Consumer behavior and algorithm design

Prabhakar Raghavan
Warm up

A small demonstration on using our understanding of behavior
Connectivity Server

- In-memory support for queries on the web graph
  - Which URLs point to a given URL?
  - Which URLs does a given URL point to?

- Applications
  - Crawl control
  - Web graph analysis
  - Link analysis
Adjacency lists

• Each URL represented by an integer
  – E.g., for a 4 billion page web, need 32 bits per node

• Naively, need 64 bits to represent each edge

• Will show scheme achieving 3 bits/edge
  – Further optimizations get to 2 bits/edge
Adjacency list compression

- Properties exploited in compression:
  - Similarity (between lists)
  - Locality (many links from a page go to “nearby” pages)
  - Use gap encodings in sorted lists
  - Distribution of gap values
Main ideas of Boldi/Vigna

- Consider lexicographically ordered list of all URLs, e.g.,
  - ...
  - www.stanford.edu/alchemy
  - www.stanford.edu/biology
  - www.stanford.edu/biology/plant
  - www.stanford.edu/biology/plant/copyright
  - www.stanford.edu/biology/plant/people
  - ...

Basic idea

• Each of these URLs has an adjacency list
• Due to templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering
• Express adjacency list in terms of one of these
• E.g., consider these adjacency lists
  – 1, 2, 4, 8, 16, 32, 64
  – 1, 4, 9, 16, 25, 36, 49, 64
  – 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
  – 1, 4, 8, 16, 25, 36, 49, 64

Encode as (-2), remove 9, add 8
Summary

- Depends on similarity/locality in canonical ordering
  - Lexicographic ordering works well for the web, thanks to the behavior of users and page creators
- Adjacency queries can be answered efficiently
  - To find neighbors, trace back the chain of prototypes
  - This chain is typically short in practice (since similarity is mostly intra-host)
- Easy to implement one-pass algorithm
Three themes

How consumer behavior is changing what we compute, and how
Three themes

• Subjective functions
• Riding the fat tails
• PRAM lost, parallelism regained
Subjective functions

- Consumers drive consumption of cycles, bandwidth, storage
- Lots of cycles still expended in SORT and SELECT but – a large and growing chunk in “making users happy”
- 7 billion “objective” functions
  - Don’t bother trying to write them down
  - Find and optimize proxies
Example subjective function

• What’s a good search ranking?
  
  *Proxy: Make each document a vector in the space of all words; do the same for a query*
  
  *Score = dot product of query and document vectors*
  
  *Algorithmically: Select 100 documents of highest dot product to query, efficiently.*

(Salton, circa 1975.)
A few other subjective functions

• Find me beautiful pictures of sunsets
  – What’s a sunset?
  – What’s beautiful?

• Given feeds from 6 cameras at a stadium, find the best camera view at each point in time

• Show me songs I’ll like

• [Spelling correction]
Subjective function characteristics

• How do we assess our algorithm?
  – Ask users! (What does this mean?)

• Users have some tolerance for junk – no notion of absolute correctness

• “Infinite” computing doesn’t help
  – Resources typically not the primary constraint in algorithm design
  – Even with unbounded computing, don’t know how to do better
The first 25 years of search ranking

• Focus on tweaking the weights in the basic vector space
  – Presumption of expert “information scientists” seeking information

• Starting 1995, sea changes
  – Ordinary consumers with extraordinary expectations
  – Adversarial content
Better search ranking

• Say your student comes to you and says – I have a new score for documents, called Pagerank
  – How do you combine dot product score and Pagerank?

• Explore combinations of scores that regress well to user judgments
  – Explicit surveys of trained raters
  – Crowd ratings
  – Clicks
More generally

• The world is full of signals for good search ranking (or beautiful pictures, likeable songs, memorable slogans)

• Combining them left to Machine Learning
  – At core, large math programs

• Algorithmic improvement entails
  – Inventing new signals ("features")
  – Volume of rating data to train ML
Upshot

• We learn functions we can’t cleanly describe

• Approximate, and iteratively improve (proxy for) user happiness
  – Standard methodology in research/industry
  – Data volume is paramount

