Enterprise Search Engine for Logs

Vineet Gangwar and Anant Srivastava

Introduction

Why Search Engine for Logs:

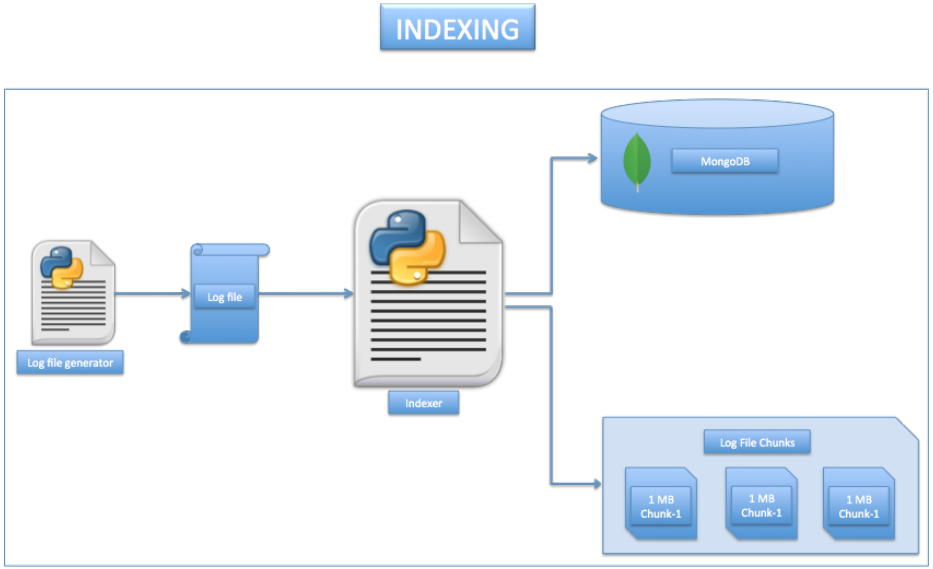
- Log processing is one of the central use cases of Big Data
- Logs are increasingly considered as a Gold Mine
- Base platform for big data apps - IT Ops, IT Security Mgmt
- Storing and Retrieving data ~ Search Engine

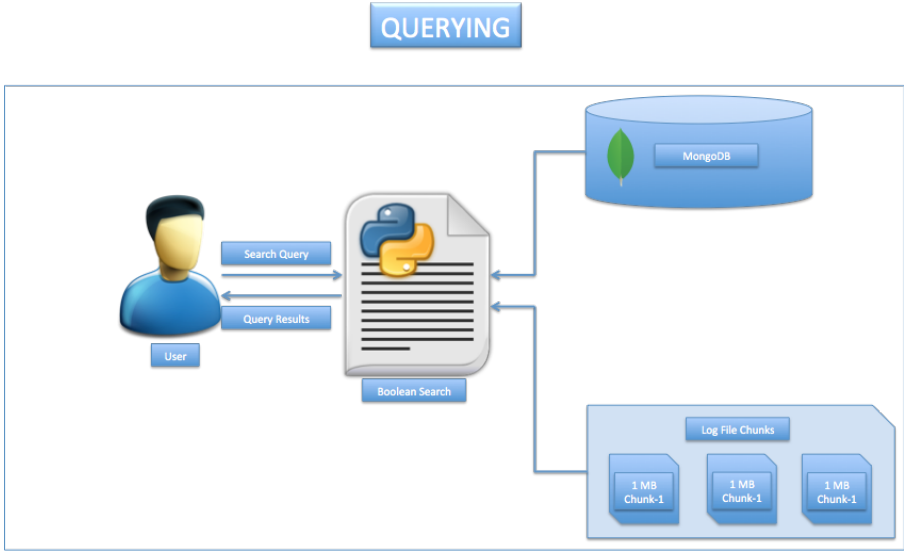
Objectives:

Create a handy tool for a sys admins using ***Inverted Index***

- Single System Implementation
  - Provide a single-pane-of-glass for searching logs
  - Relatively current logs from many sources and systems
  - Provide speeds better than traditional `cat | grep`
- Hybrid System Implementation
  - Use cloud technologies to scale up:
    - Processing
    - Storage

Single System - Implementation





Postings list:

pList = {Document Id, Offset, Log length}

MongoDB Collection Structure:

{Term: term1, postingsList: pList1}

{Term: term1, postingsList: pList2}

...

Bulk inserts into MongoDB

Query Language:

Implemented using ***Shunting Yard Algorithm***

Uses python set object methods - ***Intersection, Union*** and ***Difference***

Any number of terms and combination of Boolean ops

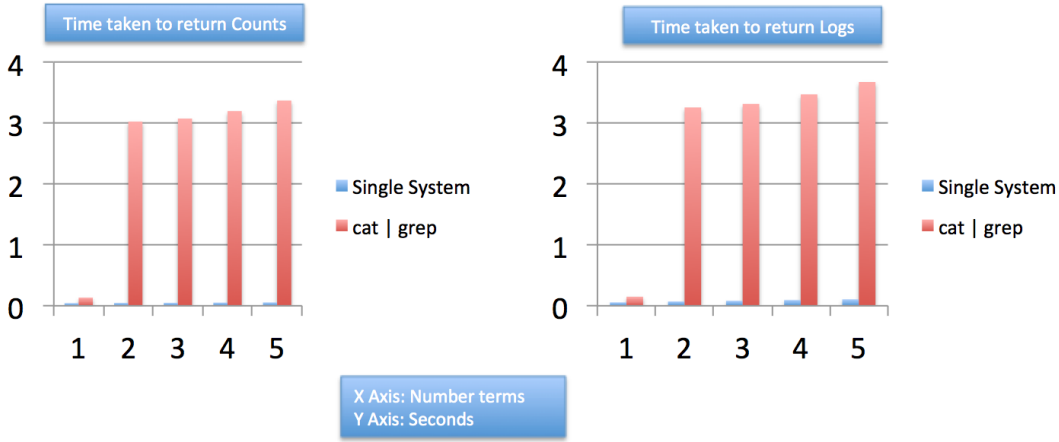
Log searching in Chunk files using File IO object methods:

