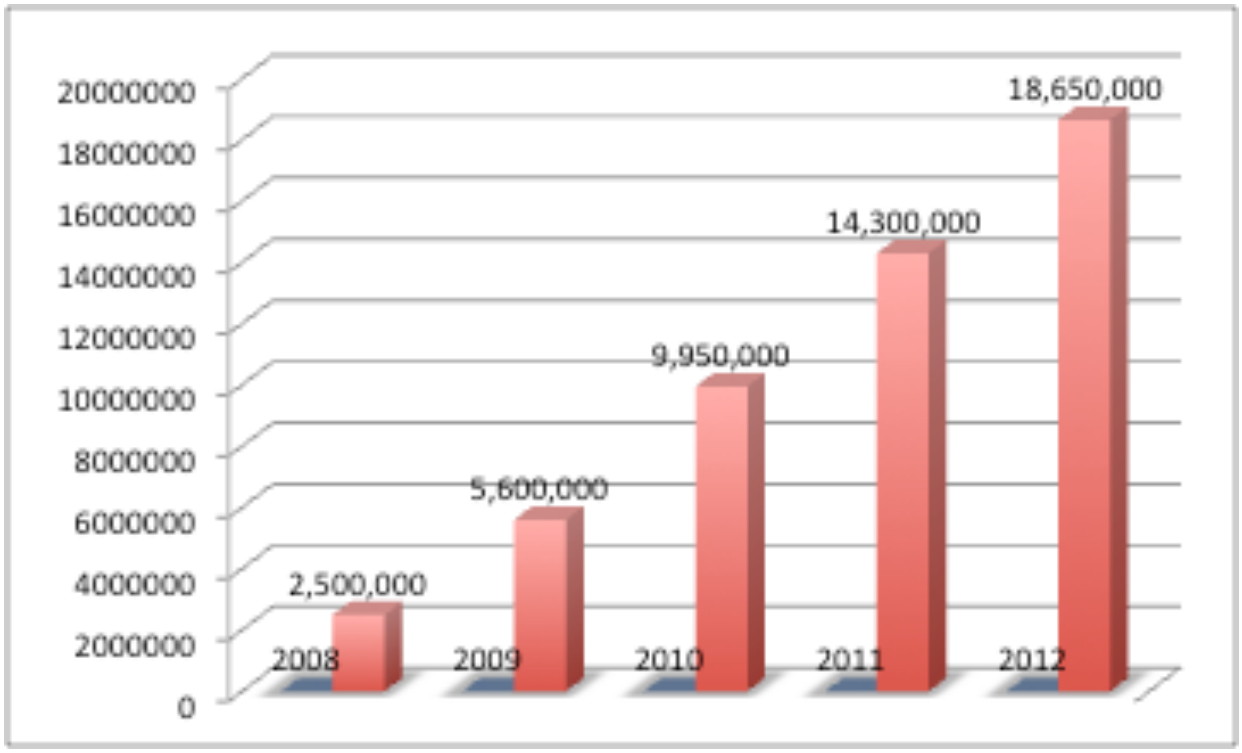
HathiTrust Large Scale Search

Tom Burton-West
Information Retrieval Programmer
August 19th 2010

Challenges

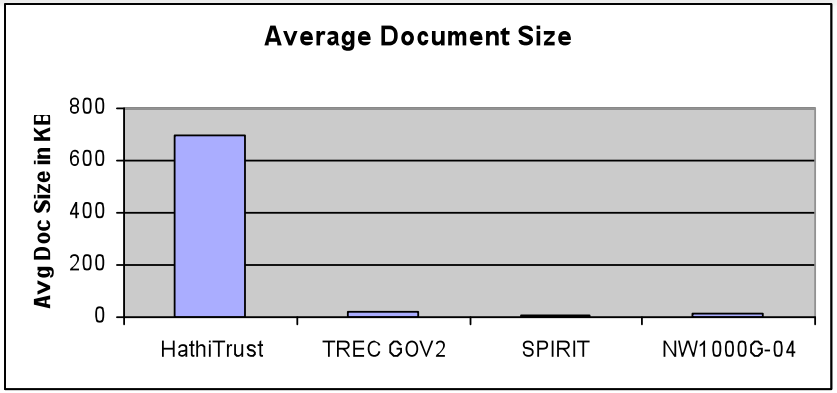
- Goal: Design a system for full-text search that will scale to 7 million -20 million volumes (at a reasonable cost.)
- Challenges:
  - Must scale to 20 million full-text volumes
  - Very long documents compared to most large-scale search applications
  - Multilingual collection
  - OCR quality varies

growth trajectory



Long Documents

- Average HathiTrust document is 700KB containing over 100,000 words.
  - Estimated size of 7 million Document collection is 4.5TB.
- Average HathiTrust document is about 38 times larger than the average document size of 18KB used in Large Research test collections



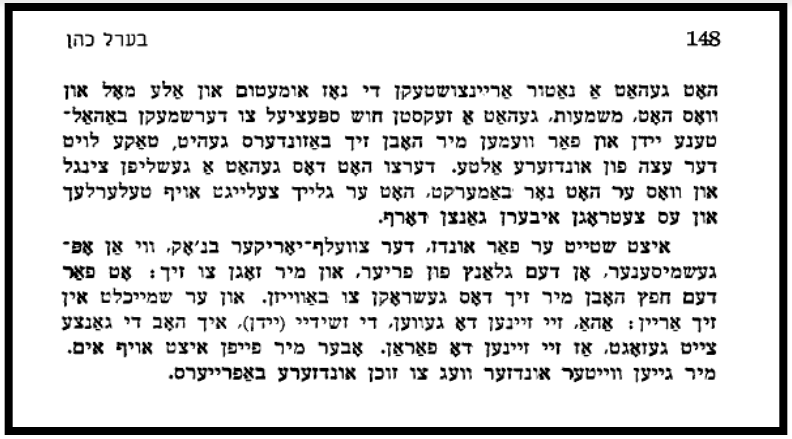
Collection	Size	Documents	Average Doc size
HathiTrust	4.5 TB (projected)	7 million	700 KB
TREC GOV2	0.456 TB	25 million	18 KB
SPIRIT	1 TB	94 million	10 KB
NW1000G-04	1.3 TB*	100 million	16 KB

