Cloud Computing mit mathematischen Anwendungen

Vorlesung SoSe 2009

Dr. Marcel Kunze
Karlsruhe Institute of Technology (KIT)
Steinbuch Centre for Computing (SCC)
Agenda Cloud Computing

1. Einleitung
   - Was ist Cloud Computing?
2. Grundlagen
   - Virtualisierung, Web Services,…
3. Cloud Architekturen
   - Infrastruktur, Plattform, Anwendung
4. Cloud Services
   - Amazon Web Services, Google App Engine
5. Aufbau einer Cloud
   - OpenCirrus Projekt, Eucalyptus, Hadoop, Google
6. Cloud Algorithmen
   - MapReduce, Optimierungsverfahren, Suchmaschinen, …

Praktische Übungen und Anwendungen
Vorlesung im Web:
http://www.mathematik.uni-karlsruhe.de/mitglieder/lehre/cloud2009s/
Search: A Technical History

- Search engines have been around a lot longer than you think.
- Almost all of them are dead and gone, but their ideas live on.
- Search existed before the Web, though it was a very different beast.

Source: M. Cafarella
Primordial Era: 1960s-1994

- Electronic content was rare and expensive
- Only large organizations with huge well curated archives (libraries, govts) had any need for search
- CPU & storage were expensive, networked systems very rare
- Most systems were small, searched only metadata (like card catalogs)

- Document ranking was not a huge problem
  - Relatively few documents
  - Clean metadata
  - Boolean operators commonplace
- Two important technical contributions
  - Inverted index
  - Tf/idf & vector document model

Source: M.Cafarella
Tf/idf: Vector Document Model

- Search for best matching document about e.g. “Information System Retrieval” in a set of D documents (number of terms k=3)
- Count term frequency Tf in each document d. Example:
  - Tf(i=0, Information) = \{0, 3, 6, 0, 1\}
  - Tf(i=1, System) = \{0, 1, 1, 0, 0\}
  - Tf(i=2, Retrieval) = \{1, 1, 3, 0, 0\}

\[
\begin{align*}
  tf_i &= \frac{n_i}{\sum_k n_k} \\
  \text{idf}_i &= \log \frac{|D|}{|\{d : t_i \in d\}|}
\end{align*}
\]

- Tf/idf: Vector of term frequency weighted by number of documents where term appears
- | D | : total number of documents in the corpus
- \{d: t_i \in d\}: number of documents where the term t_i appears (that is n_i \neq 0).
Tf/idf: Vector Document Model

- Tf/idf for a term places a search in N-dim space
- Documents that are “close together” in space are similar in meaning.
- Relevant to text processing
- Common web analysis algorithm
- NB. It is better to count references than frequency (Web has hyperlinks)!
Frequency versus References

Google trends

Searches

grid computing, cloud computing

Searches

Websites

grid computing, cloud computing

Search Volume index

News reference volume

A
Yahoo in ‘cloud computing’ research with HP-Intel
WA today - Jul 29 2008

B
Infrastructure Cloud Computing
SYS-CON Media - Oct 28 2008

C
Sun Shines on Cloud Computing
Sci-Tech Today - Mar 18 2009

D
3Leaf Addresses Cloud Computing and On Demand
SYS-CON Media - Mar 30 2009

E
Cisco Offering Cloud Computing Security
eWeek - Apr 21 2009

F
TIBCO to Merge SOA and Cloud Computing
SYS-CON Media - Jun 3 2009
The Web (1994-)

- The popularization of the Web in the 1990s led to a crazy explosion of search engine companies
- Web search was a vastly different problem compared to previous systems
  - Content was cheap but messy
  - Storage was becoming cheap
  - Finding a document became harder
  - Users were much less sophisticated

Source: M. Cafarella
Search Engine Size over Time

Number of indexed pages, self-reported

Source: M. Cafarella
Search Engine Storage Costs

- Figure 10kb to index one Web page plus a compressed cached copy
- In 2008, 1GB costs ~$0.15
  - 100k docs per gig, so $0.0000015/doc
  - 50M docs costs $75.00
- In 1990, 1GB costs $1000.00
  - 100k docs per gig, so $0.01/doc
  - 50M docs costs $500k
- Just about within reach for startup search companies