• Abdicate algorithm design to machine learning kernels
Computational aesthetics

- Consider a question such as: *Which of these web pages is more appealing?*
- An assessment of aesthetics
- We can do a decent job on such problems using the general approach above
  - No pretense of understanding aesthetics
  - Nor of synthesizing creativity
Riding the fat tails

Graphs with small average degree, but some nodes have massive degrees
“The Long tail”

ANATOMY OF THE LONG TAIL

Online services carry far more inventory than traditional retailers. Rhapsody, for example, offers 19 times as many songs as Wal-Mart’s stock of 39,000 tunes. The appetite for Rhapsody’s more obscure tunes (charted below in yellow) makes up the so-called Long Tail. Meanwhile, even as consumers flock to mainstream books, music, and films (right), there is real demand for niche fare found only online.

THE NEW GROWTH MARKET:
OBSCURE PRODUCTS YOU CAN’T GET ANYWHERE BUT ONLINE

Chris Anderson, Wired, Oct 2004
Infinite-inventory stores: two properties

1. The vast majority of products are “misses”
2. These “worst-sellers” in aggregate account for a large fraction of sales
   - 25% of Netflix sales and 30% of Amazon sales are reportedly for products not available in traditional stores
   - But ... How many people care about the tail?
     - Could the world be bimodal?
Could this be two normal distributions?

- Two views of human behavior consistent with heavy tails
  1. Bimodal population?
Many datasets show in fact that ...

- Two views of human behavior consistent with heavy tails
  
  2. Or “homogenous” population?
- We are all partially eclectic
How many links into a random web page?

\[ \Pr_u[Deg(u) = i] \approx i^{-2.1} \]
Heavy tails

• Decreasing probability distributions over integers in $[1,n]$
  – E.g., search query frequencies in decreasing order

• For any fixed $k$ (say, 10000), the probability mass in all buckets $>k$ is a “big” constant

• Arise in observed statistics stemming from human behavior
  – Number of friends, popularity of movies ...
A typical heavy tailed distribution

• $p[i] \sim 1/ i^\beta$ - so-called power law
  – log-log plot is a straight line
  – For $\beta > 1$, bounded expectation

• Other distributions – log-normal, Pareto
  – cf Mitzenmacher survey

• Misnomer:

  Long-tailed distributions
Copying model
aka preferential attachment

- Each page is a node, each hyperlink is a directed edge
- Edges arrive in sequence
- For $0 \leq \alpha \leq 1$,
  - With probability $\alpha$, link to a random page
  - With probability $1-\alpha$, copy a random link
- Explore-exploit of news stories on news sites
Informal analysis

- Let $X_j(t)$ be the number of pages of degree $j$ at time $t$

\[
\frac{dX_j}{dt} = \alpha \frac{X_j - 1}{t} - \alpha \frac{X_j}{t} + (1 - \alpha)(j - 1) \frac{X_j - 1}{t} - (1 - \alpha) \frac{X_j}{t}
\]

- Limiting distribution

\[
X_i \approx i^{-(2-\alpha)/(1-\alpha)}
\]

- Proof uses bounded difference argument
Consequence for probabilistic analysis

• Consumer phenomena yield heavy tails
• Most probabilistic analysis uses light-tailed distributions (binomials, Poisson ...)
  – Reason: we use simple generative models based on the superposition of independent events
• Work ahead – analysis of data structures and algorithms under heavy-tailed inputs
  – Early work on inverted indexes and graph compression
Work ahead

• Better modeling and analysis
  – Underlying individual and social behavior

• Behavior of consumers correlated
  – To each other
  – To “market forces”
    • Recommendation systems
    • What’s on the front page of the NY Times

• What distributions arise from consumers subject to such forces?
  – How real is the “filter bubble”?
Parallelism today

World computer market = 6

(Howard Aiken; mis-attributed to Thomas Watson Sr.)
Rethinking parallelism

• PRAMs – simple abstraction
  – Fine-grained
  – Synchronous
  – Equal access
  – (In principle) straightforward programming

• Modifications
  – Networks of fixed topology (Hypercubes, CCC’s)
  – BSP, ...
Parallelism – reality