Multilingual

- 400+ languages, 40 languages with over 1,000 volumes (some volumes in multiple languages)
- Currently all languages in one index
- Lowest common denominator tokenizing
- Some languages are challenging for Information Retrieval
  - CJK
  - Arabic
  - Vietnamese

OCR

- The OCR varies in quality
- This makes the index larger and slower
- Example: Hebrew characters not recognized



בערל כהן 148

האָט געהאַט אַ נאַטור אַריינצושטעקן די נאָז אומעטום און אַלע מאָל און וואָס האָט, משמעות, געהאַט אַ זעקסטן חוש ספעציעל צו דערשמעקן באַהאַל־ טענע יידן און פאַר וועמען מיר האָבן זיך באַזונדערס געהיט, טאַקע לויט דער עצה פון אונדזערע אַלטע. דערצו האָט דאָס געהאַט אַ געשליפן צינגל און וואָס ער האָט נאָר באַמערקט, האָט ער גלייך צעלייגט אויף טעלערלעך און עס צעטראָגן איבערן גאַנצן דאָרף.

איצט שטייט ער פאַר אונדז, דער צוועלף־יאָריקער בן׳אָק, ווי אַן אַפ־ געשמיסענער, אַן דעם גלאַנץ פון פריער, און מיר זאָגן צו זיך: אָט פאַר דעם חפץ האָבן מיר זיך דאָס געשראָקן צו באַווייזן. און ער שמייכלט אין זיך אַריין: אַהאַ, זיי זיינען דאָ געווען, די זשידיי (יידן), איך האָב די גאַנצע ציים געזאָגט, אַז זיי זיינען דאָ פאַראַן. אָבער מיר פייפן איצט אויף אים. מיר גייען ווייטער אינדזער וועג צו זוכן אונדזערע באַפרייערס.

Щи 7-11 Л

148

px "?SB y"?x px DiByaix T8J 'i т.рувглхзтчх ивхз х -'гхлха ipyatnyi ix ^yxyvo win 1ворут 8 вхлуа -, B'i1? урхв ,вмуд DiyijiTxa тт PXJI та lyayn ixs

8 ВХПУЛ 081 В8Л 1Х1У1

:Т>Т ІХ

.01У"1э«э

T'a]1X ,1У1Э Î1S

1Г"1ч?а ix

') "ТЕП '

та

pu ix луп

ПУТ

I та

"Т

www.hathitrust.org

Testing Program

- Test scalability of Solr open source search engine with HathiTrust full-text content.
  - Solr is based on the open source Lucene library
  - Solr provides a mechanism for distributed search with sharded indexes on multiple hosts for scaling the index and with replication for scaling load.
- Tested with index sizes from 100,000 volumes up to 1 million in 100,000 volume increments

Testing Program

- Tested with memory at 4, 8, 16, and 32 GB
- Tested different configurations
  - Single index single machine
  - Split index (2 shards) single machine
  - Split index (2 shards) one on each of two machines

Testing Program: Hardware

- Servers
  - 2 PowerEdge 1950 blades
  - 2 Dual Core 3.0 GHz Processors
- NFS server
  - 4 JetStor 416S Raid Arrays RAID 6
  - 7 SATA drives 7500 RPM
- Network for NFS-Servers
  - Gigabit private/unrouted network

