A Simulator for Distributed Cache Management in Friend-to-Friend Networks

Keynan Pratt
University of Calgary
Calgary, AB Canada
kjpratt@cpsc.ucalgary.ca

Carey Williamson
University of Calgary
Calgary, AB Canada
carey@cpsc.ucalgary.ca

ABSTRACT

Multimedia streaming services such as YouTube and Netflix consume a staggering amount of Internet bandwidth [1]. Furthermore, traditional mechanisms such as proxy caches, content distribution networks, and redundant traffic elimination are rendered ineffective by copyright concerns, regulatory issues, and the growing prevalence of end-to-end encryption. One possible solution is a peer-to-peer caching system with social relationships at the core of its topology construction. A social topology carries an implicit level of trust, and induces a relatively high degree of correlation between users that can be exploited by the system as a whole. For example, two users with shared interests are more likely to have relevant videos in cache for each other. This short paper discusses the design of a simulator for such a system to provide insight into the performance of different cache management policies.

CCS Concepts

•**Networks** → *Network simulations;* Network performance modeling; •**Computing methodologies** → *Simulation tools;* Massively parallel algorithms;

Keywords

Friend-to-Friend, F2F, P2P, Distributed Caching, Parallel-DES, Simulation, Haskell

1. INTRODUCTION

Friend-to-Friend (F2F) systems are a sub-class of *Peer-to-Peer (P2P)* systems. The primary distinction is that F2F systems operate with a constrained topology based on social relationships. In F2F systems, a user node may only peer with nodes for which the user knows the operator. This constraint allows the system to assume that the other nodes are trustworthy.

A notable example of an F2F system is *Freenet* [5]. Freenet is a well-known anonymity system that uses heuristic-based routing to achieve expected log-squared routing distance without any node having knowledge of the network beyond its immediate connections [5]. Additionally, the system may assume connected nodes have more properties in common than would randomly chosen nodes. Examples of such commonalities include geo-location, shared interests, education level, and socio-economic background.

This short paper describes work in progress towards a simulator for Netflix content delivery in F2F networks. One motivation for this simulator is the growing volume of media streaming traffic on campus networks. For example, recent work at the University of Calgary studied five months of campus Internet traffic [10]. Of the 2 PB of data analyzed, more than 500 TB of traffic was identified as Netflix video. Detailed analysis showed that a 12 TB cache at the campus edge could save approximately 250 TB of Internet traffic. However, even if Netflix could be persuaded to position edge nodes on every university campus, this would not solve the general case, leaving other services such as Hulu, Amazon Prime, and YouTube to install their own edge nodes. Additionally, university edge caching is made ineffective by the growing use of end-to-end encryption. This necessitates an end system solution that can operate on the encrypted data.

This work-in-progress paper outlines the construction of a simulation engine that, based on limited empirical measurements, can guide the construction and optimization of a distributed caching system. The remaining sections of this paper discuss the design of the simulator, as well as its key components for topology generation, workload generation, and cache management.

2. DESIGN RATIONALE

The Haskell language was chosen for the simulator. Haskell is a pure functional language known for its mathematical rigor. In particular, the strict type system (and the requirement that all side-effect causing code be made explicit) prevent many common programming errors that could undermine the results of simulation. Additionally, the simulator's architecture is built on Cloud Haskell[1], a message-passing library based on Erlang's OTP framework that can achieve high reliability and linear process scaling [2].

In the design of the simulator, two key properties must be maintained: the simulation must be agnostic about the data source; and the simulation must scale well with respect to the size of the social graph, and the number of content requests.

[1]https://haskell-distributed.github.io/

SIGSIM-PADS '16, May 15-18, 2016, Banff, AB, Canada

DOI: http://dx.doi.org/10.1145/2901378.2901398

In the long term, we wish to examine the efficiency of various cache management strategies and synchronization policies for F2F networks. In the initial prototype, synthetically generated data will be produced with a range of parameters to identify the properties under which each policy performs well. Thus, the simulation uses the following interface:

```
simulate :: (Cache c, Generator ContentEvent g)
         => Graph
         -> Map Vertex g
         -> Map Vertex c
         -> Process SimResults
```

where *Graph* and *Map* are standard Haskell containers, and *Process* refers to the Cloud Haskell *distributed.process*. *Process* monad.

The network topologies are assumed to be static. While social topologies are dynamic, it's assumed that changes of any one node are slight enough to have little or no impact on system behavior.