- Fault- and error-prone processors
- Coarse-grained synchrony
- Locality matters
- Software development/maintenance costs dwarf costs of many commodity machines
- Want to solve problems such as
  - “Which search queries co-occur?”
  - “Which friends to recommend?”
Parallel programming is hard

• Threaded programs are difficult to **test**
  • One successful run is not enough
• Threaded programs are difficult to **read**
• Threaded programs are difficult to **debug**
  • Hard to repeat the conditions to find bugs
• More machines means more breakdowns

Special thanks to Sergei Vassilivitskii for some of these slides.
MapReduce: easy parallel programming

- Tracks jobs and restarts if needed
- Takes care of data distribution and synchronization
- But there’s no free lunch:
  - Imposes a structure on the data
  - Only allows for certain kinds of parallelism
MapReduce basics

- **Data**: Represented as \(<\text{Key}, \text{Value}\>\) pairs

- **Example**: A Graph is a list of edges
  
  - **Key** = \((u,v)\)
  
  - **Value** = edge weight
MapReduce architecture

- Operations:
  - **Map**: \(<\text{Key, Value}> \rightarrow \text{List(<Key, Value>)}\)
  - **Shuffle**:Aggregate all pairs with the same key
  - **Reduce**: \(<\text{Key, List(Value)}> \rightarrow \text{<Key, List(Value)>}\)
MapReduce environments

- 10s to 10,000s processors
- Sublinear Memory: A few Gb of memory/machine, even for Tb+ datasets
  - Unlike PRAMs: memory is not shared
- Batch Processing
For an input of size $n$

- Cannot store entire data in memory
  - Sublinear memory per machine $O(n^{1-\varepsilon})$
- Machines in a cluster do not share memory
  - Sublinear number of machines: $O(n^{1-\varepsilon})$
- Synchronization
  - Computation proceeds in rounds
  - Aim for $O(1)$ rounds
Counting triangles in a graph

Sequential Version:
foreach $v$ in $V$
    foreach $(u,w)$ in $\text{Adjacency}(v)$
        if $(u,w)$ in $E$
            $\text{Triangles}[v]++$

- Running time: $\Sigma (d_v)^2$
- In practice this is quadratic, as some vertex will have very high degree
Naïve MapReduce version

• Round 1: Generate all possible length 2 paths
• Round 2: Check if the triangle is complete
• Round 3: Sum all the counts
Fat tails strike again

• How much parallelization can we achieve?
  – Generate all the paths to check in parallel
  – The running time becomes $\max_v (d_v)^2$

• Data skew defeats naïve parallelization

• Bottleneck: high degree nodes
  – E.g., 3.2 Million neighbors, must generate 10 Trillion ($10^{13}$) potential edges to check.
  – Generating 100M edges to check per second, 100K seconds ~ 27 hours.
Pivot on lowest degree

Sequential Version [Schank ’07]:

```plaintext
foreach v in V
    foreach {u,w} in Adjacency(v)
        if deg(u) > deg(v) && deg(w) > deg(v)
            if (u,w) in E
                Triangles[v]++
```

Why does it help?

• Partition nodes into two groups:
  – **Low**: \{v : d_v ≤√m\}
  – **High**: \{v : d_v >√m\}

• There are at most √m High nodes
  – Each produces paths to other High nodes:
    • \(O(m)\) paths per node
  – Therefore they generate: \(O(m^{3/2})\) paths
Low-degree paths

• Let \( n_i \) be the number of nodes of degree \( i \)
• The total number of paths generated by Low nodes can also be shown to be \( O(m^{3/2}) \)
• Total work is \( O(m^{3/2}) \) which is optimal
• But what about (parallel) runtime?
What made this work?

• The algorithm automatically load balances
  – Every node generates $O(m)$ paths to check
  – Hence the reducers take about the same time to finish
  – Fat tails get tamed ...

• Improvement in per-round runtimes:
  ~2 orders of magnitude on large graphs ($10^{9+}$)
Closing thoughts

• Ordinary users consume a growing share of computation
  – Their expectations shape the computational problems we pursue
  – Their behavior reshapes the data distributions we observe
  – Recasts how we think about and teach algorithms

• Tremendous challenges and benefits to CS
  – Inevitably we transform – and are transformed by
  – the social sciences