
Henry
Sebastien
Pedro
Joseph
Tristan
Ian
I DREAM IN CODE
THE LAST
Search Engine You Will Ever Need

The Last Search Team

- **Tristan Kellar** is a **senior** studying **Computer Science**
- **Henry Meng** is a **junior** studying **Computer Science**
- **Sebastien Rivest** is a **senior** studying **Computer Science**
- **Joseph Schuman** is a **senior** studying **Computer Science**
- **Ian Woodfill** is a **senior** studying **Computer Science**
- **Pedro Zamora** is a **junior** studying **Computer Science**

The Last Search Team Cont.

Crawler hits 1 million (11:45 PM EST)



Crawler Crashes at 1.1 Million (11:46 PM EST)



Team Statistics



- 827 Commits
- 52 Pull Requests
- 5 Repositories



- 5,554 Slack Messages Sent
- 81% in private channels 👀



- Eastern Standard Time (5)
- Pacific Standard Time (1)

Lines of Code Statistics

WORKER	# Files	# LOC
C++	21	2,619
C++ Header	16	722
Make	5	126
Total	42	3,467

UTILITIES	# Files	# LOC
C++ Header	16	1,206
C++	11	503
Make	1	70
Total	28	1,779

MANAGER	# Files	# LOC
C++	4	931
CSS	1	191
Python	1	103
Bash	4	97
JavaScript	1	90
C++ Header	2	88
HTML	1	76
Make	1	33
Total	16	1,609