The core simulator engine operates as follows. Once simulation has begun, the graph $G(V, E)$ is partitioned using an approximation algorithm. The result is k sets containing approximately $|V|/k$ vertices each, where k is the number of processes over which the computational load is to be distributed. Each process p_i has ownership of one set M_i of *master nodes*. Process p_i constructs the set $S_i = \{v \mid u \in M_i,\ v \notin M_i,\ (u, v) \in E \vee (v, u) \in E\}$. For all $v \in S_i$, a slave replica of the master cache is created. Whenever the master cache is modified, the new entry and the simulation time at which it was inserted are transmitted to every process that possesses a slave replica. If, at time t, the process p_i attempts to read from a slave cache where the master replica is owned by p_j, it first applies any updates it has received from p_j prior to time t. If the last update received from p_j occurred at or after time t, the read of the slave replica succeeds. If the read of the slave replica fails, then the process p_i announces an update with no content occurred at time $t-1$; it then pauses simulation. p_i continues receiving updates and resumes simulation when a read of the slave replica succeeds.

The simulator design ensures deadlock-free operation. To understand this, assume process p_i is blocked waiting for an up-to-date slave replica for a node managed by p_j at time t. Then, if p_j is blocked it must be the case that p_j is blocked waiting for an event that happens prior to $t-1$, so p_j can't be blocked waiting for an event from p_i. It is easy to see how this property generalizes to many processes. Thus, deadlock is not possible in the discrete-event simulator.

3. TOPOLOGY GENERATION

The simulator currently supports several social network topology models. Of primary interest are graphs that are small-world and scale-free, as observed in real-world social networks. We consider Watts-Strogatz (WS), Barabási-Albert, and the Stochastic Block Model. For comparison, we also include the Erdős-Rényi model of random graphs.

For each of these models, the graphs generated are analyzed to understand the properties that influence the simulation results. We focus on the graph diameter, radius, average path length, clustering coefficient, and degree distribution. Brief descriptions of these metrics and properties are given below.

3.1 Graph Properties

3.1.1 Eccentricity

The eccentricity of a vertex v, denoted $\varepsilon(v)$, is the length of the shortest path $P(v, u)$ to the vertex u that is farthest away from it (i.e., there exists no vertex w such that $P(v, w) > P(v, u)$).

3.1.2 Radius & Diameter

The radius of a graph is given by $\min_{v \in V}[\varepsilon(v)]$ and the diameter is given by $\max_{v \in V}[\varepsilon(v)]$

3.1.3 Clustering Coefficient

There exist two common methods for measuring the clustering coefficient of a graph, specifically:

$$C^{\Delta} = \frac{3 * number\ of\ triangles}{number\ of\ connected\ triplets\ of\ vertices}$$

and the Watts-Strogatz clustering coefficient [12]:

$$C^{WS} = \frac{\sum_i (c_i^{ws})}{|V|} \qquad c_i^{WS} = \frac{2e_i}{k_i(k_i - 1)}$$

where, for vertex v_i, e_i is the number of connected neighbors and k_i is the degree of the vertex.

3.2 Graph Models

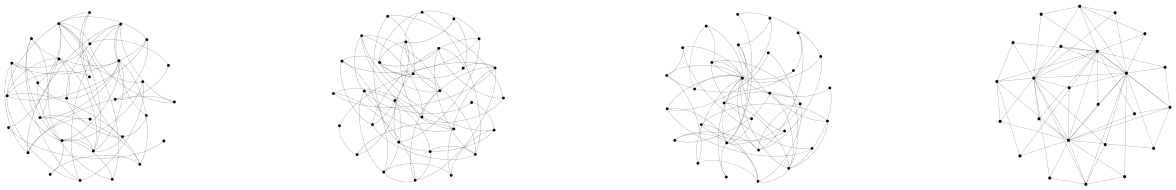
All simulated graph models accept the input parameter $n = |V|$. We constrain the parameter d_{avg}, the average degree of a vertex, to be within one order of magnitude of that observed in online social networks. According to PEW research [7], the average Facebook user has 338 friends. Thus d_{avg} is constrained to the range $[10, 4000]$. The same PEW study estimated the median number of friends at 200, with 39% of users having less than 100 friends and 15% having more than 500. This gives us a rough approximation of the slope of the degree distribution to be emulated when constructing scale-free graphs.

3.2.1 Erdős-Rényi Graphs

In an Erdős-Rényi graph, the existence of every edge is sampled according to a Bernoulli distribution with probability p [8]. Values of p are chosen such that $p = \frac{|E|}{n}$ and $p > \frac{log(n)}{n}$. The latter constraint ensures that with high probability (w.h.p.), the graph is connected [9].

