HHS Public Access

Author manuscript

Proc IEEE Int Conf Big Data. Author manuscript; available in PMC 2015 April 07.

Published in final edited form as:

Proc IEEE Int Conf Big Data. 2014 October ; 2014: 29–35. doi:10.1109/BigData.2014.7004353.

Towards Scalable Graph Computation on Mobile Devices

Yiqi Chen,
College of Computing, Georgia Tech, Atlanta, GA, USA

Zhiyuan Lin,
College of Computing, Georgia Tech, Atlanta, GA, USA

Robert Pienta,
College of Computing, Georgia Tech, Atlanta, GA, USA

Minsuk Kahng, and
College of Computing, Georgia Tech, Atlanta, GA, USA

Duen Horng Chau
College of Computing, Georgia Tech, Atlanta, GA, USA

Yiqi Chen: yiqic1993@gatech.edu; Zhiyuan Lin: zlin48@gatech.edu; Robert Pienta: pientars@gatech.edu; Minsuk Kahng: kahng@gatech.edu; Duen Horng Chau: polo@gatech.edu

Abstract

Mobile devices have become increasingly central to our everyday activities, due to their portability, multi-touch capabilities, and ever-improving computational power. Such attractive features have spurred research interest in leveraging mobile devices for computation. We explore a novel approach that aims to use a *single* mobile device to perform scalable graph computation on large graphs that do not fit in the device's limited main memory, opening up the possibility of performing on-device analysis of large datasets, without relying on the cloud. Based on the familiar *memory mapping* capability provided by today's mobile operating systems, our approach to scale up computation is powerful and intentionally kept simple to maximize its applicability across the iOS and Android platforms. Our experiments demonstrate that an iPad mini can perform fast computation on large real graphs with as many as *272 million* edges (Google+ social graph), at a speed that is only a few times slower than a 13″ Macbook Pro. Through creating a real world iOS app with this technique, we demonstrate the strong potential application for scalable graph computation on a single mobile device using our approach.

Keywords

graph mining; scalable algorithms; mobile device; memory mapping

I. Introduction

Mobile devices like smart phones and tablets have become increasingly important in our daily lives. Their portability and ever-increasing computational power have attracted much attention from computer science researchers, spurring recent research in leveraging mobile

devices for computation and analytics. For example, a recent work called *Kinetica* [1] explores how to leverage the unique multi-touch interaction capabilities that an iPad provides to allow users to perform data analysis more naturally. Motivated by the impressive hardware of today's mobile devices[1], researchers have started to investigate using mobile devices for intensive computation. For example, some research connected multiple devices together for mobile cloud computing [2], [3], through the help of dedicated servers and fast internet connections.

A. Scalable Computation on a Single Mobile Device

Intrigued by today's mobile device computational power, we wonder how much computation a *single* mobile device can perform, at a reasonable speed. Specifically, we ask the following research questions:

- As mobile devices often have limited main memory, is there a limit on the data size that they can process?
- If there is such a limit, how can we overcome that? What methods to use? Ideally we want methods that are simple to understand and implement, but powerful enough that the computation speed will not suffer.
- How fast can mobile devices perform computation? For example, how do they compare to laptops or desktop computers?

To investigate these research questions, we decided to focus on graph computation, as graph datasets are prevalent in many domains, with compelling applications in intelligence and consumer applications.

1) Military intelligence applications—A recent DARPA project, called GUARD DOG [4], aimed to enable soldiers deployed in the field, without wireless or cellular access, to perform in-situ graph analysis on their mobile devices (e.g., Personal Digital Assistant devices). The soldiers would need to perform analysis using their devices when they were on patrol. Then, at the end of a mission, they would return to base and upload their new data and analysis results to a server.

2) Consumer applications—In the consumer space, there are engaging applications such as the best-selling Discovr Movie app [5], which recommends related movies to the user through interactive graph visualization, and allows the user to incrementally reveal larger part of a similar-movie graph. The app currently relies on the cloud to perform graph computation to determine which part of the graph to show to the user.

The two examples above illustrate just a couple of use cases that would significantly benefit from fast scalable graph computation performed directly on the device itself.