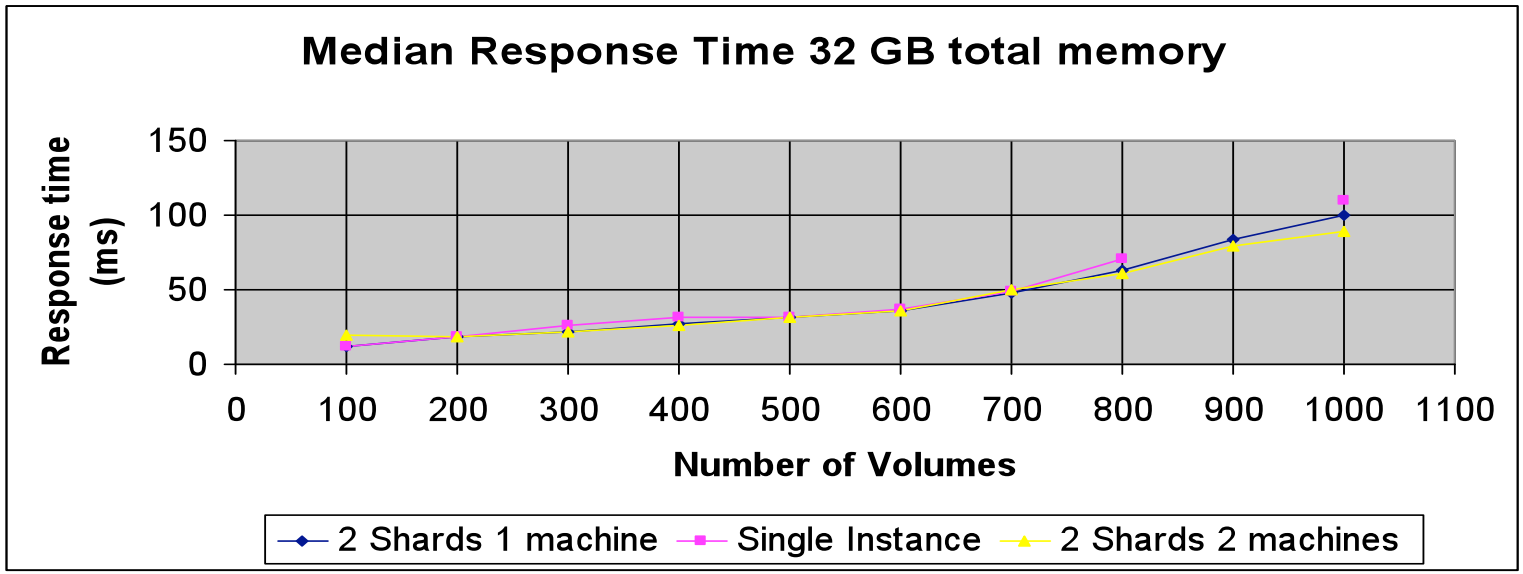
Testing Program: Protocol

- 10,000 warm-up queries (from OPAC logs)
- 1,000 test queries (from OPAC logs)
- More details in report:
- http://www.hathitrust.org/technical_reports/Large-Scale-Search.pdf

Testing Program Results

Scalability and Machine Configurations

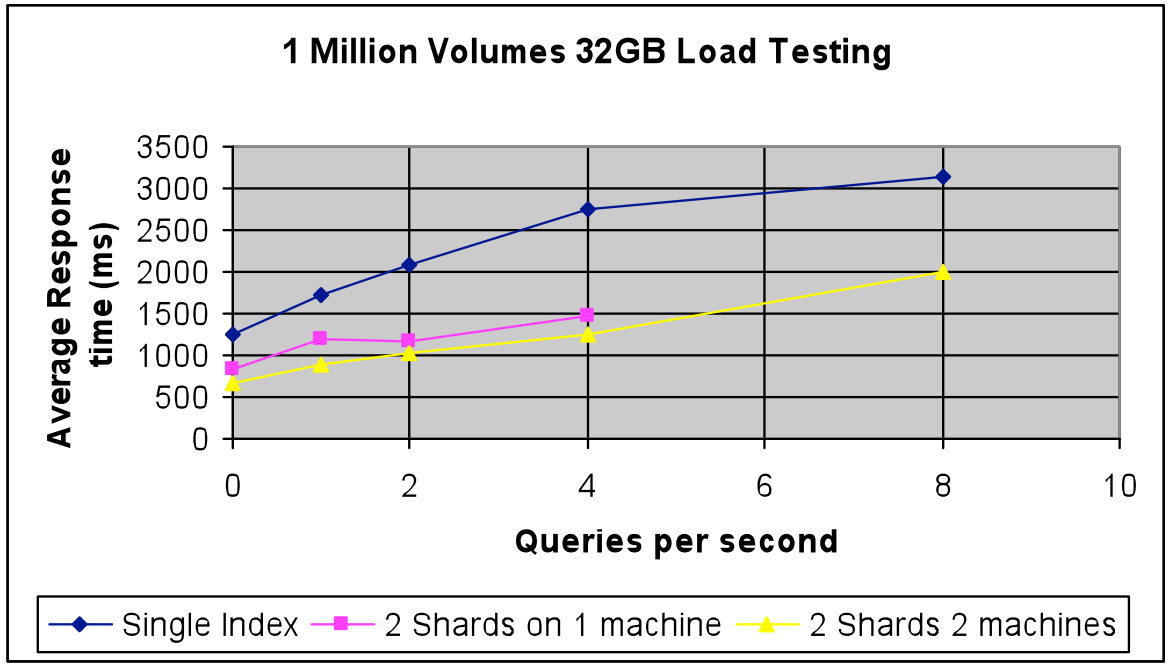
- Solr scales linearly. At 32 GB total memory there is an increase in slope above 600 thousand documents.
- Best response time was for 2 shards each on its own machine



Testing Program Results

Load testing and Machine Configurations

- Request rates over 1 query per second increased response time.
- Rates over 4 qps were not sustainable at larger index sizes.
- 2 shards on 2 machines handled load best
- Above 600 thousand documents response time increases rapidly

