Computer Science and Engineering 2016, 6(2): 33-38
DOI: 10.5923/j.computer.20160602.03

Page Rank Performance Evaluation of Cluster Computing Frameworks on Cray Urika-GX Supercomputer

Robert W. Techentin[1,*], Matthew W. Markland[1], Ruth J. Poole[1], David R. Holmes III[2], Clifton R. Haider[1], Barry K. Gilbert[1]

[1]Special Purpose Processor Development Group, Department of Physiology and Biomedical Engineering, Mayo Clinic, Rochester, USA
[2]Biomedical Analytics and Computational Engineering Group, Department of Physiology and Biomedical Engineering, Mayo Clinic, Rochester, USA

Abstract Modern "big data" and analytics software platforms offer a variety of algorithms to the analytics practitioner. It is often possible to choose from several available implementations to solve a given set of problems, even on the same set of hardware, within the same software ecosystem, or leveraging completely different software environments. Choices of techniques should be informed by the relative performance and scalability of the implementations, whether or not they run on the same hardware platform. This paper presents a relative performance comparison of three implementations of the popular PageRank graph analytic algorithm running on the Cray Urika-GX high performance analytics appliance. Relative performance and scaling presented here, along with additional information about the problem set and resources available to the analyst, could be used to make an informed decision about which PageRank implementation to choose.

Keywords Graph theory, Computational efficiency, Parallel Algorithms, Analytical Models

1. Introduction

Graph analytics is becoming an increasingly popular approach to realizing value from "big data," particularly when the datasets are naturally sparse and irregular. Clinical data is a classic example of "big data," with the "four V" characteristics of volume, variety, velocity and veracity. Veracity is particularly problematic, since medical records and even clinical study data are often incomplete or recorded in incompatible dialects of different medical subspecialties. Even before the inception of integrated fully-electronic medical records in the last century, there have been significant efforts to glean insights from aggregations of patient data [1, 2], but the sparsity and veracity of the data challenged the best statistical analysis and (later) data mining techniques [3, 4].

Tera Computer began developing a new kind of computer system in the early 1990s specifically intended to address sparse problems. The Multithreaded Architecture (or MTA) was based on custom processors with thousands of execution cores and gigantic shared memory. This new kind of machine excelled at processing data with complex and sparse relationships. Cray Inc. brought the massively multithreaded technology to market in their Cray XMT computer system in the mid-2000s, and developed a follow-on XMT2 system that became available in 2011. The custom Threadstorm processors (each processor is a single custom-designed integrated circuit) had uniform access to memory via the SeaStar supercomputer network. An XMT2 with 8,196 cores and 2 TB of memory (Serial Number 2) was installed at Mayo Clinic in 2011, and has been used for a number of studies, including government sponsored research, health care, and computer network defense [5-7]. Graph analytics, statistical computation, and machine learning have all been applied using this specialized hardware.

However, specialized hardware, although very capable in its domain, cannot take advantage of the surge of new data analytics algorithms and software generated within academia and industry, which is generally developed for networks of commodity computers. Foundational open source software packages such as Hadoop and Spark have grown into software ecosystems, enabling a huge number of software developers to contribute and improve implementations of analytics algorithms. With these open source frameworks, analytics practitioners can choose among algorithms, different implementations of the same algorithm, or even implementations in disparate software ecosystems. The practitioner's choices should, however, be informed by an understanding of the relative performance and scalability of analytics techniques available in these environments.

In this paper we present an evaluation of the performance of the PageRank algorithm, a classic graph analytic, as implemented in three software ecosystems while running on

* Corresponding author:
techentin.robert@mayo.edu (Robert W. Techentin)
Published online at http://journal.sapub.org/computer

the same high-performance cluster computing platform. We measured computation time for large graphs of more than a billion edges, and we report strong scaling, where the graph size is held constant and the number of compute nodes varies. This algorithm evaluation approach is similar to algorithm performance comparisons presented in [4] and database comparisons in [8], which explored algorithm optimization in different environments. However, instead of hand-tuning algorithms for each environment, we simply chose implementations available in the various software ecosystems in the same manner as analytics practitioners would do. Analytics workflows are seldom composed of a single algorithm, and thus time spent tuning the raw performance of any given implementation must be balanced against other efforts, including other algorithms, processing, and even data curation.

In the following sections, we describe the hardware and software platforms and the nature of the graph datasets and algorithms. We compare computation times and scaling for each of the ecosystems, and suggest guidelines for choosing one implementation over another.