[1]Notably, the computational power of today's devices is often comparable to that of desktop machines from just a few years ago. For example, the Samsung Galaxy S5 is equipped with 2 GB RAM, which was the mainstream RAM size of laptop computers a few years ago.

As we will explain in Section II, our main idea to enable scalable graph computation is to leverage the *memory mapping* technique [6], [7] to fit a large graph's edges into the virtual memory of the operating system (see Figure 1). This way, the graph size is no longer constrained by the main memory available on the mobile device, which is typically no more than a few gigabytes. Memory mapping is a common and familiar capability provided by all modern operating systems. We intentionally leverage it so that we can keep our approach simple, robust, and powerful, as our experiments in Section III demonstrate. Using a simple, ubiquitous approach also allows us to generalize it to both iOS and Android with minimal modifications (see Section III).

To demonstrate the potential of our approach in the real world, we developed an iOS app called *GraphRec* (see Figure 2) that can perform real-time personalized recommendation by leveraging a large graph of related movies, collected from the *Rotten Tomatoes* movie review website [8]. This graph contains hundreds of thousands of movies (nodes). An edge in the graph connects two movies that Rotten Tomatoes users have voted to be similar. For example, the two *James Bond* movies *Casino Royale* and *Quantum of Solace* are connected. Unlike the DiscovrMovie app we previously mentioned, all computation in *GraphRec* is done on the iPad itself, without needing the cloud, thanks to the *memory mapping* technique. Furthermore, *GraphRec* is able to perform personalized recommendation by considering multiple seed movies, and suggest movies that combine multiple genres. For example, in Figure 2, our user has indicated an interest in a horror film *Evil Dead 2* and a comedy film *Hot Fuzz*. By using the personalized PageRank graph algorithm [9], *GraphRec* recommends some movies that are intersection of those two genres, e.g., *Shaun of the Dead*. In Section IV, we will describe the algorithm and data that we used in this application in greater detail.

B. Our Contributions

In this work, we answer the research questions that we raised, through the following contributions:

- **The First Study of Million-scale Computation on Mobile Devices.** To the best of our knowledge, this is the first work that studies how to scale up computation on mobile devices, for large-scale graph datasets that do not fit in the device's main memory. We believe our work opens up the possibility of performing on-device analysis of large datasets, without relying on the cloud.
- **Simple and Scalable Approach based on Memory Mapping.** We provide the critical insight that memory mapping is a feasible and enabling technique that allows mobile devices to perform computation at a speed that is only a few times slower than that of a laptop computer (see Figure 3).
- **Large Scale Evaluation.** Through experiments with large, real-world graphs with up to *272 million* edges (Google+ social graph), we demonstrate that our approach can handle a much larger scale that previous work has not explored.
- **An App to Demonstrate Memory Mapping's Benefits for Mobile** Use. Through creating a real world iOS app with the *memory mapping* approach (see Figure 2),

we demonstrate the strong potential application for scalable graph computation on a single mobile device using our approach.

II. Our Approach: Scaling up via Memory Mapping

A. Background: Memory Mapping

Memory mapping is a well-known technique that has been around since the advent of operating systems (OS). It enables the creation of virtual memory so the OS can provide virtual address spaces that exceed the capability of the main memory. With a large virtual address space applications do not need to compete for or manage shared memory space. Below, we give a brief overview of memory mapping and highlight some of its key features that we leverage to scale up graph computation.

The main feature of memory mapping is to allow the OS to map a file on disk to a process' virtual memory, whose size can be much larger than that of device's physical memory. The OS divides the file into pages, and loads them into physical memory if those pages are accessed. If physical memory is full, the OS will determine which pages are more frequently accessed and keep those active pages in physically memory. This way the OS unloads the relatively inactive pages out of physical memory. The mechanism of memory mapping is shown in Figure 1. For more details, please refer to [6], [7] and [10].

We summarize the following benefits of memory mapping:

- Fast I/O Operations: The mechanism provided by memory mapping is able to determine frequent access pages of the file and keep those pages in memory. This low-level optimization technique provided by OS results in fast I/O operations.
- Less Overhead: Since the OS does most of the low-level optimization work, there is less strain on developers and algorithm users. With memory mapping, we are able to access a file as if the entire file is in physical memory. This results in much less overhead to developers and much simpler code.