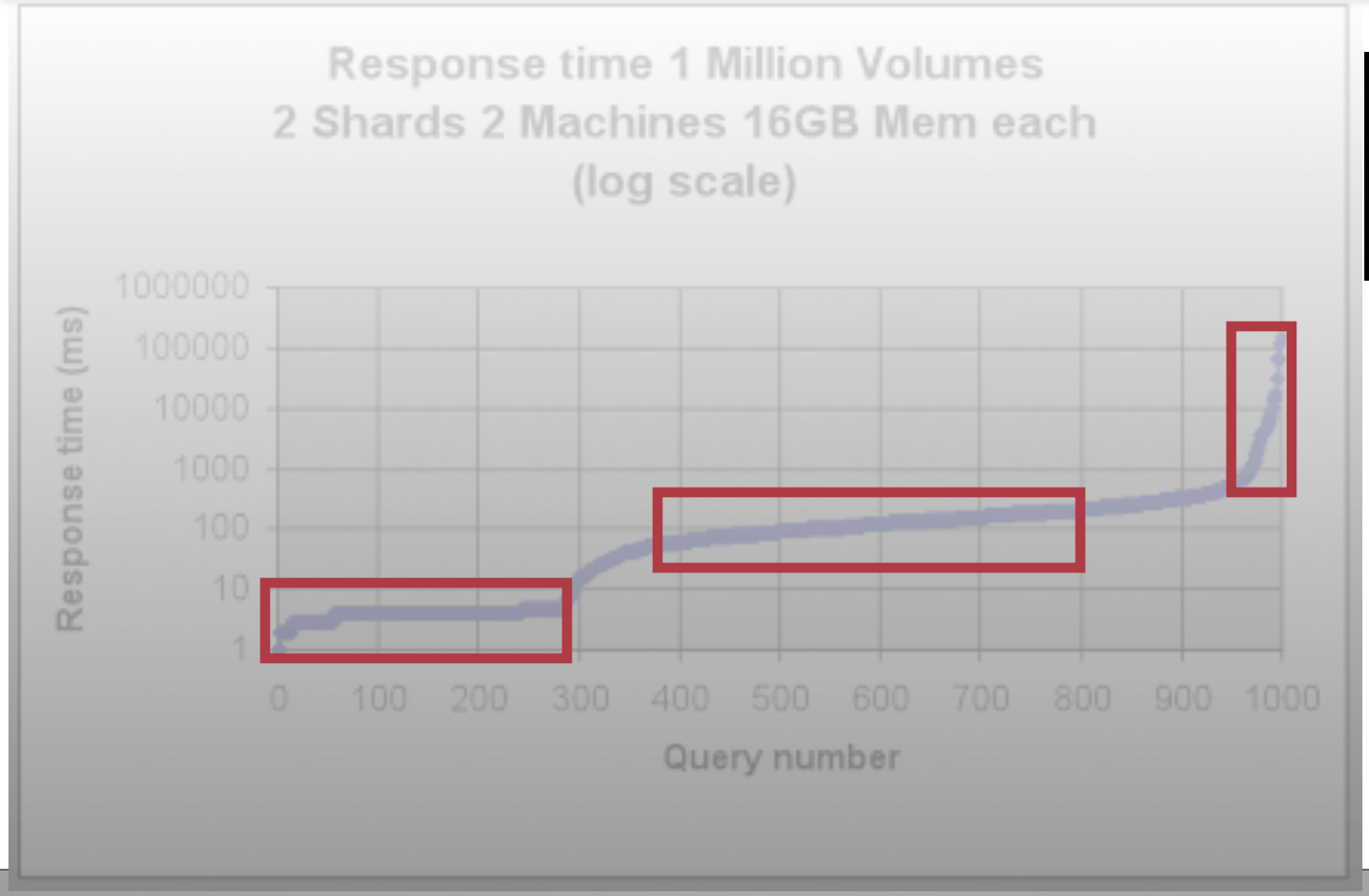


Testing Program Results

Caching and Memory

- Solr uses OS memory for caching of postings
- Memory available for caching has most impact on response time
- Based on the 1 Million volume index size of 240GB, the index for 7 million documents would be 1.6TB
- Fitting entire index in memory not feasible with terabyte size index

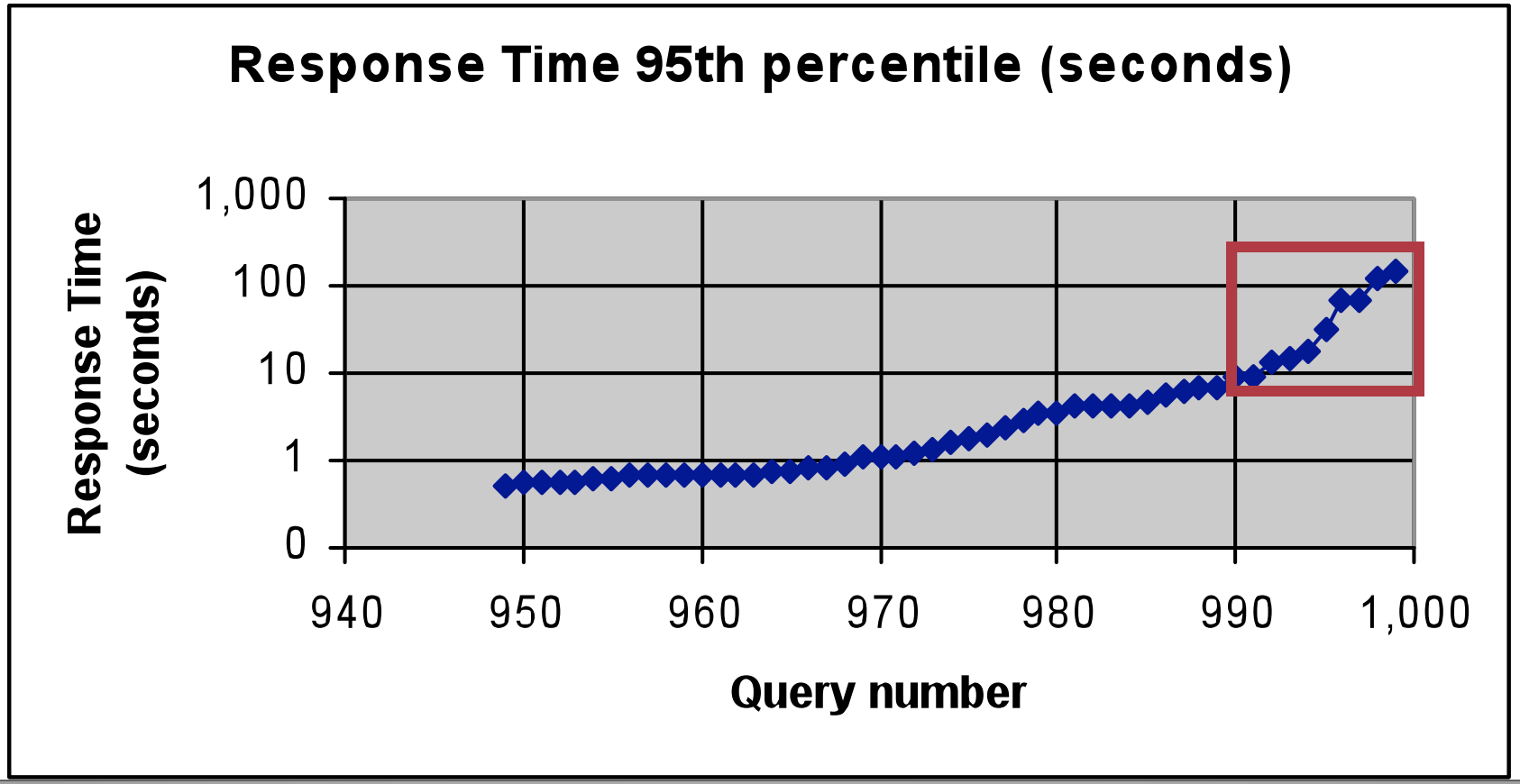
Response time varies with query



Average:	673
Median:	91
90th:	328
99th:	7,504

www.hathitrust.org

Slowest 5% of queries



Slow Queries

- The slowest 1% of queries took between 10 seconds and 2 minutes.
- Slowest 0.5% of queries took between 30 seconds and 2 minutes
- These queries affect response time of other queries
  - Cache pollution
  - Contention for resources
- Slowest queries are phrase queries containing common words