3.2.2 Watts-Strogatz Graphs

The Watts-Strogatz model [13] is designed to produce graphs with a notably higher *clustering coefficient* than in Erdős-Rényi graphs. The model takes two additional parameters: k where $2|k$ and $n \gg k \gg \log n \gg 1$; and $\beta \in [0, 1]$. The graph is initially constructed as a ring lattice with neighborhood size k. Therefore the graph will contain $\frac{nk}{2}$ edges. Every edge is moved according to probability β. If $\beta = 0$, the lattice structure is preserved. As β approaches unity, the graph resembles an Erdős-Rényi graph where $p = \frac{nk}{2n^2}$. Additionally, as β goes to 1, the average path length rapidly approaches $\frac{\log n}{\log k}$, and the clustering coefficient degrades to $\frac{k}{n}$. The small-world properties of these graphs are most prominent for intermediate values of β, before the clustering coefficient drops, and the average path length shrinks. One limitation of this model is the unrealistic degree distribution. The WS degree distribution tends



(a) Erdős-Rényi Graph (b) Watts-Strogatz Graph (c) Barabási-Albert (d) q-Clique Tree

Figure 1: Graph examples for $n \approx 30$ and $d_{avg} \approx 4$

Table 1: Small ($n \approx 30$, $d_{avg} \approx 4, \beta = 0.28$)

Model	Diameter	Avg Path Length	C^{WS}
Erdős-Rényi Graphs	6	2.559	0.134
Watts-Strogatz	5	2.578	0.133
Barabási-Albert	4	2.308	0.238
q-Clique Tree	3	1.80	0.810

Table 2: Medium ($n = 1000$, $d_{avg} = 10, \beta = 0.28$)

Model	Diameter	Avg Path Length	C^{WS}
Erdős-Rényi Graphs	6	3.307	0.009
Watts Strogatz	6	3.660	0.263
Barabási-Albert	5	2.989	0.036
q-Clique Tree	5	2.787	0.874

Table 3: Large ($n = 10,000$, $d_{avg} = 20, \beta = 0.28$)

Model	Diameter	Avg Path Length	C^{WS}
Erdős-Rényi Graphs	5	3.401	0.002
Watts Strogatz	5	3.744	0.269
Barabási-Albert	5	3.065	0.011
q-Clique Tree	5	3.064	0.875

towards a Poisson, while observed social networks have a power-law distribution.

3.2.3 Barabási-Albert Graphs

The Barabási-Albert model [3] produces scale-free graphs. From an initial graph containing n_0 vertices, the remaining $n - n_0$ vertices are added. For each additional vertex, m edges are created. Each edge connects the vertex to an existing vertex v_i with probability $p_i = \frac{d_i}{2E}$ where d_i is the current degree of v_i. The resulting graph has degree distribution roughly $P(d) \sim d^{-3}$. Scale-free graphs are ultra-small-world with average path length $\frac{\log n}{\log \log n}$. However, as n grows, the clustering coefficient tends towards $n^{-0.75}$, much lower than observed in social networks.

3.2.4 Recursive Clique Trees

Recursive Clique Trees, as discussed by [6], are deterministically generated hierarchical graphs that exhibit both the small-world and scale-free properties. The graph is constructed according to two parameters q and t, where q is the clique size, and t is the recursive depth. The graph grows exponentially with parameter t, while q controls both the clustering coefficient and the degree distribution.

Examples of the four graph models are shown in Figure 1, with their graph metrics given in Table 1.

4. WORKLOAD GENERATION

Workload generation is focused on content requests that reflect general viewing habits. As a control, one class of generator samples *independently* from the empirical Zipf-like distribution identified by [10]. However, it is reasonable to consider correlation between the viewing habits of friends in a social network, as well as for content events to depend on previous events observed by the same node. The correlations arise from common interests, social recommendation, word-of-mouth, and social engagement. Dependence is most pronounced in terms of viewing habits. After watching episode 6, a user is very likely to watch episode 7, and extremely unlikely to watch episode 28.

In order to include these dependencies in our simulation, samples are taken from a composition of multiple distributions, each of which is unique to a particular node. Each node starts with the same distribution as the control. As events are processed the distribution is updated to reflect the correlation and dependence outlined above. The primary technique used is the Least-Recently-Used (LRU) stack model commonly used for modeling temporal locality in caching simulations.

A practice among some viewers is to "binge watch" specific TV series. Thus, there will be a strong tendency to watch only a small number of series at a time, consuming one to completion before moving on. Additionally, in order to accurately reflect the real world, new content will become available mid-simulation. The arrival of new content may interrupt the current series being viewed.

5. CACHING ARCHITECTURE

The design of an effective caching architecture is the main motivation for our simulator. It is desirable to avoid a cross-network query if the query would fail. Therefore, each node maintains a Bloom filter [4] of each neighbor's cache. Tuning the size of the Bloom filter relative to the cache size will be an important optimization question during simulation.

Prior to their deployment of end-to-end encryption, Netflix was known to buffer video via multiple HTTP request-response operations. In a production system, each individual HTTP request must be representable as a cache query. Additionally, each request would specify a time offset into the video. The flexibility offered by this API must be mirrored in a production cache. Therefore, each video occupies a single entry in the cache, though the entry may be incomplete or even contain holes. In order to minimize message-passing overhead in simulation, each movie or TV episode is treated as a single cache entry, storing only the integer identifying the video requested.