B. Our Approach: Memory-map edge list file for fast computation

We represent a graph as an edge list text file, with each row in the file is an edge, identified by its source node ID and target node ID, both stored as integers. Then, we convert the edge list file into its binary representation. In the binary format, each row (edge) in the original text file is converted into two 4-byte integers. Working with binary files is in general faster than working with the raw text files, which requires parsing and converting the text. Therefore, this text-to-binary conversion is commonly employed by many approaches [11], [12]. Furthermore, for a graph that has a large number of nodes and edges, its binary edge list file size can be significantly smaller than its original text-based file (e.g., can be up to 10 times smaller).

A developer who wants to run an algorithm over the graph (i.e., its disk-based binary edge list) can access the edges as if they were stored in an in-memory array.

C. Our On-going Work for the PC

In our recent on-going work [10], [13], we are exploring how to scale up computation on a PC using memory mapping. We did not consider the mobile setting, and it was also unclear if a similar approach would work on mobile devices. Our best guess was that even with special performance tuning, mobile devices would perform significantly worse. To our surprise, as we will show next, an iPad mini's performance is on par with a 13″ Macbook Pro.

III. Experiments

A. Goal and Overview

We measure the running time of two classic graph computation algorithms, connected components and PageRank, on an iPad mini and a 13″ Macbook Pro, and compare their results. Since we are comparing performance of a mobile device to that of a computer, results on the Macbook Pro are very likely to be much better than those on the iPad mini. We intend to investigate the amount of performance drop when moving from a laptop to a mobile device.

First we will describe the graph datasets used and our experimental setup. Then we will discuss the results.

B. Graph Datasets and Experiment Setup

1) Test devices—Our primary experiment device is an iPad mini with Retina display (WiFi-only), with Apple A7 chip with 64-bit architecture, 1 GB RAM and 32 GB disk space, running iOS 7.1. The laptop computer used is a 13.3″ Macbook Pro with Retina display, with 2.8 GHz dual-core Intel Core i7 processor, 16 GB RAM and 512 GB SSD, running OS X 10.9.2.

2) Test algorithms and configuration—There are many possible graph algorithms that we may use to evaluate our approach. However, as Kang et al. pointed out [14], many graph algorithms are composed of iterative matrix-vector multiplications at their core (e.g., computing graph radius, eigensolver, PageRank, connected components, etc.). Therefore, among all these possibilities, we chose two representative, classic graph algorithms: (1) weakly *connected components* and (2) *PageRank*, and focused our evaluation on them. For more details of these two and other graph algorithms, please refer to [14].

Our test program was written in Objective-C and compiled using Xcode 5.1. All tests are conducted with the device's Wi-Fi and bluetooth turned off. No other user applications were running, other than our test program. For each graph dataset, we ran our test program 3 times and report their average runtime. Before each test, the page caches are cleared by restarting the device.

3) Datasets—To better understand how our approach performs under various conditions, we carefully chose four graph datasets with different scales. Their statistics are shown in Table I. Two of the four graphs have binary edge list files that are smaller than the iPad's physical memory (1 GB): a Pokec network [15] with 31 million edges (245 MB) and a

LiveJournal network [16] with 69 million edges (552 MB). The other two graphs have binary edge list files that are close to or larger than the iPad's main memory size: an Orkut network [17] with 117 million edges (938 MB) and a snapshot of Google+ social structure [18] with 272 million edges (2.2 GB, crawled in August 2011).

4) Monitoring memory usage—We use the VM Tracker for iOS to analyze the behavior of the iPad mini during the algorithm execution. VM tracker is a program profiling tool within Xcode that allows us to keep track of physical and virtual memory activity of the iPad mini at run time.

C. Test Results

Figure 3a and 3b compare the speed of connected components and PageRank algorithm on the iPad mini and the Macbook Pro, and Figure 3c shows the linear scalability of memory mapping. We will discuss these results in detail next.