Query processing

- Phrase queries use position index (Boolean queries do not).
- Position index accounts for 85% of index size
- Position list for common words such as "the" can be many GB in size
- This causes lots of disk I/O .

Query processing

- Solr depends on the operating systems disk cache to reduce disk I/O requirements for words that occur in more than one query
- I/O from Phrase queries containing common words pollutes the cache

Query Processing: Postings List

Doc ID	Content
1	The black dog chased the white cat
2	The hungry dog ate the cat's food
3	The cat sat in the sun.
4	The dog buried a bone.

Word	Doc IDs
a	4
ate	2
black	1
bone	4
buried	4
cat	1,2,3
chased	1
dog	1,2,4

Word	Doc IDs
food	2
hungry	2
in	3
sat	3
sun	3
the	1,2,3,4
white	1

Query Processing: Position List

Doc ID	Content
1	The black dog chased the white cat 1 2 3 4 5 6 7
2	
3	
4	The dog buried a bone.

Word	(Doc ID) positions
a	(4) 4
ate	(2) 4
black	(1) 2
bone	(4) 5
buried	(4) 3
cat	(1) 7, (2) 6,(3) 2
chased	(1) 4
dog	(1) 3,(2) 3, (4) 2

Word	(Doc ID) Positionss
food	(2) 7
hungry	(2) 2
in	(3) 4
sat	(3) 3
sun	(3) 3
the	(1) 1, 5, (2) 1, 5 , (3) 1,5, (4) 1
white	(1) 6

Query Processing

- "Dog" AND "Cat"

cat	1, 2, 3
dog	1, 2, 4

"Dog" AND "Cat"

Docs: 1,2

"The Cat" (Phrase)

the	1, 2, 3, 4
cat	1, 2, 3

"The" AND "Cat"

Docs: 1,2,3

the	(1) 1, 5, (2) 1, 5, (3) 1, 5, (4) 1
cat	(1) 7, (2) 6, (3) 2

"The Cat"

Docs: 2,3

Slow Queries

- Slowest test query: “the lives and literature of the beat generation” took 2 minutes.
- 4MB data read for Boolean query.
  9,000+ MB read for Phrase query.

WORD	NUMBER OF DOCUMENTS	POSTINGS LIST (SIZE MB)	TOTAL TERM OCCURRENCES (MILLIONS)	POSITION LIST (SIZE MB)
the	800,000	0.8	4,351	4,351
of	892,000	0.89	2,795	2,795
and	769,000	0.77	1,870	1,870
literature	435,000	0.44	9	9
generation	414,000	0.41	5	5
lives	432,000	0.43	5	5
beat	278,000	0.28	1	1
TOTAL		4.02		9,036

Why not use Stop Words?

- The word “the” occurs 4 billion times; average of 15,000 times per document in about 80-90% of all documents.
- Removing “stop” words (“the”, “of” etc.) not desirable
- Couldn’t search for many phrases
  - “to be or not to be”
  - “the who”
  - “man in the moon” vs. “man on the moon”

Why not use Stop Words?

- Stop words in one language are content words in another language
- German stopwords “war” and “die” are content words in English
- English stopwords “is” and “by” are content words (“ice” and “village”) in Swedish

“CommonGrams”

- Bi-Grams triple index size
- Nutch and CDL XTF implement “CommonGrams”
- Create Bi-Grams for any two word sequence containing common terms
- “The rain in spain falls mainly” = “the-rain” “rain-in” “in-spain” “falls” “mainly”

“CommonGrams”

- Ported Nutch “CommonGrams” algorithm to Solr
- Create Bi-Grams selectively for any two word sequence containing common terms

CommonGrams Example

- Slowest query: “The lives and literature of the beat generation”
  - “the-lives” “lives-and”
  - “and-literature” “literature-of”
  - “of-the” “the-beat” “generation”