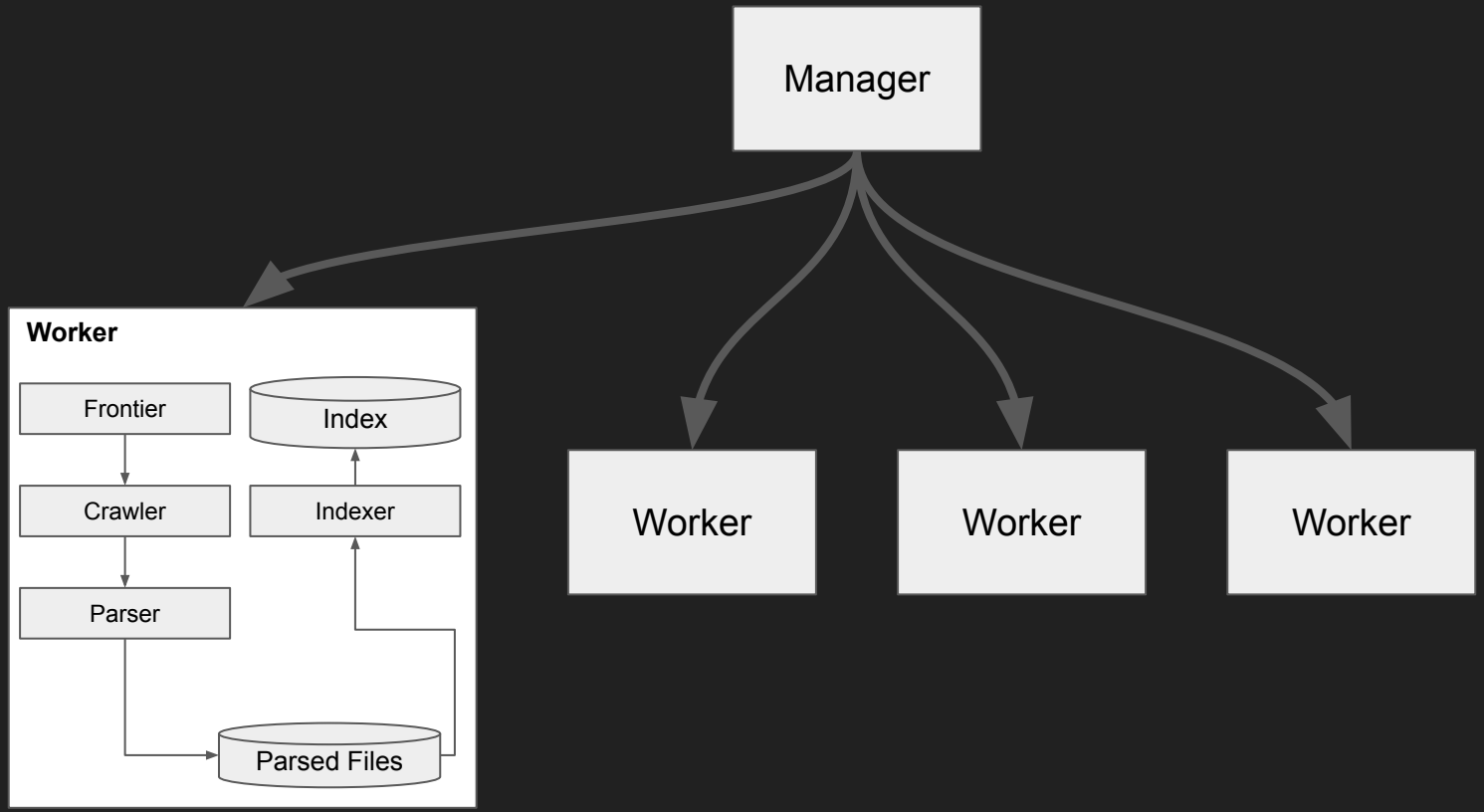
TOTAL LOC	6,855

30,421,706 LOC including blanks, comments, and test files

High Level Components

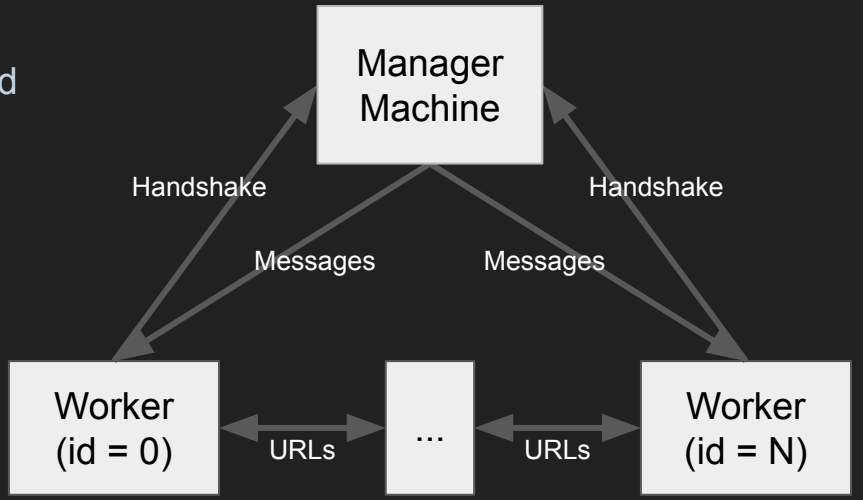
<table>
<tr><td colspan="6">Search Engine</td></tr>
<tr><td colspan="3">Building the Index</td><td colspan="3">Querying</td></tr>
<tr><td colspan="2">Crawler</td><td>Indexer</td><td>Query Compiler</td><td>Constraint Solver</td><td>Ranker</td></tr>
<tr><td>Robots.txt</td><td>Frontier</td><td>Parser</td><td colspan="3">Index Chunks</td></tr>
</table>

Building a Distributed Index



Manager-Worker Communication

- Handshake
  - Worker notifies Manager that is is connected
- Messages (Manager → Worker)
  - Start/Stop Crawling
  - Start/Stop Indexing
  - Query Request
  - Request Status Update
- Messages (Worker → Worker)
  - URL from Coworker