Source: M.Cafarella
WebCrawler

- Created in 1994 by a University Washington student
- Notable features:
  - First dynamic crawler (rather than using hand-curated corpus)
- Fate:
  - Bought by AOL, then Excite, then InfoSpace
  - Now a meta-engine, serving results from elsewhere

Source: M.Cafarella
Excite (aka Architext)

- Created in 1994 by Stanford University grads
- Notable features:
  - Full-text indexing for Web pages
  - “Related search” suggestions
  - Famous in mid-90s for consuming tons of expensive high-end Sun machines
- Fate:
  - Went public, bought many other companies, merged with @Home, collapsed in bankruptcy, then sold for parts

Source: M. Cafarella
Infoseek

- Created in 1994
- Notable features:
  - Very fancy query language (booleans, NEAR, etc)
  - Performed some linguistic analysis, including stemming. Gave stemming a bad name for a decade.
- Fate:
  - Bought by Disney in 1998

Source: M.Cafarella
Inktomi

- Created in 1996 by UCB grad student
- Notable features:
  - Distributed commodity-box infrastructure
  - Resold its search engine to other destination sites (Hotbot, Yahoo, others)
  - Search was just one of several products (others were caches and video serving)
- Fate:
  - Went public, stock collapsed in crash, sold to Yahoo in 2002

Source: M.Cafarella
 AltaVista

- Created in 1995 as a DEC research project
- Notable Features:
  - Originally meant to demo new 64-bit Alpha processor: high speed & huge address space
  - First really high-quality multithreaded crawler: 30m pages at launch!
  - Recognized that page ranking was an issue, but used awful solution: URL length
- Fate:
  - Compaq bought DEC, then sold AV to CMGI, which sold AV to Overture, which was then bought by Yahoo!

Source: M.Cafarella
Google

- Founded in 1998 by Larry Page and Sergey Brin
- Major feature was PageRank (Page, 1998)
  - Largely solved page-ranking problem faced by AltaVista
  - First major commercial deployment of link-based methods
  - Really miraculous when compared to other methods at the time
- However, link-based methods were common in academia

Source: M.Cafarella
PageRank

- If a user starts at a random web page and surfs by clicking links and randomly entering new URLs, what is the probability that s/he will arrive at a given page?
- The PageRank of a page captures this notion
  - More “popular” or “worthwhile” pages get a higher rank

Source: A. Kimball
PageRank: Formula

Given page A, and pages \( T_1 \) through \( T_n \) linking to A, PageRank is defined as:

\[
PR(A) = (1-d) + d \left( \frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)} \right)
\]

\( C(P) \) is the cardinality (out-degree) of page \( P \)
\( d \) is the damping ("random URL") factor

Source: A. Kimball
PageRank: Intuition

- Calculation is iterative: \( \text{PR}_{i+1} \) is based on \( \text{PR}_i \)
- Each page distributes its \( \text{PR}_i \) to all pages it links to. Linkees add up their awarded rank fragments to find their \( \text{PR}_{i+1} \)
- \( d \) is a tunable parameter (usually = 0.85) encapsulating the “random jump factor”

\[
\text{PR}(A) = (1-d) + d \left( \frac{\text{PR}(T_1)}{C(T_1)} + \ldots + \frac{\text{PR}(T_n)}{C(T_n)} \right)
\]

Source: A. Kimball
PageRank: First Implementation

- Create two tables 'current' and 'next' holding the PageRank for each page. Seed 'current' with initial PR values.
- Iterate over all pages in the graph, distributing PR from 'current' into 'next' of linkees.
- current := next; next := fresh_table();
- Go back to iteration step or end if converged.

