure 3: Bidirectional BFS graph query in performance study.

Query 4 shown in Figure 2.d finds two authors who co-authored two or more papers. For this query, all the instances of two authors who co-authored any papers are identified first followed by a self-join.

Figure 2.e presents Query 5 that represents sophisticated queries in which specific types of edges are excluded. Here, the Query 5 finds the instances of two articles that do and do not have associated grants, respectively. The pattern benchmarks are depicted in Figure 2. Readers should refer [7] for more details on the implementation of the graph pattern queries.

The graph search queries used in the experiments are based on conventional breadth-first search (BFS) algorithm. Though simple, the BFS algorithm is an important benchmark, because the BFS is a fundamental algorithm that is used by a wide range of common graph mining algorithms and the execution of BFS algorithm exhibits random memory access behavior that is shared by majority of graph algorithms. Two types of BFS algorithms are examined in this paper: uni- and bi-directional BFS.

The uni-directional BFS is the ordinary BFS algorithm, which finds the shortest path between given source and destination vertices. The bi-directional BFS is a variation of the uni-directional BFS, where two separate breadth-first searches start from the source and destination vertices. These searches continue until a common vertex that can be reached from both ends is found or all the vertices in the graph are visited. Figure 3 describes the bi-directional BFS algorithm pictorially.

# 5 Performance Study Results

We conducted experiments to measure the performance of benchmark queries on the DAS system. We also ran the same queries on state-of-art relational database machines and compared the performance. We report the results in this section.

We used a real-world graph constructed from the PubMed data [20], which contains information on articles published in medical journals such as title, authors, keywords, and abstract, is used in this performance study. The raw PubMed data is scanned first to extract the entities and the relationship between the entities, and then the graph is created as defined by a given ontology. The constructed PubMed graph has about 30 million vertices and 540 million edges. A subgraph with about 1 million vertices is also prepared by sampling the full-scale PubMed graph for studying the performance of smaller systems.

We first measured the scalability of the graph search benchmark on the full-scale PubMed graph. Figure 4 shows the results from strong scalability experiments, where we measure the performance of the search as we increases the number of nodes. Both horizontal and vertical axises are presented in logarithmic scale for clarity.

The DAS system scales very well as shown in the figure. Typically, strong-scaling curve (for run time) decreases as the number of nodes (or processors) increases and then flattens out. In Figure 4, however, the scalability curve continues to decline as the number of nodes increases. We believe this is attributed to that performance gain from processing more data in memory due to reduced per-node problem size outweighs the increased communication overhead due to the increased number of nodes.

Table 1 presents uni- and bi-directional BFS search performance on the DAS system. Full-scale PubMed graph is searched in this experiment. Each search is conducted to find the shortest path between two randomly chosen vertices. The search performance is obtained from the average of 20 searchers. The performance of optimized and unoptimized searches is compared in the table. Here, the search is optimized by transforming input edge table to an adjacency list to reduce the amount of processed data. As Table 1 indicates, the optimized

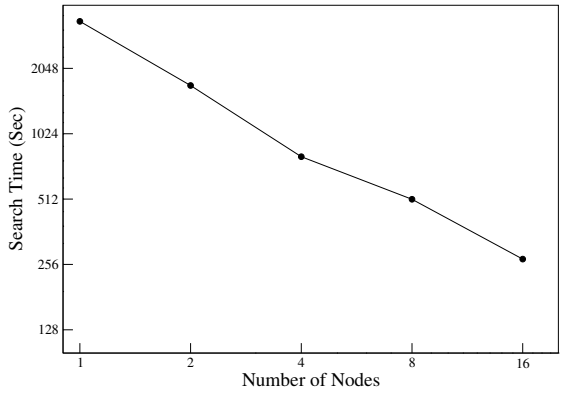



Figure 4: Strong scaling results for BFS graph benchmark on DAS system. A full-scale PubMed graph with 30 million vertices are used in this experiment.

BFS algorithms achieved 2.4X and 3.6X speedups over their unoptimized counterparts.

The performance of the queries on the DAS system is compared to that of equivalent SQL queries on a single-node Oracle RDBMS server in Table 2. The smaller PubMed graph is used here, due to the limited hardware resources on the Oracle server. As Table 2 shows, the queries on the dataflow system outperform corresponding SQL queries on Oracle by orders of magnitude in most cases, achieving as much as 16X speedup per node. Such performance improvement can be attributed to the fact that the the dataflow model allows certain optimization that maximizes data parallelism and results in small intermediate results and minimal communication. Highly optimized underlying query execution engine of the DAS system also contributes to the fast execution of the queries.