Standard index vs. CommonGrams

Standard Index

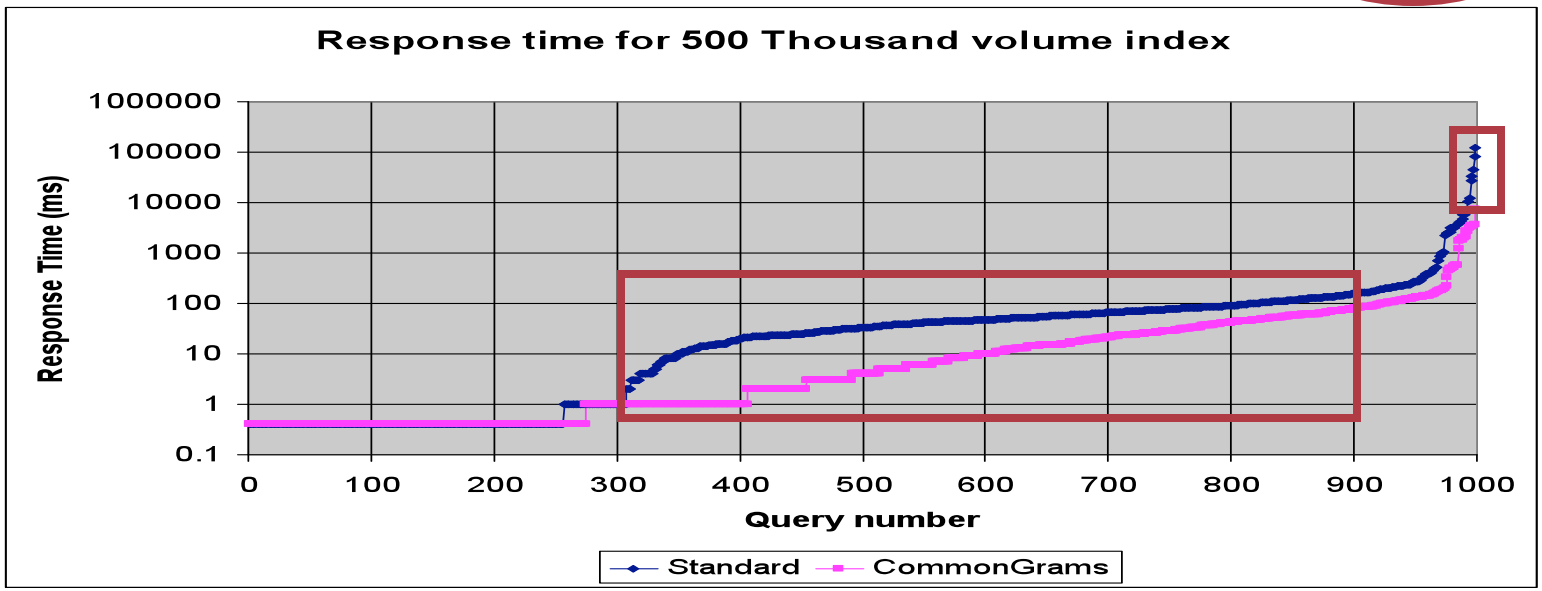
WORD	TOTAL OCCURRENCES IN CORPUS (MILLIONS)	NUMBER OF DOCS (THOUSANDS)
the	2,013	386
of	1,299	440
and	855	376
literature	4	210
lives	2	194
generation	2	199
beat	0.6	130
TOTAL	4,176	

Common Grams

TERM	TOTAL OCCURRENCES IN CORPUS (MILLIONS)	NUMBER OF DOCS (THOUSANDS)
of-the	446	396
generation	2.42	262
the-lives	0.36	128
literature-of	0.35	103
lives-and	0.25	115
and-literature	0.24	77
the-beat	0.06	26
TOTAL	450	

CommonGrams
Comparison of Response time (ms)

	average	median	90th	99th	slowest query
Standard Index	459	32	146	6,784	120,595
Common Grams	68	3	71	2,226	7,800



Scaling up to 5 million+

- Solr distributed search 10 shards, 500K docs each
- Tried local storage but ran into contention between indexing/optimizing and serving searches
- Moved to separate build and serve machines and shared storage with the Isilon

We Broke Solr!

- At around 500,000 documents (300GB index size) we started getting this error:

  java.lang.ArrayIndexOutOfBoundsException: -14127432 at org.apache.lucene.index.TermInfosReader.get(TermInfosReader.java:246)

- Dirty OCR in combination with over 400 languages created indexes with over 2.4 billion unique terms
- Lucene/Solr had a limit of 2.1 billion unique terms per index.

We broke Solr!

- We wrote to Solr list and Mike McCandless patched Lucene to raise the limit to 274 Billion
- Dirty OCR is difficult to remove without removing “good” words.
- Because Solr/Lucene tii/tis index uses pointers into the frequency and position files we suspect that the performance impact is minimal compared to disk I/O demands, but we will be testing soon.

Current System

- Solr distributed search. Split index into 10 shards
- Currently about 6.5 million docs and total index size is about 3 terabytes.
- About 650,000 docs (300GB index) per shard over 4 serving machines
- Two dedicated indexing machines
- Isilon shared storage

Overall Architecture (1)

- VuFind catalog (Solr index)
  - Source of update information and bibliographic metadata large scale search indexing
- SLIP (Solr Large-scale Indexing processor)
  - Determines which volumes should be indexed
  - Builds documents to index
  - Sends these to Solr for indexing

Overall Architecture (2)

- Index
  - 10 shards
  - 6 machines
    - 2 build index
      - 6 Solr instances on each machine
      - Each writes to one shard
    - 4 serve index
      - 3 Solr instances on each machine
      - Each queries one shard
  - Stored on Isilon Storage
    - Index is built, optimized, snapshot taken, and mounted

Overall Architecture (3)