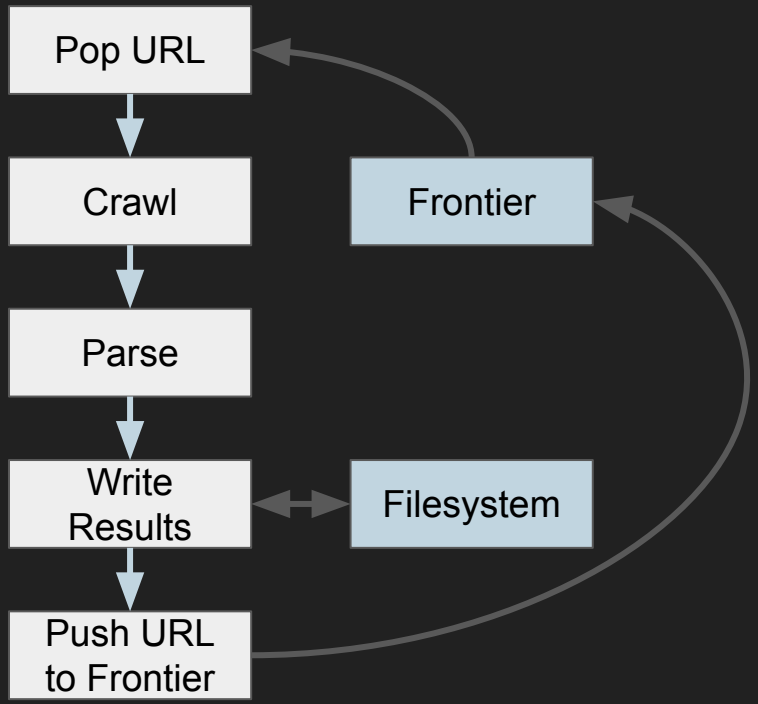
Manager tells workers to start Crawling

```
Select a manager option:
1: StartCrawling: tell the connected worker machines to begin crawling
2: StopCrawling: tell the connected worker machines to stop crawling
3: StartServer: initialize the webserver
4: See crawling statuses
5: StartIndexing
6: StopIndexing
7: See indexing statuses
8: Send UpdateManager to Workers

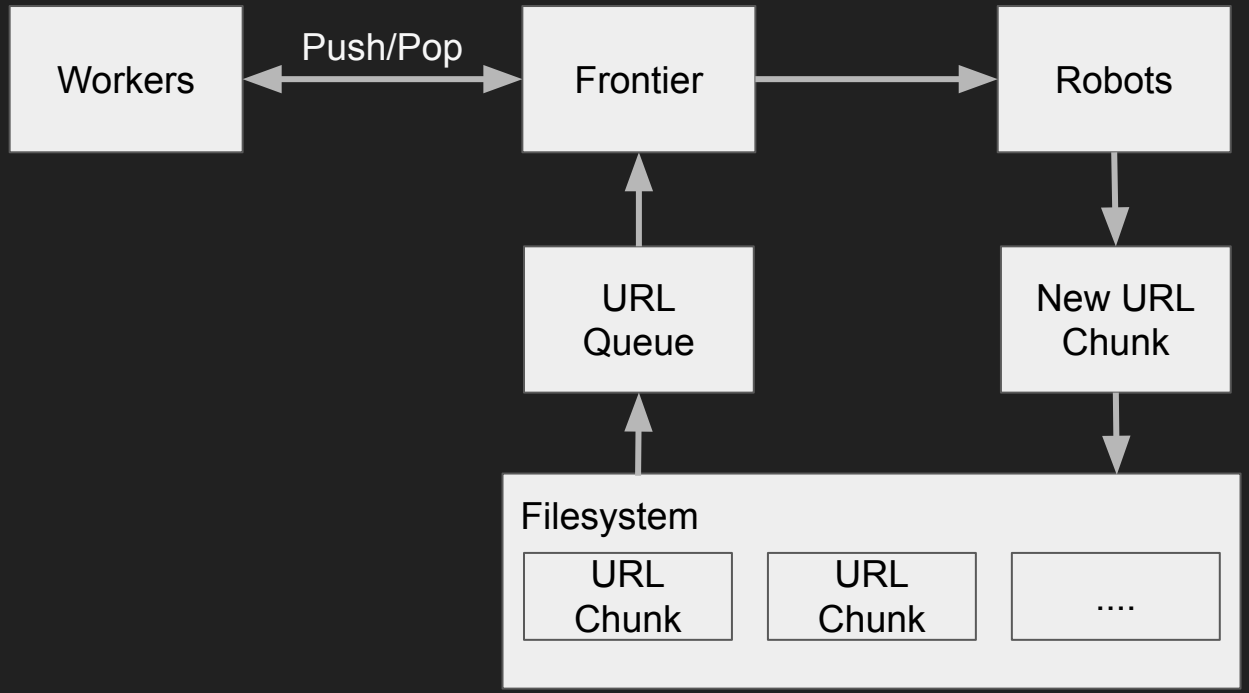
[PROD] (manager/ManagerCommunicator.cpp:91) Manager listening for messages from workers on port: 8000
[PROD] (manager/ManagerCommunicator.cpp:94) Status updates will not be printed automatically. Use command (4) or (7) to get updates
[PROD] (manager/ManagerCommunicator.cpp:288) New worker connected
        IP: localhost
        Port: 9000
1

Enter machine ( all / pedro / henry / custom ):
all
PERFORMING ACTION ON MACHINE LIST:
0
[PROD] (manager/ManagerCommunicator.cpp:442) Start crawling message sent
```