The performance of the same set of queries for the large-scale PubMed graph on the DAS system and a 54-node Netezza Performance Server (NPS) [11], a specialized active-disk based distributed relational data management system, is compared in Table 3. We relied on the Netezza's SQL compiler for the query optimization. As the table shows, the DAS system achieves almost two orders of magnitude speedup per node over the Netezza system. The performance improvement is most noticeable for the Queries 3 and 4. This is because these queries consider all the possible combinations of intermediate results, easily leading to combinatorial explosion of data. Since the amount of the intermediate data to be joined is reduced to a great extent by the distribution/localization method for the dataflow queries, the negative effect of the combinatorial explosion is small on DAS.

We also measure the effect of some common optimization techniques on the query performance. The optimization techniques we tested are *column reduction* (CR), *row reduction* (RR), and *distribution/localization* (DIST/LOC). The column reduction basically concerns with reducing the number of columns in tables by constructing virtual tables containing only those columns that are needed for processing subsequent queries. The row reduction concerns with reducing the number of rows in the

| | Unoptimized | Optimized |
|---|---|---|
| Uni-dir. BFS | 287.926 | 120.359 |
| Bi-dir. BFS | 204.902 | 56.431 |

Table 1: Performance of uni- and bi-directional BFS searches on the DAS system (in seconds). Full-scale PubMed graph with 30 million vertices is used in this experiment.

| | DAS | RDBMS | Speedup/Node |
|---|---|---|---|
| Query 1 | 0.343 | 3 | 0.44 |
| Query 2 | 3.182 | 1014 | 15.93 |
| Query 3 | 0.573 | 21.8 | 1.90 |
| Query 4 | 1.742 | 387 | 11.11 |
| Query 5 | 4.575 | 282 | 3.08 |

Table 2: Comparing the performance of queries on DAS and Oracle RDBMS for 1 million-vertex PubMed graph (in seconds).

| | DAS | Netezza | Speedup/Node |
|---|---|---|---|
| Query 1 | 9.422 | 27.3 | 7.82 |
| Query 2 | 142.099 | 834.47 | 15.86 |
| Query 3 | 469.511 | 15392.96 | 88.52 |
| Query 4 | 37.803 | 741.42 | 52.95 |
| Query 5 | 44.6 | 496.48 | 30.06 |

Table 3: Comparing the performance of queries on DAS and Oracle RDBMS for 30 million-vertex PubMed graph (in seconds).

tables by eliminating any duplicate rows and filtering out the rows that do not satisfy the constraints of given queries.

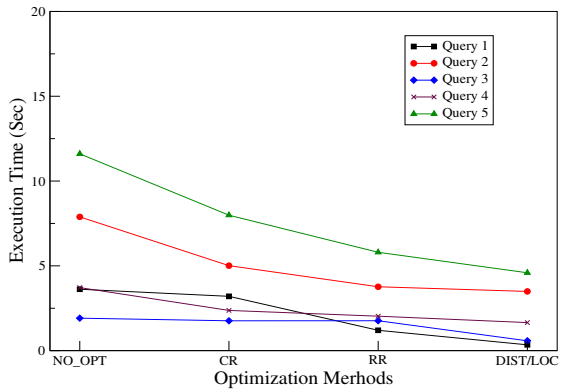
The basic idea of the data distribution and query localization is to distribute data in such a way that subsequent operations after the distribution can be performed only on the local data stored in each node. The distribution of data requires data communication, but it has been observed that the performance gain obtained from localized data operations outweighs this overhead.

Both CR and RR techniques aim at reducing the volume of data to be processed to improve the performance, whereas the DIST/LOC technique focuses on reducing the communication time by eliminating need to transfer large volume of data. Combining these techniques, we can improve the query performance significantly.

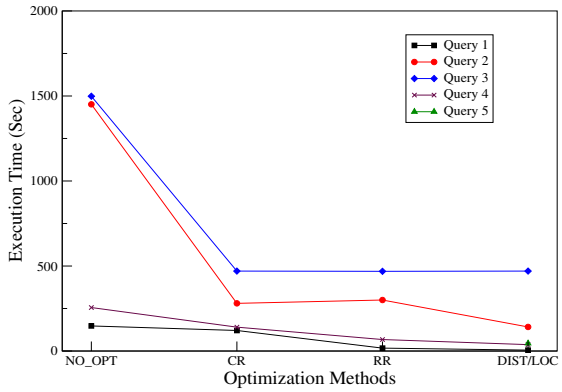
Figure 5 presents the effect of the aforementioned optimization techniques on the query performance. Three optimization techniques are gradually applied to the baseline (unoptimized) queries. Hence, the data points for DIST/LOC in the graphs correspond to the performance of fully optimized queries.