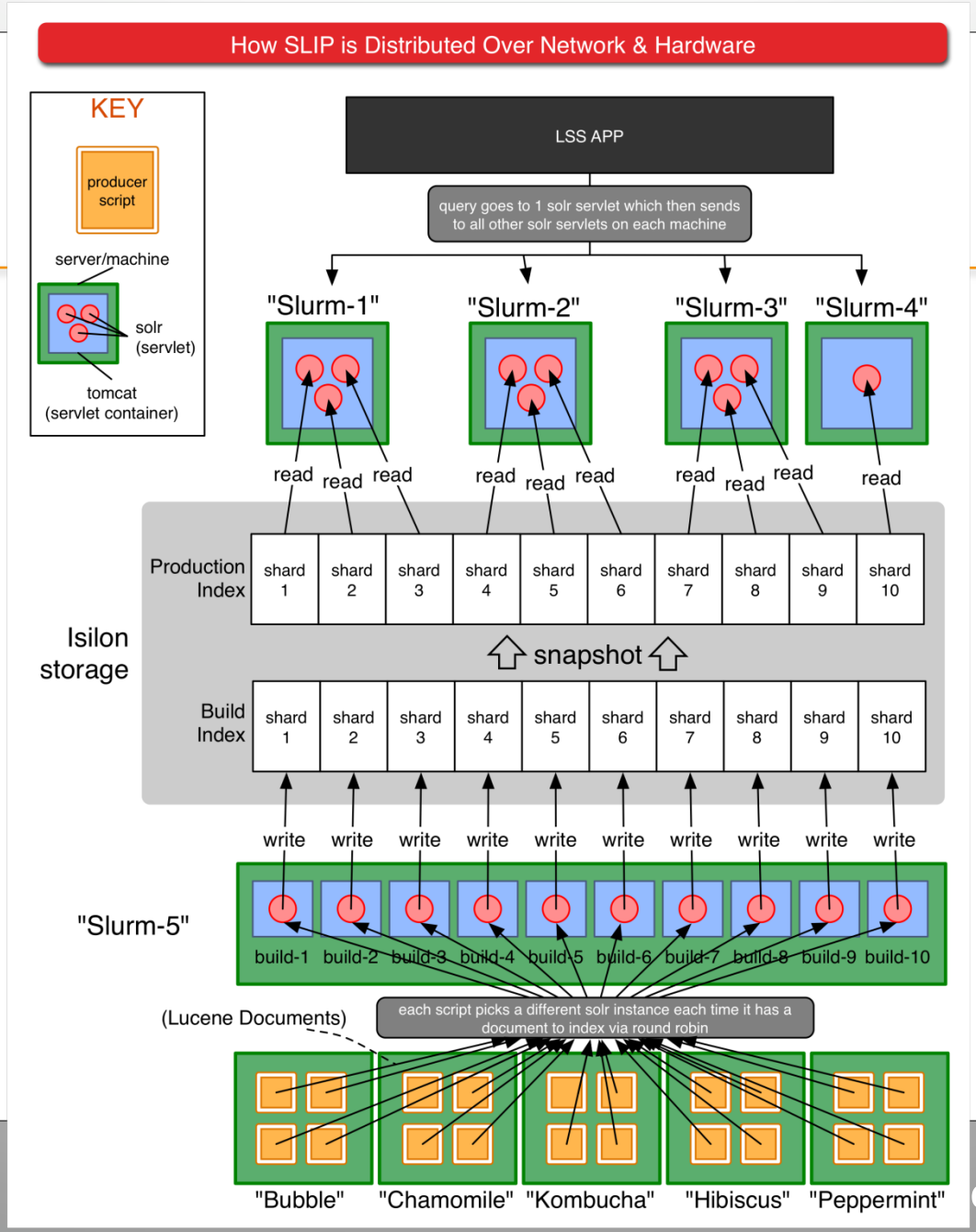
- Application/UI
  - Sends queries in random fashion to one of the 10 Solr instances
    - That instance distributes query to the other nine and merges the results
- Connection with PageTurner (search inside a single book, XPAT)
- Mirroring at IU

Hardware (1)

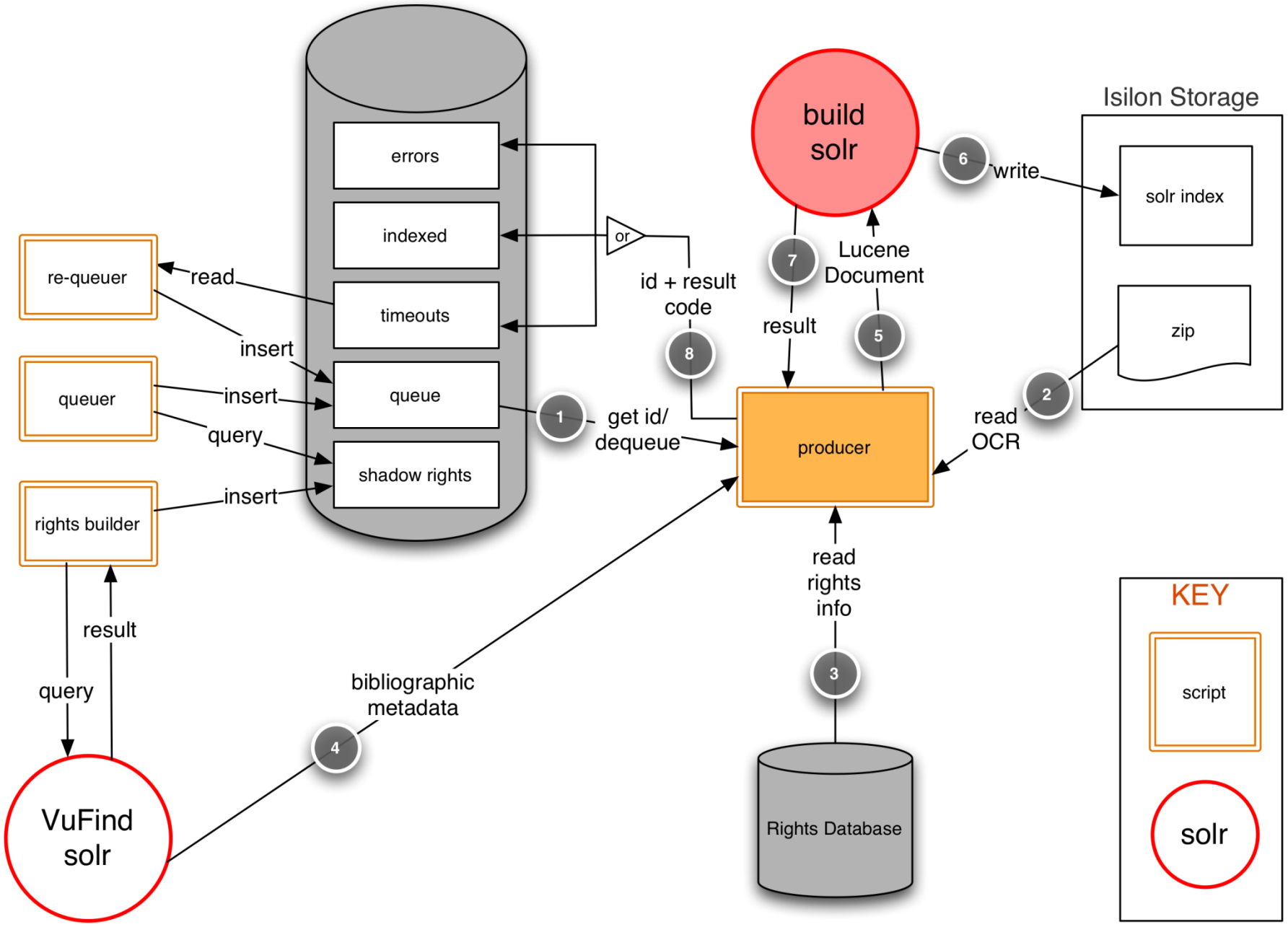
- Solr Server configuration
  - Dell PowerEdge R710
  - 2 x Quad Core Intel Xeon E5540 2.53GHz processors (Nehalem)
  - 72 GB RAM
  - Red Hat Enterprise Linux 5.4 (kernel: 2.6.18 X86_64)
  - Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
  - Solr 1.3.0.2009.09.03.11.14.39 (1.4-dev 793569)
  - Tomcat 5.5.27