1) Results on Pokec and LiveJournal graph—As shown in Figure 3, for both the Pokec (30M edges) and LiveJournal (69M edges) graph, the runtimes of both algorithms on the Macbook Pro is around 3× faster than that on the iPad mini (e.g., 0.7s vs 2.97s and 1.75s vs 6.31s for 5 iterations of PageRank). Given that the Macbook Pro is equipped with a more powerful processor and more RAM (16 GB vs 1 GB) than the iPad mini, we were surprised by such small speed differences between the two devices.

Note that Pokec and LiveJournal are relatively small graphs, at 245 MB and 552 MB respectively. Through the VM Tracker, we observed that the entire graph file was fully mapped into the physical memory during the algorithm execution.

2) Results on Orkut graph—Figure 3b shows that for the Orkut (117M edges) graph, its runtime of PageRank on the Macbook Pro is 4.4× faster than that on the iPad mini (2.73s vs 14.7s), more than the runtime difference on Pokec and LiveJournal graph (3×). Through the VM Tracker, we observed that during the algorithm execution on the iPad mini, only around 77% of the graph file was mapped into the physical memory at a time. It is because the graph file of Orkut is around 950 MB, while the physical memory size is only 1 GB, and some portion of the main memory is reserved by the operating system. Therefore, on the iPad mini, some pages of the graph file, which are in the virtual memory, may be mapped into the physical memory, or unmapped as needed. However, on the Macbook Pro, where the physical memory size is 16 GB, the entire graph file can be fully mapped into the physical memory, explaining the higher speed comparison ratio when compared against the iPad mini.

3) Results on Gplus graph—While we first ran the algorithms on the Google+ graph (272M edges) on the iPad mini, we unexpectedly got an *out of memory* error when memory mapping the graph into the device's virtual memory. After closely examining both the physical and virtual memory activities through the VM tracker during graph loading, and consulting a Mac Developer Library article [19], we learned that the memory address space of virtual memory for each process on the iPad is limited to 2 GB, and an out of memory error occurs when all of the space is used up by memory mapped file.

This unexpected limitation unfortunately creates a “ceiling” for the binary graph file size when leveraging memory mapping for graph computation. To address this issue, we devise a simple *SelectiveMMap* approach, which shares the same idea of round-robin scheduling. We divide the graph file into two intervals and only map one of them into the virtual memory at a time. During each iteration of graph computation, we only process the currently loaded interval, and then unmap the current interval and map the other one. By leveraging this *SelectiveMMap* approach, we can handle graphs that are larger than 2 GB, easily overcoming the size limit.

However, using *SelectiveMMap* requires us to write a little more code than we originally intended. As mobile devices become more powerful through hardware advances, we expect this approach will no longer be needed.

D. Generalizing to the Android Platform

We also experimented with the Android platform. We were able to test our approach on this popular mobile operating system without any algorithm modifications. We used a Nexus 4 phone, which is a two-year-old hardware (released in October 2012). The runtime on this phone was, unsurprisingly, about 15 times slower than that of the iPad mini. This significant difference is very likely due to the aging hardware. We expect today's Android hardware to demonstrate much higher speed.

IV. Graphrec: An App Using Memory Mapping for Real-Time Movie Recommednation

We created an iOS app, called *GraphRec* (see Figure 2), to demonstrate how fast and scalable graph computation with our *memory mapping* approach can be applied in the real world.

As described in Section I, *GraphRec* is able to perform real-time personalized recommendation by leveraging a large graph of related movies. Unlike the existing iOS app Dis-covrMovie, *GraphRec* is able to give recommendations based on multiple movies that our user chooses. Using the *memory mapping* technique, the interactive recommendation can be done on demand without the cloud. Note that we expect media like movie trailers, movie poster images, etc., could still be served from a web server, which do not require computation, while the expensive computation is on the iPad itself without the cloud. To speed up such media's retrieval, we could use caching, etc., which is outside the scope of this work.

Our app uses a similar-move graph collected from Rotten Tomatoes [8], which is a popular movie information and review website. The graph contains 197,787 movies (nodes) and 184,805 edges. An edge in the graph connects two movies that Rotten Tomatoes users have voted to be similar. For example, the two *James Bond* movies *Casino Royale* and *Quantum of Solace* are connected. The graph is converted into a binary edge list file and memory mapped into device's virtual memory when the application started, as described in Section II-B. Also, a dictionary file containing movie titles and URLs to movie posters is also loaded when the app is just launched.