2. Computing Platform

The Cray® Urika®-GX Agile Analytics Platform, announced in May 2016, is positioned as a fusion of supercomputing technology and enterprise-class open source data analytics frameworks. The hardware platform inherits high performance computing (HPC) features from Cray's computational cluster supercomputers, with modifications suitable for "big data" enterprise analytics. This is the third generation Cray data analytics platforms, combining graph analysis and "big data" mining technologies from the Urika-GD "Graph Discovery" appliance and the Urika-XA "Extreme Analytics" platform. This new machine supports several different open source and proprietary software ecosystems, with cluster resource management features that allow them to be run in any combination on independent dynamically allocated partitions.

Individual cluster nodes provide substantial computational resources, with up to 32 cores of Intel® Xeon® processors and 256 GB of DRAM. A single rack contains up to 48 compute nodes which are clustered by the Cray Aries™ supercomputing network, providing a bisection bandwidth of 378 GB/s. The computational cluster is augmented by disk storage on every compute node, with both solid state and spinning disk drives providing over 200 TB of local storage distributed throughout the rack.

The Urika-GX provides three software ecosystems for data analytics: Hortonworks Data Platform (HDP™) Hadoop®; the Spark™ Ecosystem; and the proprietary Cray Graph Engine. Hadoop and Spark are open source and widely used in the data analytics community. The Cray Graph Engine is a direct descendent of the Urika-GD "graph discovery" appliance software, providing a semantic graph database using Resource Description Framework (RDF) [9] triples to represent data, and the SPARQL query language [10] and built-in graph algorithms to support discovery and analysis. The ecosystems can share the Hadoop Distributed File system (HDFS) or an attached Lustre® file system. Application productivity tools (e.g., Jupyter Notebooks) are layered on top of analytics programming environments, including Java, Scala, R, and Python.

In addition to three data analytics ecosystems, the system offers computational supercomputer software development environments, including compilers and libraries for multithreaded (e.g., OpenMP), multiprocessor (e.g., MPI) and shared memory (e.g., Partitioned Global Address Space, or PGAS) parallel processing. While these resources can be valuable for data analytics workflows, they were not addressed in this study.

3. Graph Data Generation

Recursive MATrix (R-Mat) graphs [11] have a power law distribution of edges (i.e., a small number of highly connected vertices) and have structure similar to social networks and other real-world datasets. With a small number of parameters, graphs can be synthesized with specific characteristics, matching models such as Erdős-Rényi or Pennock, and of arbitrarily size. The datasets for this evaluation were synthesized with the graph generator from the Graph500 benchmark suite [12] using the default parameters A=0.57, B=0.19, C=0.19, D=0.05 and an edge factor of 16. The generated graphs were undirected and contained duplicate edges. The Graph500 benchmark defines problem size in terms of the "scale" of the graph, where there are 2^{scale} vertices and 16 times as many edges as vertices. A "toy" problem is defined as scale 26 and having approximately one billion edges. A "small" graph, at scale 29, could be stored in as little as 128 GB of memory when edges are represented as pairs of 64-bit integer vertices. Table 1 summarizes the dataset sizes for this study. The scale 24 graphs were used as software test cases, while scale 26 and 29 were employed in the study.

Table 1. Graph Sizes for Page Rank Comparison

Scale	Vertices	Edges
24	16,777,216	268,435,456
26	67,108,864	1,073,741,824
29	536,870,912	8,589,934,592

The binary graph data structures were translated into formats suitable for the three software ecosystems. For Hadoop and Spark, the graph was represented in a text file, one line of text per edge, with tab-separated decimal numbers for the source and destination vertices. The Cray Graph Engine required the RDF "N-triples" format, naming the edges between vertices, so vertices and a single edge name were encoded into compact identifiers, of the form "<urn:37136534> <urn:e> <urn:48299673>." When compared to the 128 GB scale 29 binary file, the translated

files were surprisingly compact, at 154 GB and 339 GB for text and triples files, respectively.

4. Algorithm Selection

Many different graph algorithms have proven useful in data analytics, but there is no general consensus on an essential subset of "the most important" algorithms. Analytics software packages and libraries usually provide several algorithms, often optimized for their particular environments. Instead of choosing a specific algorithm and then implementing, debugging and tuning code to match each of the software ecosystems, we simply selected algorithms from the provided packages. The lists of available graph algorithms for the three ecosystems, presented in Table 2, had surprisingly little overlap, with only two algorithms available in all three frameworks. Because it is well known and widely implemented, we chose the PageRank algorithm [13] to compare the three software ecosystems. It should be noted, however, that although PageRank has been widely implemented, efficient execution using parallel processing or acceleration remains an active research area [14-16].