Hardware (2)

- Storage
  - Isilon IQ NAS cluster (20 I/X-series nodes, 4 GB RAM per node)
  - 480 750GB or 1TB SATA drives providing 420 TB raw storage
  - 4GB RAM per node giving 80 GB of coherent cache in aggregate
- Network
  - NFS uses a dedicated/private 9K MTU GbE network on Dell PowerConnect 5448 switch
  - NFS clients single-homed and mounts automatically distributed across all cluster nodes


How SLIP is Distributed Over Network & Hardware
KEY
producer script
server/machine
solr (servlet)
tomcat (servlet container)
LSS APP
query goes to 1 solr servlet which then sends to all other solr servlets on each machine
"Slurm-1"
"Slurm-2"
"Slurm-3"
"Slurm-4"
read
Isilon storage
Production Index
shard 1
shard 2
shard 3
shard 4
shard 5
shard 6
shard 7
shard 8
shard 9
shard 10
snapshot
Build Index
write
"Slurm-5"
build-1
build-2
build-3
build-4
build-5
build-6
build-7
build-8
build-9
build-10
(Lucene Documents)
each script picks a different solr instance each time it has a document to index via round robin
"Bubble"
"Chamomile"
"Kombucha"
"Hibiscus"
"Peppermint"

SLIP Data Flows



Bottlenecks, Problems and Challenges Indexing (1)

Problem: Scalability of page level indexing (1.5 billion pages) unknown

- Solution: Index full documents in Solr. When user clicks on document, do on-the-fly indexing to provide page level within-document searching

Bottleneck: Assembling Solr documents from files in repository slow

- Solution: Multiple Solr document producers running on multiple machines

Bottleneck: Full optimization of nightly index builds takes too long

- Solution: Nightly optimize to 2 segments. When appropriate optimize to 1 segment

Bottlenecks, Problems and Challenges Indexing (2)

Problem: Indexing program must handle errors and timeouts from multiple cooperating processes

- Solution: errors recorded for manual investigation
- Solution: some fix themselves, others are due to bad data
- Solution: timeouts automatically re-queued, gives a check on balance between document creation vs. indexing rates

Bottleneck: Optimizing index takes more disk space than available

- Solution: Optimize in stages
- Solution: Move to NAS with unlimited storage

Bottlenecks, Problems and Challenges Search Performance

- **Bottleneck:** Solr relies on memory based OS disk caching for performance Our index is too big to fit in memory (400-600GB per million documents).
  - Solution: Spread index over several Solr instances on several machines
  - Solution: Increase memory on each machine
- **Bottleneck:** Slowest 1% of queries have unacceptable performance due to high disk I/O
  - Solution: Reduce disk I/O requirements using CommonGrams and Punctuation Filter
  - Solution: Run cache-warming queries
- **Problem:** Use of truncation operator results in response time over 5 minutes for some queries and impacts performance of subsequent queries
  - Solution: Prohibit multiple letter truncation operator

Bottlenecks, Problems and Challenges General Scalability issues

- **Bottlenecks:**
  - Performance of un-optimized index too slow
  - Optimization takes multiple hours due to size of index
  - Optimization on Solr instance serving queries degrades query response time
  - Solution: Use separate Solr instances for indexing/optimizing and serving queries
- **Bottleneck:** Optimized index too large to efficiently copy from build to serve instances
  - Solution: Use LVM snapshots
- **Bottleneck:** Indexing and serving instances sharing RAID array compete for disk I/O during optimization.
  - Solution: Move storage to high performance NAS (Isilon)
  - Solution: Use NAS file-based snapshots

Bottlenecks, Problems and Challenges Other Challenges

- Malformed queries and Boolean Operators
- Transition from repository search to single-volume search
- Syncing Indexes
  - Bibliographic index, full-text index, rights database

Next Steps

- Investigate further optimizations
- Investigate facets and fielded search
- Integrate with other HathiTrust applications such as Collection Builder

Possible Future Development

- Investigate relevance ranking
  - Boost rank for words occuring in MARC subject, author, title fields.
  - Investigate document size normalization
- Improve multilingual access and retrieval
- Investigate page level indexing and integration with book indexing

Possible Future Development

- Investigate user interface/user interaction issues
  - How best to display large result sets
  - How can users best narrow their search?
  - Integration with user's tasks

thank you!

- http://catalog.hathitrust.org
- http://www.hathitrust.org/blogs