When the app is launched, our user has the option to specify movies that are of interest (e.g., movies that they have already watched). In this demo, we have pre-selected a horror film *Evil Dead 2* and a comedy film *Hot Fuzz* (see Figure 2). Our user can then select movies for which they want recommendations by tapping the movie posters, highlighting them in orange. In our case, both seed movies are selected. With a press of the "Recommend" button, the app will recommend movies that are relevant to the selected movies, and will add those newly recommended movies to the screen (see Figure 2). The edges connecting the recommendations show how they are connected to the seed movies. Users can iteratively follow the same procedure to explore more movies that they may like. All computation is done without the cloud thanks to the *memory mapping* technique.

The graph computation algorithm we use to generate recommendations is the well-known personalized PageRank graph algorithm [9], which computes the steady-state probability of a *walker* ending up on each node after iterations of random walk. It has been widely used to perform personalized recommendations. In each iteration, the walker randomly picks a neighbor to visit, with a certain probability that he would go back to one of the starting seed nodes. The end result is that nodes (movies) that are in *close proximity* to the seed movies would have higher personalized PageRank scores (e.g., few hops away, and have many paths to the seed nodes). The algorithm is almost identical to the PageRank algorithm, thus it can easily take advantage of *memory mapping*.

When our user asks for recommendations, the app runs personalized PageRank for 5 iterations and displays nodes with highest personalized PageRank results as recommendations. The algorithm running time is 34 milliseconds on average, which does not affect the smoothness of the application.

Empowered by *memory mapping*, the *GraphRec* iOS app is able to give real-time movie recommendations without the cloud. It demonstrates the strong potential usage for scalable graph computation on a single mobile device. Our next step is to generalize this app, so that it can give recommendations based on not only the Rotten Tomatoes graph, but also other graphs such as knowledge graphs and citation networks.

V. Related Work

In recent years, the mobile computing research community has started to investigate mobile cloud computing [2], [3], where mobile devices are connected to remote data center, to perform large-scale computation. These systems often relies on a fast internet connection and dedicated servers to operate.

Relatedly, there is increasingly interest in low-power computing (e.g., using ARM-based architectures), which naturally leads researchers to study mobile devices as they often use low-power chipsets (e.g., Apple's iOS devices use ARM-based processors). The architecture of Tibidabo [20], the first large-scale HPC cluster built using ARM multicore chips, aims to achieve HPC capacity while increasing its energy efficiency.

In the field of machine learning and data mining, recent single-machine graph computation techniques have received a lot of interest. GraphChi [11] is a disk-based parallel graph

computation engine that aims to maximize the power of a single machine and achieve comparable performance to distributed graph computation engines that use parallel computation techniques. TurboGraph [12] improves on GraphChi to deliver even higher speed for their algorithms. As both techniques require sophisticated data structure and careful usage of physical memory, it is unclear if the same approaches may work on mobile devices.

VI. Conclusions and Future Work

To the best of our knowledge, this work is the first foray into performing million-scale graph computation on a *single* mobile device. Thanks to the impressive computation power of today's mobile devices, our investigation shows that an iPad mini can perform fast computation on large real graphs with as many as *272 million* edges, at a speed that is comparable to a 13″ Macbook Pro. Based on the familiar memory mapping capability provided by today's operating systems, our approach to scale up computation is intentionally kept simple, yet powerful. Our work demonstrates that it is a straightforward and empowering technique that allows us to handle graphs that do not fit in a portable device's limited main memory.

For the road ahead, we plan to (1) try our approach on more graph algorithms and on even larger graphs; (2) study other more general data mining and machine learning algorithms and investigate if our approach can help increase their scalability and speed in ways that it does for graph algorithms; (3) investigate how our approach may extend to distributed settings, e.g., in mobile cloud computing.