Each free Crawler thread consumes a prioritized url from the Frontier



Each URL is added to the Frontier, checking robots.txt before pushing



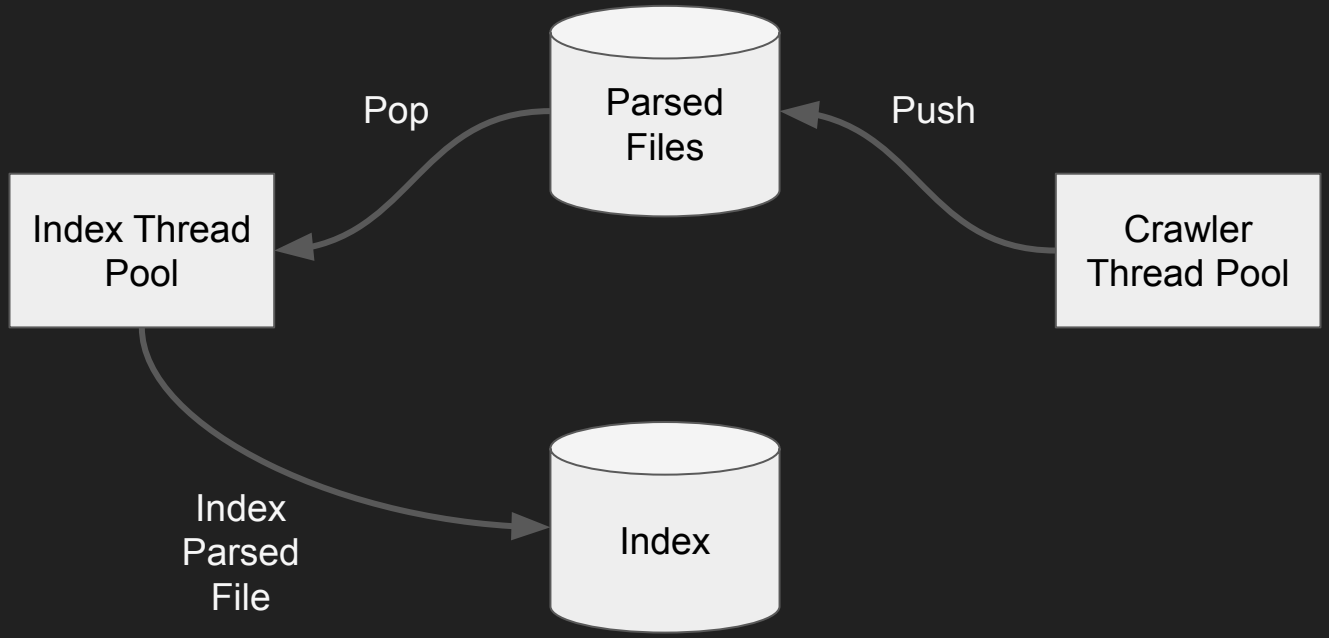
Manager tells Worker to start Indexing

```
Select a manager option:
1: StartCrawling: tell the connected worker machines to begin crawling
2: StopCrawling: tell the connected worker machines to stop crawling
3: StartServer: initialize the webserver
4: See crawling statuses
5: StartIndexing
6: StopIndexing
7: See indexing statuses
8: Send UpdateManager to Workers

[PROD] (manager/ManagerCommunicator.cpp:91) Manager listening for messages from workers on port: 8000
[PROD] (manager/ManagerCommunicator.cpp:94) Status updates will not be printed automatically. Use command (4) or (7) to get updates
[PROD] (manager/ManagerCommunicator.cpp:288) New worker connected
        IP: localhost
        Port: 9000
5

Enter machine ( all / pedro / henry / custom ):
all
PERFORMING ACTION ON MACHINE LIST:
0
[PROD] (manager/ManagerCommunicator.cpp:463) Start indexing message sent
```

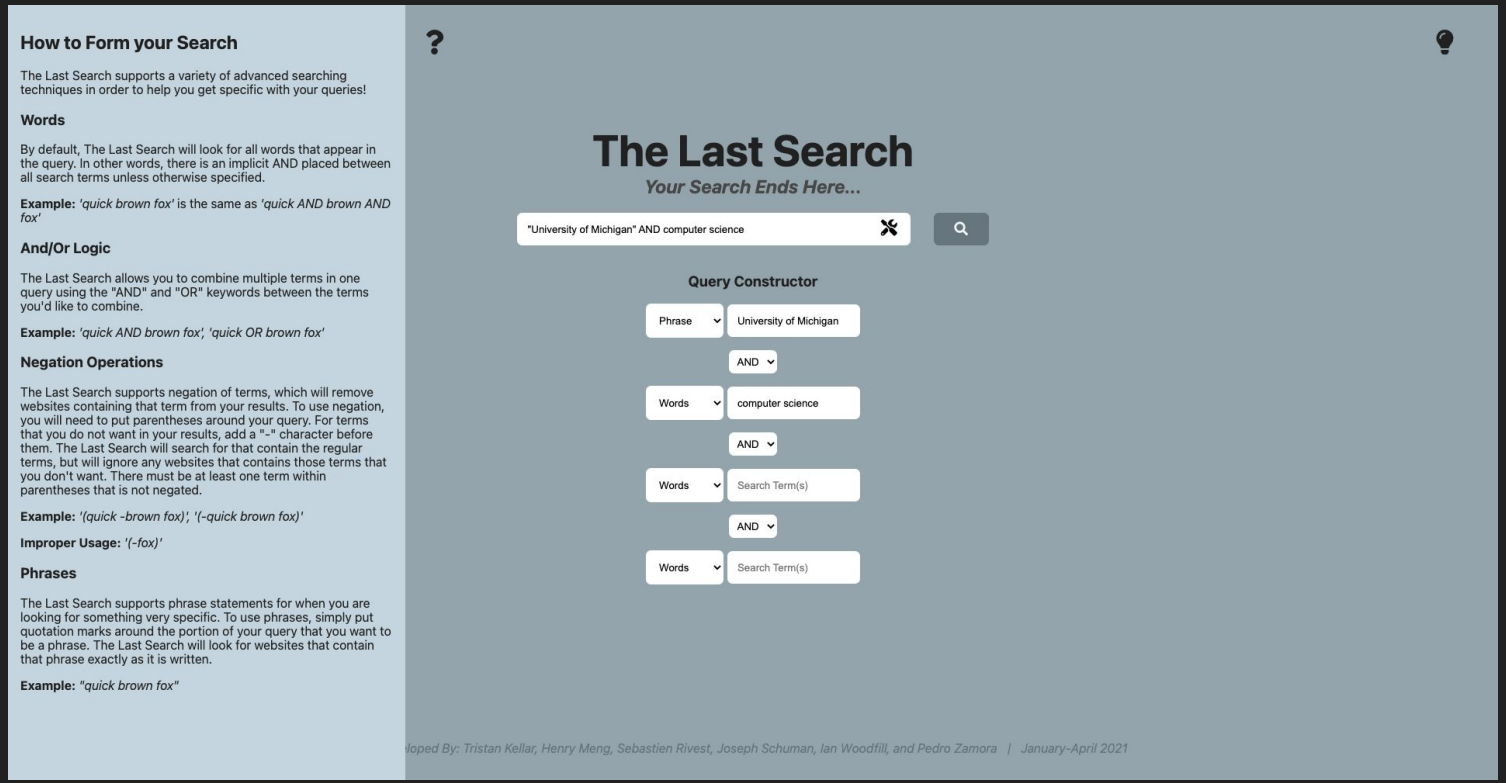
Each free Index thread consumes a Parsed File