As shown in the figure, the simple column reduction can improve the performance significantly. Its impact is more evident for the Queries 2 and 3 in Figure 5.b, as these queries are more complicated and generate larger intermediate results than others. In contrast, Query 1 contains constraints that its baseline implementation can take advantage of to filter out a large number of records prior to join operations, and therefore, the effect of the column reduction is minimal as shown in Figure 5.b. The performance impact of the column reduction for Query 3 is not clear in Figure 5.a, because a smaller PubMed graph is used in the experiment.

In Figure 5, the effect of the row reduction varies to great extent largely depending on the queries, since some queries benefit from this optimization as it offers smaller tables for fast joins, whereas for others the number of rows reduced is too small to have significant performance impact. It was observed that DIST/LOC technique improves the performance of all the queries examined in the experiments, because this technique basically can eliminate any data skews. It is clear that the effect of the optimization techniques varies to great extent




(a) Effect of optimization for small PubMed graph




(b) Effect of optimization for large PubMed graph

Figure 5: Measuring the effect of optimization techniques on the query performance. A smaller PubMed graph with 1 million vertices and a full-scale PubMed graph with 30 million vertices are used in this experiment. Each technique is gradually applied in optimizing the target queries.

depending on the query and data. However, the fact that using any combinations of the techniques does not degrade performance as shown in Figure 5 suggests that users should exercise these optimization techniques when writing a query.

# 6 Concluding Remarks and Future Work

We evaluated the use of dataflow systems, where a large number of tasks operate independently in parallel to perform given data processing job, as a platform to analyze large-scale graphs in this paper. We developed a suite of benchmarks that encompass a wide range of complex graph queries for the performance study. We measured the performance and scalability of the benchmarks on an actual dataflow machine. Their performance is also measured to that on state-of-art parallel relational database machines. The results show that we can achieve orders of magnitude improvement in query performance on dataflow systems compared to the advanced relational database systems, suggesting that the dataflow machines can be a viable and scalable alternative to run large complex graph queries.

The future research will be focused on other applications that will benefit to the fullest extent from the inherent data parallelism offered by the dataflow model. In an early effort, we are investigating the use of the dataflow system to disambiguate the names of authors that appear in different articles published in technical journals. The algorithm is data-intensive in that it performs little computation but requires a large number of data to be transfered from a database before computation, and hence an ideal application for dataflow system in which the computation is performed where the data is. Algorithms for other graph analysis algorithms including community detection and link analysis will also be developed in dataflow programming model

# References


[1] D. Bayliss, R. Chapman, J. Smith, O. Poulsen, G. Halliday, and N. Hicks. Query scheduling in a parallel-processing database system, 2007. US Patent 7185003.

[2] D. Bayliss, R. Chapman, J. Smith, O. Poulsen, G. Halliday, and N. Hicks. System and method for configuring a parallel-processing database system, 2007. US Patent 7240059.

[3] D. Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[4] J. B. Dennis and G. R. Gao. An efficient pipelined dataflow processor architecture. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE conference on Supercomputing*, pages 368–373, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[5] J. B. Dennis and D. P. Misunas. A preliminary architecture for a basic data-flow processor. In *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, pages 125–131, New York, NY, USA, 1998. ACM.

[6] G. Estrin and R. Turn. Automatic assignment of computations in a variable structure computer system. *IEEE Trans. Electronic Computers*, EC-12(5):755–773, 1963.

[7] I. Kaplan, G. Abdulla, T. Brugger, and S. R. Kohn. Implementing graph pattern queries on a relational database. Technical Report LLNL-TR-400310, Lawrence Livermore National Laboratory, 2008.

[8] R. M. Karp and R. E. Miller. Parallel program schemata: A mathematical model for parallel computation. In *FOCS*, pages 55–61, 1967.

[9] Lexis-Nexis Data Analytics Supercomputer. `http://www.lexisnexis.com`.

[10] W. A. Najjar, E. A. Lee, and G. R. Gao. Advances in the dataflow computational model. *Parallel Comput.*, 25(13-14):1907–1929, 1999.

[11] Netezza Performance Server. `http://www.netezza.com`.

[12] M. E. J. Newman. Detecting community structure in networks. *European Physical Journal B*, 38:321–330, May 2004.

[13] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(6):066133, June 2004.

[14] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb. 2004.

[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

[16] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814, 2005.

[17] I. Raicu, I. Foster, and Y. Zhao. Many-task computing for grids and supercomputers. In *Proc. IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)*, 2008.

[18] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle. Active disks for large-scale data processing. *IEEE Computer*, June 2001.

[19] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[20] U.S. National Institutes of Health (NIH). `www.pubmedcentral.nih.gov`.

[21] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and Ümit Çatalyürek. A scalable distributed parallel breadth-first search algorithm on bluegene/l. In *Proceedings of Supercomputing'05*, Nov. 2005.