References

1. Rzeszotarski, JM.; Kittur, A. Proceedings of the 32nd annual ACM conference on Human factors in computing systems. ACM; 2014. Kinetica: naturalistic multi-touch data visualization; p. 897-906.
2. Marinelli, EE. Masters Thesis. Carnegie Mellon University; 2009. Hyrax: cloud computing on mobile devices using MapReduce.
3. Fernando N, Loke SW, Rahayu W. Mobile cloud computing: a survey. Future Generation Computer Systems. 2013; 29(1):84–106.
4. DARPA. [Accessed: 2014-04-30] DARPA-BAA-10-50: Graph Understanding and Analysis for Rapid Detection - Deployed on the Ground (GUARD DOG). [Online]. Available: https://www.fbo.gov/spg/ODA/DARPA/CMO/DARPA-BAA-10-50/listing.html
5. iTunes. [Accessed: 2014-04-30] Discovr Movies - discover new movies. [Online]. Available: https://itunes.apple.com/us/app/discovr-movies-discover-new/id488025044
6. MSDN. [Accessed: 2014-04-30] Memory-Mapped Files. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd997372.aspx
7. Tevanian, A.; Rashid, RF.; Young, M.; Golub, DB.; Thompson, MR.; Bolosky, WJ.; Sanzi, R. USENIX Summer. Citeseer; 1987. A UNIX Interface for Shared Memory and Memory Mapped Files Under Mach; p. 53-68.
8. Tomatoes, Rotten. [Accessed: 2014-08-29] [Online]. Available: http://www.rottentomatoes.com
9. Tong, H.; Faloutsos, C.; Pan, JY. ICDM'06: Proceedings of the 6th IEEE International Conference on Data Mining. IEEE; 2006. Fast random walk with restart and its applications; p. 613-622.
10. Lin Z, Chau DHP, Kang U. Leveraging memory mapping for fast and scalable graph computation on a PC. IEEE BigData'13 Scalable Machine Learning: Theory and Applications Workshop. 2013

11. Kyrola A, Blelloch G, Guestrin C. GraphChi: Large-scale graph computation on just a PC. Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2012:31–46.
12. Han, WS.; Lee, S.; Park, K.; Lee, JH.; Kim, MS.; Kim, J.; Yu, H. Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2013. TurboGraph: a fast parallel graph engine handling billion-scale graphs in a single PC; p. 77-85.
13. S, KMD.; C, H.; L, H.; K, U.; Lin, Zhiyuan; Kahng, Minsuk. Big Data, 2014 IEEE International Conference on. IEEE; 2014. MMap: Fast Billion-Scale Graph Computation on a PC via Memory Mapping.
14. Kang, U.; Tsourakakis, CE.; Faloutsos, C. Data Mining, 2009 ICDM'09 Ninth IEEE International Conference on. IEEE; 2009. Pegasus: A peta-scale graph mining system implementation and observations; p. 229-238.
15. Takac L, Zabovsky M. Data analysis in public social networks. Intl Scientific Conf & Intl Workshop Present Day Trends of Innovations. 2012
16. Backstrom, L.; Huttenlocher, D.; Kleinberg, J.; Lan, X. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2006. Group formation in large social networks: membership, growth, and evolution; p. 44-54.
17. Mislove, A.; Marcon, M.; Gummadi, KP.; Druschel, P.; Bhattacharjee, B. Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. ACM; 2007. Measurement and analysis of online social networks; p. 29-42.
18. Gong, NZ.; Xu, W.; Huang, L.; Mittal, P.; Stefanov, E.; Sekar, V.; Song, D. Proceedings of the 2012 ACM conference on Internet measurement conference. ACM; 2012. Evolution of social-attribute networks: measurements, modeling, and implications using google+; p. 131-144.
19. Apple. [Accessed: 2014-04-30] About the Virtual Memory System. [Online]. Available: https://developer.apple.com/library/mac/documentation/performance/conceptual/managingmemory/articles/aboutmemory.html
20. Rajovic N, Rico A, Puzovic N, Adeniyi-Jones C, Ramirez A. Tibidabo: Making the case for an ARM-based HPC system. Future Generation Computer Systems. 2013