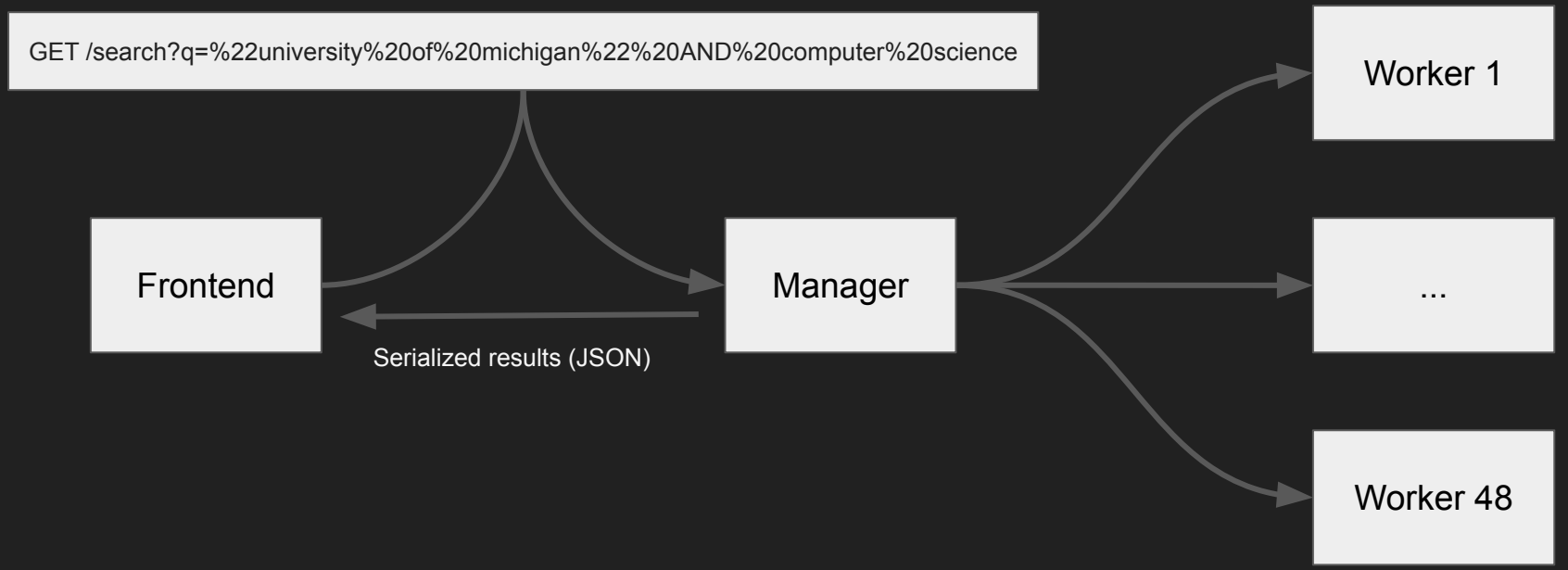
Index Statistics

- Crawled 191.4 Million Pages
- Indexed 144.1 Million Pages
- 48 Worker Machines
  - Google Compute Engine
  - 4x e2-standard-2 (2 vCPU, 8GB RAM)
  - 12x GCP Free Trial Accounts
- 1 Manager Machine
  - AWS EC2
  - 1x t2.xlarge (4 vCPU, 16GB RAM)

It's time to search!