Table 2. Graph Algorithms Available In Three Software Ecosystems (Highlighted Algorithms Are Available In More Than One Framework)

Hadoop Flink	Spark GraphX	Cray Graph Engine
Adamic-Adar		BadRank
Clustering Coefficient		Betweenness Centrality
Community Detection		
Connected Components	Connected Components	
Jaccard Index		
Label Propagation		Label Propagation
PageRank	PageRank	PageRank
Single Source Shortest Paths		S-T Connectivity
Triangle Counting	Triangle Counting	Triangle Counting

The Hadoop ecosystem does not support graph analytics by default, so we installed the Flink package for distributed streaming and batch processing [17], which includes a graph API called "Gelly." This library provides a graph representation, data structures and several graph algorithms. Flink runs on the Hadoop YARN resource manager and works with the Hadoop Distributed File System (HDFS) and other open source data processing ecosystems. However, Flink provides its own processing model and memory management, and does not utilize the MapReduce computation approach typically associated with Hadoop data processing. Instead, PageRank is implemented as scatter/gather operations in Flink, with vertices alternately sending update messages to their neighbors through their outgoing edges and collecting updates from their neighbors via their incoming edges.

For the Spark ecosystem, we selected the PageRank algorithm implemented in the GraphX library [18], which was already available on the system. Spark generally claims to run programs 100X faster than Hadoop MapReduce in memory. However, since neither Flink nor GraphX rely upon MapReduce, it was not clear what sort of a performance difference we should expect from these libraries.

The Cray Graph Engine includes several built-in graph algorithms, including PageRank. CGE is proprietary software, written using co-array C++ and the Partitioned Global Address Space (PGAS) programming model [19]. CGE can query or run graph algorithms on in-memory semantic graphs, taking full advantage of the processors, memory, and Aries supercomputer backplane of the Urika-GX machine.

5. Run Time Results

PageRank was run on each size graph, using 4, 8, 16, and 32 compute nodes to evaluate strong scaling of each software ecosystem. Only minimal performance tuning was conducted, using command line options: the algorithm implementations themselves were not modified. Tuning parameters depend on the ecosystem features. The Hadoop Flink PageRank was run with 32 cores per compute node, while CGE was launched with 16 images per node, which is roughly equivalent to cores. Spark GraphX parameters, on the other hand, were oriented towards the problem space and memory, specifying 256 edge list partitions and 192 GB memory per executor.

Flink does not have tuning parameters itself, but relies on characteristics set by the YARN resource manager. The YARN configuration on the Urika-GX limited the number of tasks per task manager to 8, so in order to take advantage of 32 cores per compute node, it was necessary to use 4 task managers per node with 8 tasks per manager. For larger graphs, running multiple task managers failed with insufficient memory, so it was necessary to reduce the number of task managers to one per node.

Spark GraphX tuning parameters are extensive and complex. For this study, we focused on balancing data partition counts and memory allocation across executors. GraphX launches one executor on each compute node. We initially allocated 192 GB of memory for each executor, leaving some for operating system functions. Empirical tests with the scale 29 graph data and 32 compute nodes indicated that 29 data partitions per compute node provided optimal performance. Each data partition is processed by one executor thread, so this configuration utilized 29 cores for data processing and left 3 cores for operating system functions. However, the overhead of running 29 data partitions required reducing memory allocation to 128 GB.

CGE has only two tuning parameters available to the command line database launcher: the number of compute nodes and the number of images per node. Images are roughly equivalent to cores. The shared memory

programming model of co-array C++ allows many processes in a compute cluster to share a partitioned global address space. Each process is aware of its local memory partition, and remotely accesses the global partitions when necessary. Remote memory accesses take advantage of the Cray Aries interconnect features such as fine granularity and atomic operations. Varying the number of images per node changes the balance between the local and global memory partitions, and can affect performance. For this study we fixed the number of images at 16 as recommended by Cray as a starting point for reasonable performance.

Run times for the scale 26 graph are presented in Figure 1. With four compute nodes, CGE was only slightly faster than Spark, and Hadoop was 10X slower than both of them. For larger numbers of compute nodes, Hadoop performance improved substantially, while Spark performance actually deteriorated. CGE was consistently 5X faster than the others.