Common cache management policies such as FIFO, LRU, and LFU will be considered. However, traditional caching policies are applied in a single user environment; the distributed environment creates opportunities for unique cache

semantics. Consider the perception of a single node. It does this node no good for all of its neighbors to store the most popular piece of content. As soon as the closest neighbor has the content in cache, the node ceases to gain benefit from other neighbors caching the same content. In some sense, cache space is wasted if they do so. The neighbor, when pulling a piece of content, may choose not to cache it.

Recall that the social topology is small-world and therefore has a high clustering coefficient. In other words, your friends are frequently friends with each other. Assume that a node can discover or approximate the distance of each neighbor to the closest source (i.e., Netflix server), and to any mutually connected neighbors. Let the in-network availability of a piece of content γ be defined as the percentage of neighbors who have a mutual neighbor who has the content and is closer to them than to the source. Then, the node may choose to store the content with probability $p = f(\gamma)$. Two functions of interest are:

$$f(\gamma) = 1 - \gamma \qquad \text{and} \qquad f(\gamma) = 1 - S(12\gamma - 6)$$

where $S(x)$ is the sigmoid function. An alternative measure of availability would be for each node to provide each neighbor with a second Bloom filter suggesting the pieces of content to which it has nearby access. This measure would produce lower message overhead and greater accuracy at the expense of leaking information to nodes without a direct connection.

6. CURRENT CHALLENGES

6.1 Random Number Generation

One unexpected technical challenge was random number generation. Implementations of common pseudo-random number generators (PRNG), such as the Mersenne twist, are available in Haskell. However, these implementations may be incomplete, or have drawbacks such as the inability to safely split the period across multiple parallel streams. We have implemented the xorshift-128-plus algorithm to serve as the underlying U(0,1) PRNG. While no PRNG is known to pass all of the standard TestU01 [11] statistical tests, xor0shift passes many more of these tests than does the Mersenne twist, which is believed to be the most popular PRNG in use today. Additionally, the xorshift-128-plus is fast, requires minimal state, and has a period many orders of magnitude larger than required in most simulations.

6.2 Parallel Performance

Our simulator is designed to exploit the inherently distributed nature of the problem. Each simulated node depends only on having current knowledge of its neighbors. Simulation run-time is expected to scale linearly with the number of vertices in the social graph for a fixed value of d_{avg}. To see why this is true, consider the extreme case $n = k$, where all edges cross partition boundaries. Then for a given content event, the node must process d incoming messages, perform the update reading from d slave replicas, and broadcast d outgoing messages. Each process will need to remain in near lockstep, so the entire simulation will progress at the rate of the slowest node (with highest degree). Synchronization overhead in a partition is bounded by the number of partition-crossing edges, while event processing time is bounded by the degree of the node.

The key to simulation performance will be good partitioning of the vertices.

An optimal partitioning minimizes the number of edges between processes and the number of slave replicas that need to be maintained. The upper bound on the number of crossing edges for a 2-cut is $\frac{nd}{2}$. As the slope of the distribution increases, efficient partitioning is expected to become more likely.

Inefficient partitioning results in higher memory usage and higher communication overhead when cache updates are broadcast. Provided $n \gg d_{avg}$, performance should scale well if k is increased proportionally. As either k or d approach n, message overhead will become large.

7. FUTURE DIRECTIONS

The path to a working simulation is clear in most respects. It is still necessary to identify a good partitioning algorithm for each graph model. The implementation is expected to be complete in time to present preliminary results in May.

8. REFERENCES

[1] V. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. Zhang, M. Varvello, and M. Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM Transactions on Networking*, 23(6):1984–1997, Dec 2015.

[2] J. Armstrong. Erlang. *Communications of the ACM*, 53(9):68, 2010.

[3] A. Barabási and R. Albert. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(January):48–94, 2002.

[4] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Designing Privacy Enhancing Technologies.*, pages 46–66, 2001.

[6] F. Comellas, G. Fertin, and A. Raspaud. Recursive graphs with small-world scale-free properties. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 69(3 2):2–5, 2004.

[7] M. Duggan and A. Smith. Social media update 2013. *Pew Internet and American Life Project*, 2013.

[8] P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[9] P. Erdos and A. Renyi. The evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5(1):17, 1961.

[10] M. Laterman. NetFlix and Twitch Traffic Characterization. Master's thesis, University of Calgary, 2015.

[11] P. L'Ecuyer and R. Simard. TestU01: A c Library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4), 2007.

[12] Q. Telesford, K. Joyce, S. Hayasaka, J. Burdette, and P. Laurienti. The Ubiquity of Small-World Networks. *Brain Connectivity*, 1(5):367–375, Dec. 2011.

[13] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–2, 1998.