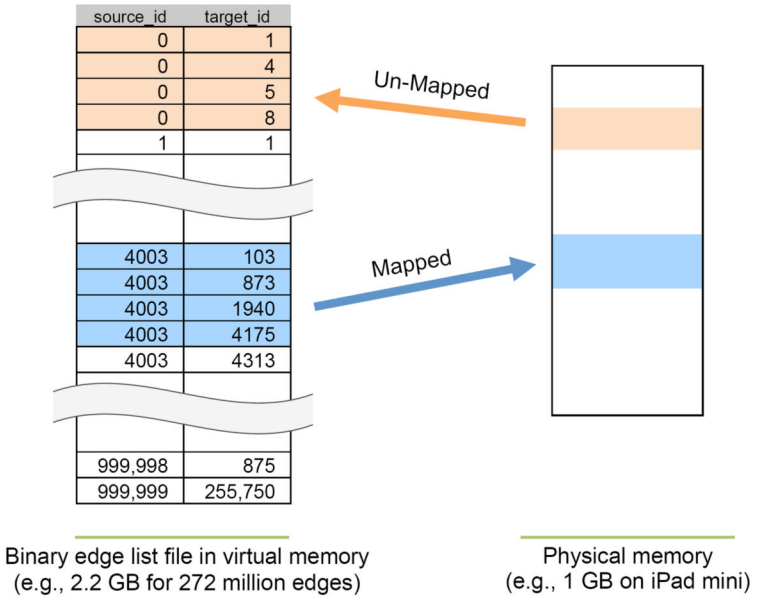


Fig. 1.

Mechanism of memory mapping. The entire file is mapped to the virtual memory (on the left), whose size can be much greater than that of device's physical memory (on the right). During the algorithm execution, different pages of the file are selectively loaded into the physical memory by OS, depending on which pages are frequently accessed. This way, OS will handle most of the memory management optimization.

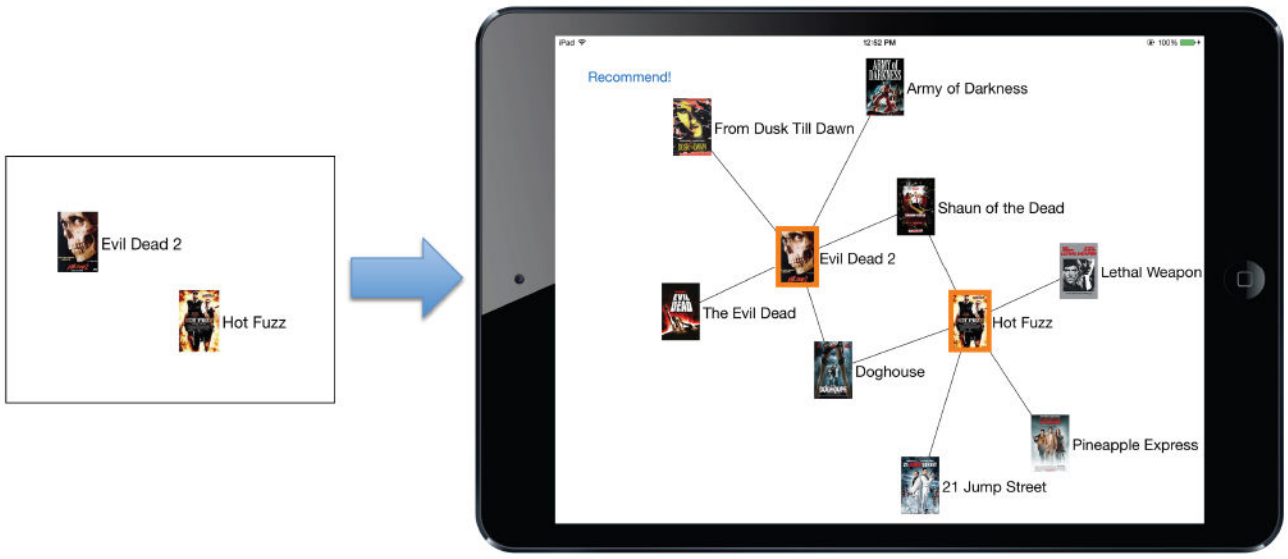
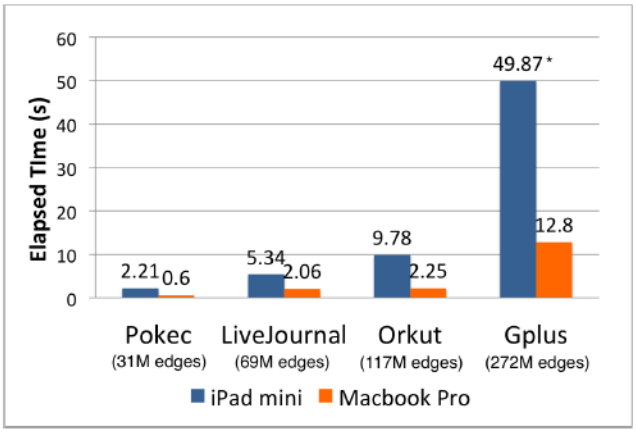
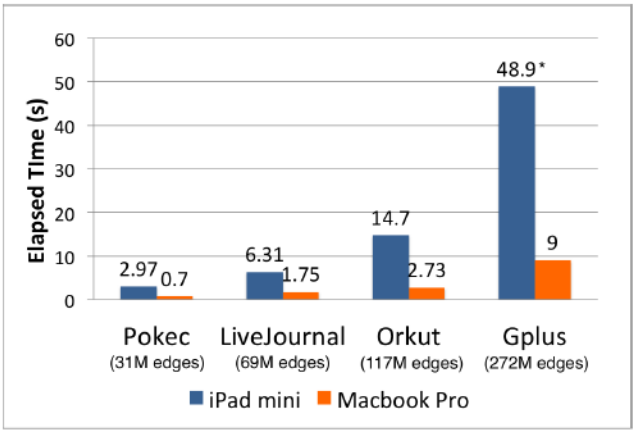