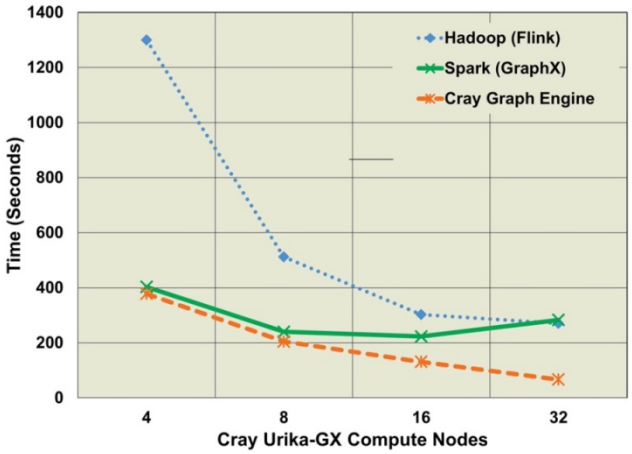

Figure 1. PageRank Run Time Comparison for Scale 26 Graph on Urika-GX shows strong scaling (varying number of compute nodes) for 100 Iterations of Hadoop Flink, convergence of Spark GraphX, and convergence of Cray Graph Engine (45422)

For the scale 29 graph, which is 8X larger than scale 26, none of the PageRank implementations would run on four nodes, despite the availability of an aggregated 1 TB of physical memory. When the algorithms were able to run on larger numbers of compute nodes, PageRank took 10X longer to complete than for scale 26, with the same performance trends as depicted in Figure 1. Actual run times in Table 3 show that CGE was more than5-10X faster than Hadoop Flink and Spark GraphX.

Table 3. PageRank Run Time (in seconds) Comparison for Scale 29 Graph

Implementation	8 Nodes	16 Nodes	32 Nodes
Hadoop (Flink)	8,760	4,769	6,199
Spark (GraphX)			2,640
Cray Graph Engine		1,082	500

6. Discussion

PageRank run time trends were similar to our expectations, with Hadoop yielding the lowest performance and CGE providing the fastest solution. The poor scaling for Spark GraphX was somewhat surprising, since we expected each implementation to perform better with more processors and memory. This result may be due to the relative immaturity of the GraphX software, or it might indicate a need for additional performance tuning, perhaps related to graph partitioning. To the best of our knowledge, these software ecosystems have not been rigorously tested against datasets this large, and we surmise that good performance may require both tuning and modifications to the implementation.

Run times for both scale 26 and 29, exemplified by Figure 1, show strong scaling for both Hadoop Flink and the Cray Graph Engine implementations of PageRank. Spark GraphX scaled poorly for this example. The clear winner in raw computation time and scaling is CGE, which can complete 5X faster than the other implementations. As a native C++ application, CGE takes full advantage of the Cray Aries supercomputer interconnect, which includes low-level primitives that accommodate the inevitable latencies that plague clustered computer systems. The Aries network features and the shared memory model may also account for the superior scaling of CGE. It must be noted, however, that only minimal tuning was performed for this study, as would be typical of business environments where costs and schedules often constrain the analytics practitioner to simply achieving an expedient answer. Academics may further tune open source implementations or develop new algorithms, but those improvements would not be generally accessible for some time.

Run times, however, do not tell the entire story. PageRank is an iterative algorithm. Both Spark and CGE run the computation iteratively until a convergence criterion is achieved. Hadoop, on the other hand, runs for a specified number of iterations regardless of convergence. We were able to compare Spark and Hadoop running 100 iterations each, and observed that Hadoop's Flink implementation was actually 3X-5X faster than Spark's GraphX for the same number of iterations. An in-depth analysis would be required to determine convergence, since an equal number of iterations might not produce equivalent results from the two implementations.

Raw computation time of the PageRank algorithm would very likely not be the only issue driving the practitioner's choice of implementation. Run time for PageRank (or any algorithm) may be only one small part of a complex analytical workflow that includes data curation, format translations, statistical analysis of algorithm output, data exploration, and presentation of results. For this specific use case, running PageRank on a scale 29 graph, the run time difference of 10 versus 100 minutes may be sufficient motivation to integrate CGE's PageRank into an established Hadoop workflow that includes data curation, post-PageRank statistics and graphical presentation of results.

There are drawbacks to each of these PageRank implementations which must be considered by analytics practitioners. The Spark execution environment offers a

large number of tuning parameters, and individual algorithms may have additional controls. The wide variety of tuning parameters can enable performance improvements, but the complexity of tuning and the interrelationship between parameters can make optimal performance elusive.

Another performance consideration for both Hadoop and Spark is that they rely upon Java Virtual Machines, which can introduce their own complexities into optimization. For example, changing memory allocations or data partitions to optimize one set of parameters may change the amount of data shuffling and even influence garbage collection behavior of the underlying JVM. Changing the problem size or hardware allocation may invalidate optimizations found for the previous run.