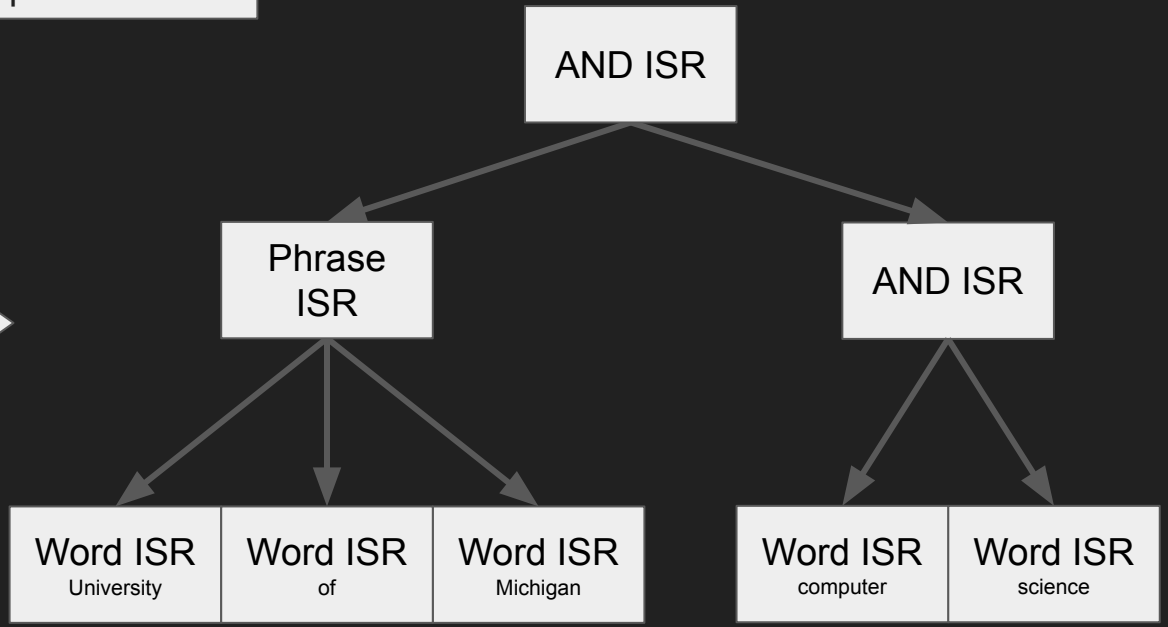
Manager broadcasts query to workers and waits for their response



Each worker compiles the query

"University of Michigan" AND computer science





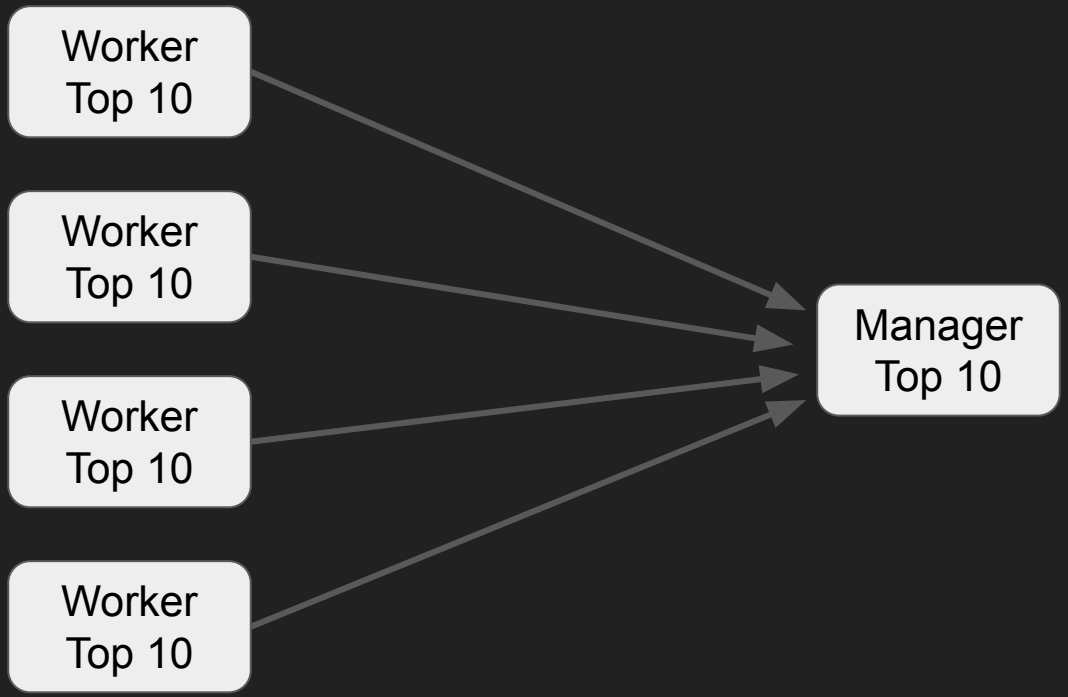
Relevant Documents are Ranked

- Static Ranking Factors
  - Query Words in the URL
  - URL Domain Length
  - Top Level domain (.edu, .com)
  - HTTPS vs HTTP
  - Prioritize fewer slashes in URL
- Dynamic Ranking Factors
  - Spanning