Fig. 2.

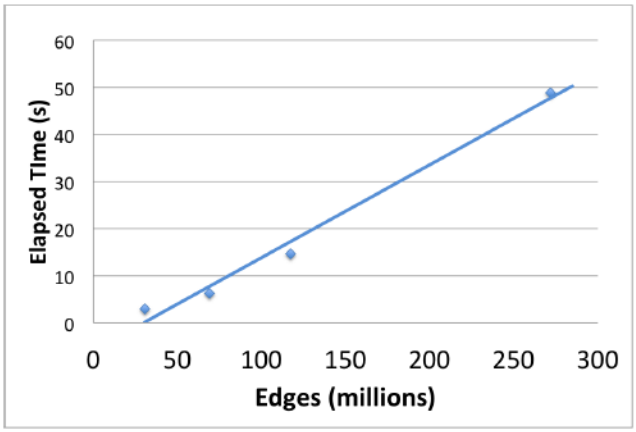
Screenshot of our iOS app called *GraphRec* that allows our user to interactively explore a graph of similar movies based on multiple seed movies. In this example, our user has chosen two movies (on the left), a horror film *Evil Dead 2* and an action-packed comedy *Hot Fuzz*. By using the personalized PageRank algorithm [9], we offer movie recommendations for horror films (like *Army of Darkness*) and comedy choices (like *Pineapple Express*), and for both aspects (as in the suggestion *Shaun of the Dead* and *Doghouse*). This app uses our *memory mapping* technique, and does not rely on the cloud for recommendation, providing smoother interaction, faster computation, and better overall user experience.



(a) Runtime of connected components



(b) Runtime of 5 iterations of PageRank



(c) PageRank runtime scales linearly with graph size; each point is one graph dataset described in Section III-B3

Fig. 3.

Comparing the runtimes of the connected components and PageRank algorithms, on the iPad mini and the Macbook Pro. The runtime of the iPad mini (1 GB RAM) is only a few times slower than the Macbook Pro (16 GB RAM). Note that to run the algorithms on the Google+ graph on the iPad mini (marked by *), we used the *SelectiveMMap* approach described in Section III-C3.

Table I

Information about the four real-world graphs used in our experiment, downloaded from the Stanford SNAP project (http://snap.stanford.edu) and Neil Zhenqiang Gong's home page at UC Berkeley (http://www.cs.berkeley.edu/~stevgong)

Graph	Nodes	Edges	Edge List File	Source
Pokec	1,632,803	30,622,564	245 MB	SNAP
LiveJournal	4,847,571	68,993,773	552 MB	SNAP
Orkut	3,072,441	117,185,083	937.5 MB	SNAP
Gplus (Google+)	17,091,929	271,915,755	2.18 GB	UC Berkeley