In addition to performance considerations, there are practical issues of using the different environments. On the Cray Urika-GX, Hadoop job resources are managed by YARN and require the user to allocate and de-allocate resources manually. Spark and CGE will dynamically allocate resources and release them when the job completes. Data file formats can also be an issue that affects the analysis effort. Both Hadoop Flink and Spark GraphX use a simple integer edge list to describe a directed graph. CGE, however, requires data in RDF format, and even when the analysis requires only a simple directed graph, it is necessary to translate the data with the correct grammar.

7. Conclusions

The Cray Urika-GX system provides several useful software ecosystems for performing analytics on large and complex datasets, including Hadoop, Spark, and the Cray Graph Engine. A particular algorithm may be uniquely implemented take advantage of characteristics of the software ecosystem, offering substantially different performance characteristics, even when running on the same hardware. If there is a choice of ecosystems, or if the various ecosystems can be integrated into a hybrid application, comparison and validation of the implementations is appropriate.

We evaluated the performance of the PageRank algorithm implemented in the Hadoop Flink package, the Spark GraphX library, and the Cray Graph Engine using synthetic datasets from the Graph500 benchmark. We presented strong scaling results for run times on graphs of up to 8 billion edges, and showed that performance could vary by more than 10X between the implementations.

From these results, we concluded that analytics practitioners may choose implementations of various algorithms from different software ecosystems, evaluate their performance on relevant problem sets, and determine if performance differences warrant integration of the specific implementations into their analytics workflows. While graph algorithm research and optimization is still an active area of research, practitioners should take advantage of readily available implementations to improve their workflows.

ACKNOWLEDGEMENTS

The authors thank T. Funk and S. Neumann for artwork and manuscript preparation. This work was funded by DARPA's Microsystems Technology Office.

REFERENCES

[1] P. C. Carpenter, "The electronic medical record: perspective from Mayo Clinic," *International journal of bio-medical computing,* vol. 34, pp. 159-171, 1994.

[2] C. G. Chute, D. Crowson, and J. Buntrock, "Medical information retrieval and WWW browsers at Mayo," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*, 1995, p. 903.

[3] T. Botsis, G. Hartvigsen, F. Chen, and C. Weng, "Secondary use of EHR: data quality issues and informatics opportunities," *AMIA Summits Transl Sci Proc,* vol. 2010, pp. 1-5, 2010.

[4] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health Information Science and Systems,* vol. 2, p. 1, 2014.

[5] R. Techentin, D. Foti, S. Al-Saffar, P. Li, E. Daniel, B. Gilbert, and D. Holmes, "Development of a Semi-Synthetic Dataset as a Testbed for Big-Data Semantic Analytics," presented at the IEEE International Conference on Semantic Computing, Newport Beach, CA, 2014.

[6] D. Ediger, K. Jiang, J. Riedy, D. A. Bader, and C. Corley, "Massive social network analysis: Mining twitter for social good," in *2010 39th International Conference on Parallel Processing*, 2010, pp. 583-593.

[7] R. Techentin, J. S. Sauver, J. Huddleston, B. Gilbert, and D. Holmes, "Lessons learned from the semantic translation of healthcare data," in *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on*, 2014, pp. 513-518.

[8] R. C. McColl, D. Ediger, J. Poovey, D. Campbell, and D. A. Bader, "A performance evaluation of open source graph databases," in *Proceedings of the first workshop on Parallel programming for analytics applications*, 2014, pp. 11-18.

[9] Resource Description Framework (RDF), ed. https://www.w3.org/RDF/.

[10] SPARQL 1.1 Query Language, ed. https://www.w3.org/TR/sparql11-query/.

[11] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A Recursive Model for Graph Mining," in *SDM*, 2004, pp. 442-446.

[12] The Graph 500 List, ed. http://graph500.org/.

[13] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: bringing order to the web," 1999.

[14] A. Cevahir, C. Aykanat, A. Turk, B. B. Cambazoglu, A. Nukada, and S. Matsuoka, "Efficient PageRank on GPU clusters," *IPSJ SIG Notes,* pp. 1-6, 2010.

[15] S. Sangamuang, P. Boonma, and L. L. W. Kyii, "An

algorithm to improve MPI-PageRank performance by reducing synchronization time," in *2015 International Computer Science and Engineering Conference (ICSEC)*, 2015, pp. 1-4.

[16] Y.-J. Xie and C.-F. Ma, "A relaxed two-step splitting iteration method for computing PageRank," *Computational and Applied Mathematics,* pp. 1-13, 2016.

[17] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Data Engineering,* p. 28, 2015.

[18] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 599-613.

[19] K. Maschhoff, R. Vesse, and J. Maltby, "Porting the Urika-GD graph analytic database to the XC30/40 platform," in *Cray User Group Conference (CUG'15), Chicago, IL*, 2015.