Source: A. Kimball
Distribution of the Algorithm

Key insights allowing parallelization:
- The 'next' table depends on 'current', but not on any other rows of 'next'
- Individual rows of the adjacency matrix can be processed in parallel
- Sparse matrix rows are relatively small

Consequences of insights:
- We can map each row of 'current' to a list of PageRank “fragments” to assign to linkees
- These fragments can be reduced into a single PageRank value for a page by summing
- Graph representation can be even more compact; since each element is simply 0 or 1,
- only transmit column numbers where it's 1

Source: A.Kimball
PageRank: Iteration

Map step: break page rank into even fragments to distribute to link targets

Reduce step: add together fragments into next PageRank

Iterate for next step...

Source: A. Kimball
Phase 1: Parse HTML

- Map task takes (URL, page content) pairs and maps them to (URL, (PR_{init}, list-of-urls))
  - PR_{init} is the “seed” PageRank for URL
  - list-of-urls contains all pages pointed to by URL
- Reduce task is just the identity function

Source: A. Kimball
Phase 2: PageRank Distribution

- Map task takes \((URL, (cur\_rank, url\_list))\)
  - For each \(u\) in \(url\_list\), emit \((u, cur\_rank/|url\_list|)\)
  - Emit \((URL, url\_list)\) to carry the points-to list along through iterations

- Reduce task gets \((URL, url\_list)\) and many \((URL, val)\) values
  - Sum vals and fix up with \(d\)
  - Emit \((URL, (new\_rank, url\_list))\)

\[
PR(A) = (1-d) + d \left( \frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)} \right)
\]

Source: A.Kimball
Finishing up...

- A subsequent component determines whether convergence has been achieved (Fixed number of iterations? Comparison of key values?)
- If so, write out the PageRank lists - done!
- Otherwise, feed output of Phase 2 into another Phase 2 iteration

Source: A. Kimball
“Google” Circa 1997 (google.stanford.edu)

Source: J. Dean
Research Project, circa 1997

Source: J. Dean
Google Search Engine

- The Google Web Server (GWS) takes your query and coordinates the search and response.
- The index is partitioned into “shards.” Each shard indexes a subset of the docs (web pages) and can be searched by multiple index servers.
  - The GWS routes your search to one index server.
  - The result is an ID for every doc satisfying your search, rank-ordered by relevance.
- The docs, too, are partitioned into “shards” – the partitioning is a hash on the doc ID. Each shard contains the full text of a subset of the docs and can be searched by multiple doc servers.
  - The GWS sends appropriate doc IDs to one doc server.
  - The result is a URL, a title, and a summary for every relevant doc.
- The GWS builds an HTTP response to your search and ships it off.
“Corkboards” (1999)

Source: J. Dean
Serving System, circa 1999

Source: J.Dean
Google Data Center (2000)

Source: J. Dean
Early 2001: In-Memory Index

Source: J. Dean
Serving Design, 2004 edition

- Cache servers
- Requests
- Parent Servers
- Repository Manager
- Leaf Servers
- File Loaders
- GFS

Source: J. Dean
2007: Universal Search

Source: J. Dean
Current Machines

- In-house rack design
  - PC-class motherboards
  - Low-end storage and networking hardware
  - Linux + in-house software

Source: J. Dean
Google IT-Factory

Based on Container Technology

Source: J. Dean
Energy Efficiency

http://www.youtube.com/watch?v=zRwPSFpLX8I

Source: J. Dean
Design of Very Large-Scale Computer Systems

Future scale: $\sim 10^6$ to $10^7$ machines, spread at 100s to 1000s of locations around the world, $\sim 10^9$ client machines

- power adaptivity
- zones of semi-autonomous control
- consistency after disconnected operation

Source: J. Dean
Power by component at different activity levels

- CPU no longer dominates system power
- CPUs are more energy-proportional than the rest of the system

Source: J. Dean
A Tale of two Machines

Source: J.Dean

Average power: ~120W

2x

80W

160W

Idle

Activity Level

Peak

20x

60W

Rest

Lance’s Tour stage

Source: J.Dean
Consistency in Internet Services

Source: J. Dean
Consistency in Internet Services (Actually)

Source: J. Dean
Summary

- **Google Search Engine**
  - PageRank Algorithm
  - Based on link reference count

- **Google Development**
  - Infrastructure drives system development
  - Energy efficient datacenters and algorithms
Steinbuch Centre for Computing (SCC)
Thank you for your attention.