Snippets are created for Top N results

- Implementation
  - For each result, determine the Snippet Bounds
    - Seek “dominant” query term ISR to the first appearance in the document
    - Determine snippet LHS, RHS
  - Search Index Chunk for all posts that appear in these bounds
    - Add to vector Term and Position
  - Quick Sort position to retain the original order
  - Add snippet to Result
- Challenges
  - Casing parsed files
  - Poorly parsed files
  - Decorated terms

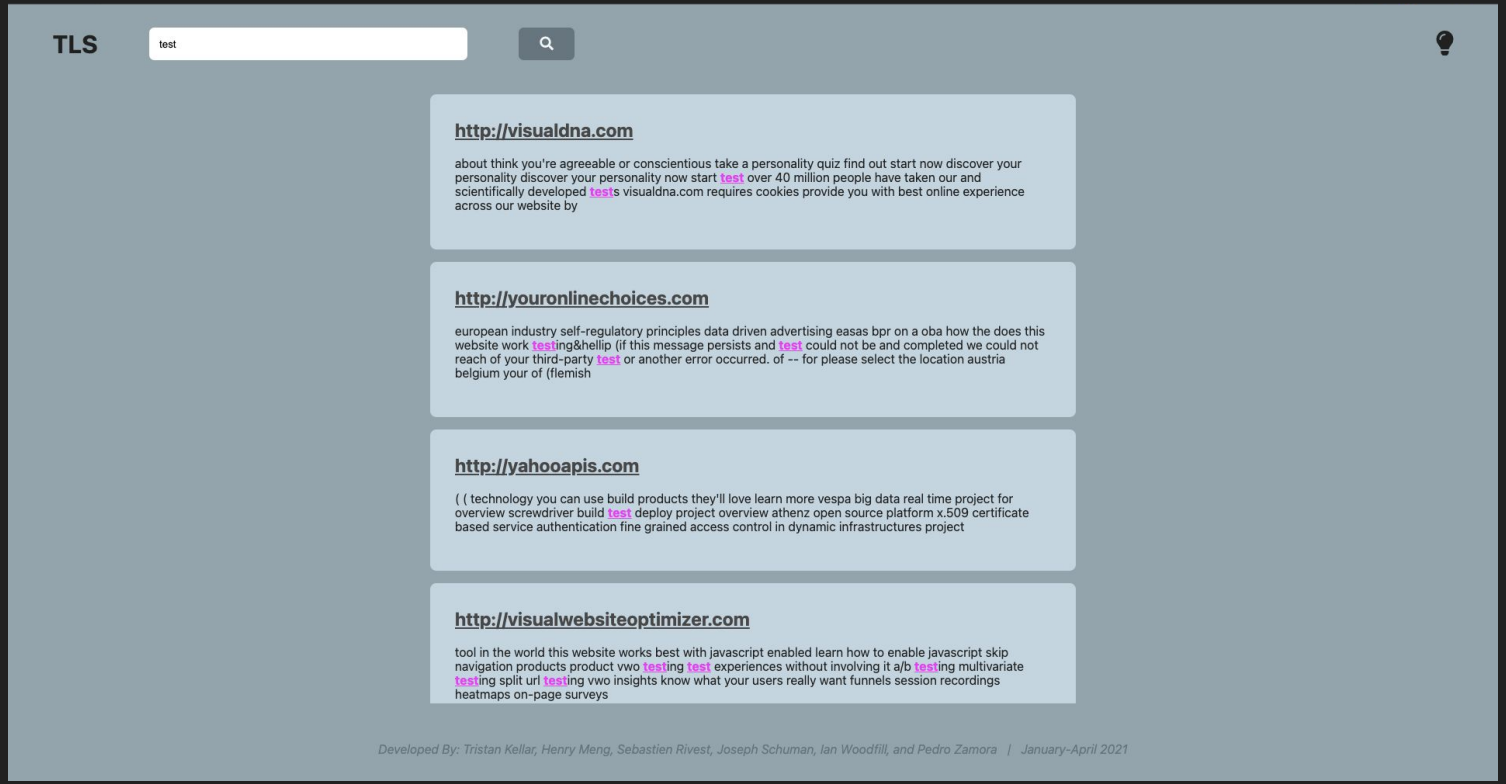
Results w/ normalize scores are serialized and sent to Manager



Manager receives and combines results into JSON object



Frontend displays Top N results



Project Management

- Standup
  - Kanban
  - Stories
  - Blockers
  - Weekly Goals
- Slack
- Discord
- Pair Coding
- Version Control
  - Code Reviews
- Testing

Conclusion

- Gained Experience
  - Systems Design at Scale
  - Largest OOP C++ Project
  - Mmap’ing
  - Version control
    - Submodules
  - Development Vs Production
  - Dev-Ops
  - Parallelization and Concurrency
- What’s next?
  - Improve Dynamic Ranking
  - Optimize Robots Parsing
  - Better support of concurrent queries

Thanks for a wonderful semester!

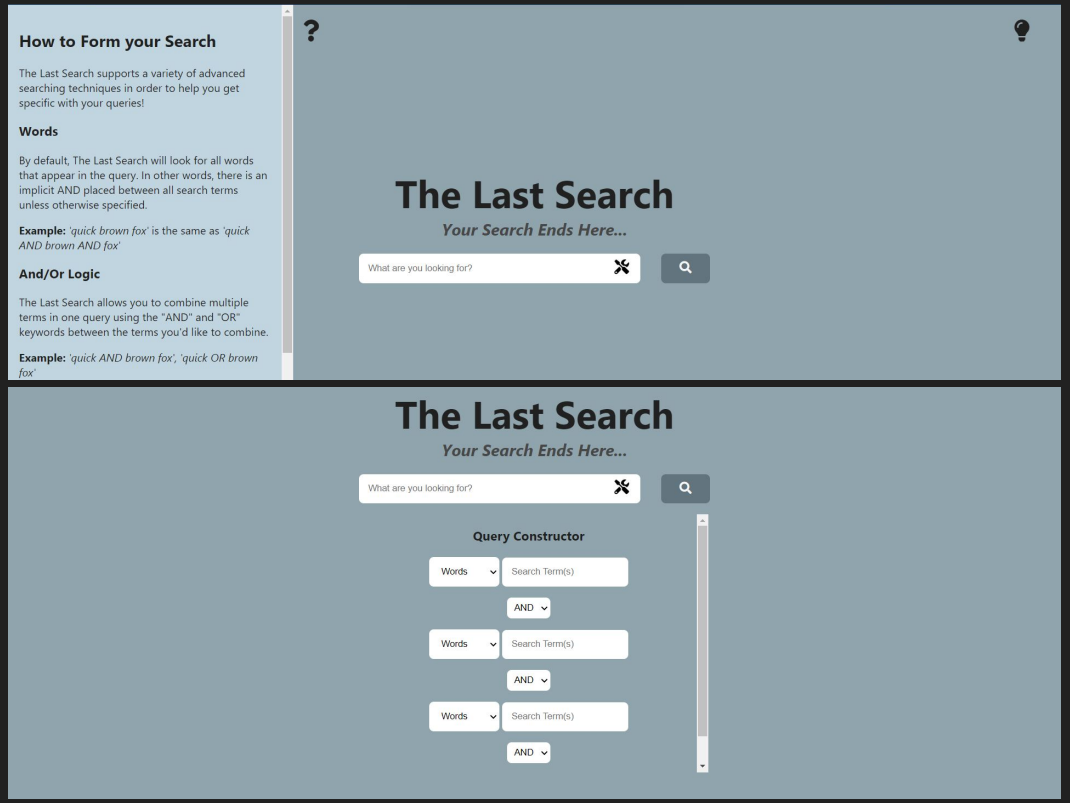


http://lastsearch.org

Appendix Slides

Frontend Usability

- Created a help tool to inform users on how to properly use advanced query functions
- Created a query constructor that allows users to use advanced functions without typing



Query Handling and Results

- Query compiler/Constraint solver supported query operations
  - Or/And, Phrase, Container, Not
- Same ISR structures used by Compiler/Constraint Solver/Ranker
- Constraint solver receives ISR tree from compiler and calls ISR find methods on indexed pages
- Ranker utilizes linear combination of multiple techniques to rank:
  - Static Factors: Query words in url, url domain length, is HTTPS?
  - Dynamic Factors: Spanning, phrase matching
- Each machine produces its own top 10 results, and the manager determines the top 10 from those

Snippets + ISRs

- ISR infrastructure was extremely solid
- Snippets implementation
  - Determine Snippet Bounds
    - Seek “dominant” query term ISR to the first appearance
    - Determine snippet LHS, RHS
  - Search Current Index Chunk for all posts that appear in these bounds
    - Add to vector Term and Position
  - Quick Sort position to retain correct order

Communication Difficulties

- Problem
  - Worker machines could receive URLs from coworkers too quickly, bogging crawling
- Idea
  - Throttle outgoing messages to coworkers
  - Throttle incoming messages from coworkers
- Solution
  - Listen for messages from coworkers on a different port from manager
  - Maintain a limited threadpool that manages incoming/outgoing messages from/to coworkers

Crawling and Indexing at Scale

- Problem
  - Crawler crashing often - segfaults, memory leaks, more
  - Babysitting machines takes >14 hours/day
  - When the Crawler Crashes we lose all current in-memory Index Chunks
- Solution
  - Worker process automatically restarts on crash
  - Automated Google Cloud scripts for restarting and updating worker machines
  - Index Chunks size halved so we write more chunks often, Crawling and Indexing at separate times, fewer indexing threads

Query Support

- Problem: We wanted to include support of not operations on terms in queries, but building ISR Container support was challenging
- Solution: Allow containers to be explicitly defined by users on the frontend by using parentheses

(quick -brown fox)

- This allowed us to easily support not operators in our queries, and the Frontend added features to aid users in using this functionality