Seek method to position pointer

Read method to read exact amount of bytes

Single System - Measurements

# of terms	Rows returned	Counts		Logs	
		Single Sys	Grep	Single Sys	Grep
1	~350	0.044	0.134	0.056	0.149
2	~750	0.047	3.029	0.071	3.261
3	~1150	0.049	3.078	0.082	3.319
4	~1550	0.052	3.201	0.094	3.476
5	~1900	0.055	3.374	0.106	3.679
	Average:	0.0494	2.5632	0.0818	2.7768



Input log file:

~83MB
1 Million logs
10 words/log

Tests:

5 sets of tests
Terms with OR operator

```
grep -w -c "chore\|bland\|out"
search.py "((chore|bland)|out)"
```

Results (single system):

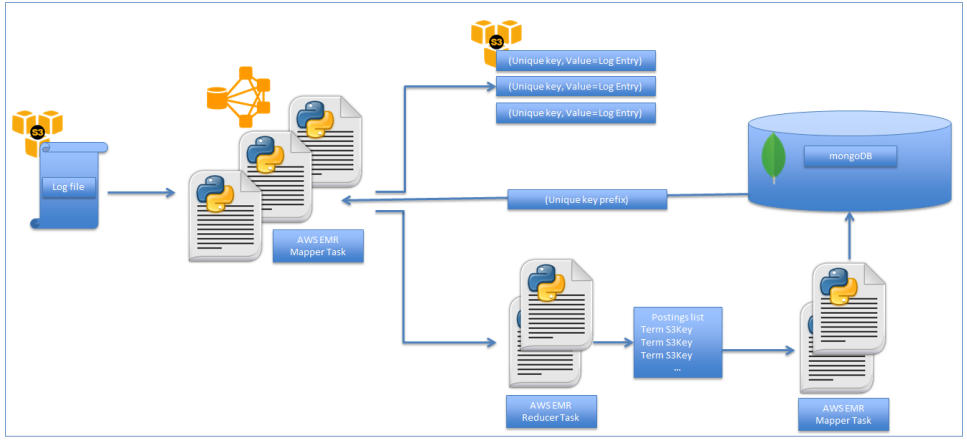
Counts: ~ 52 times faster than grep
Logs: ~ 34 times faster than grep

Additional Measurements:

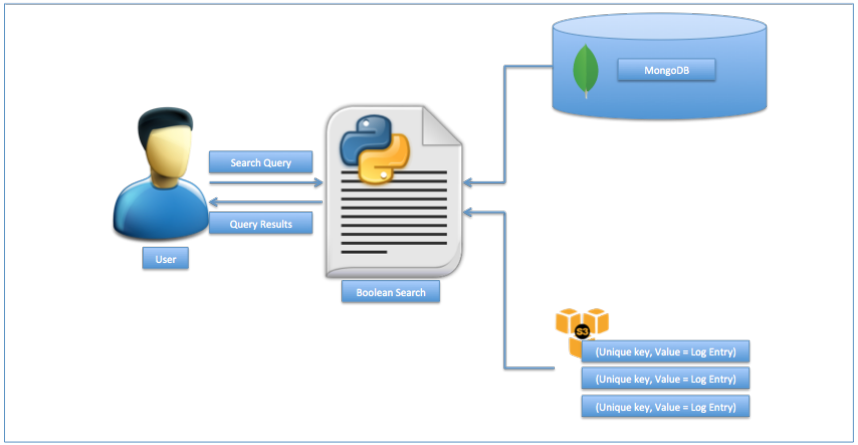
Single System Indexing time:
~ 4 mins

Hybrid System - Implementation

INDEXING



QUERYING



Log Storage:

One log per S3 Key/Value pair

Postings list:

pList = [1,2,3]

Key generation:

Counter in mongoDB (findAndModify)

MongoDB Collection Structure:

{Term: term1, postingsList: [1,2,3]}

Reducer Output:

[{"Term": "One"}, {"$addToSet": {"postingList": {"$each": ["49a5770","
49"]}}}]

Query Language:

Implemented in mongoDB

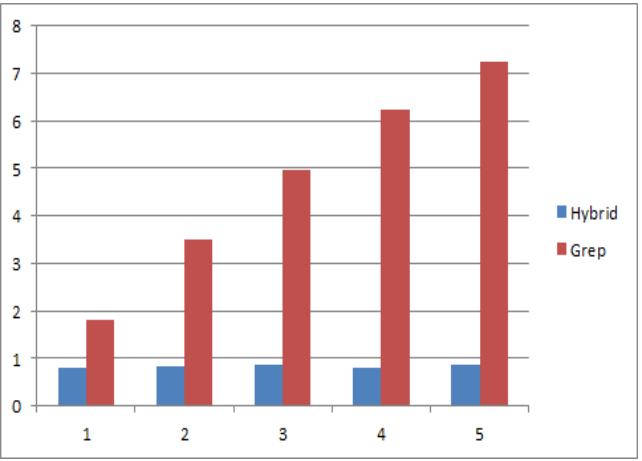
Leverages mongoDB to perform in database posting aggregation using ***Aggregate, Unwind, setIntersection and setDifference***

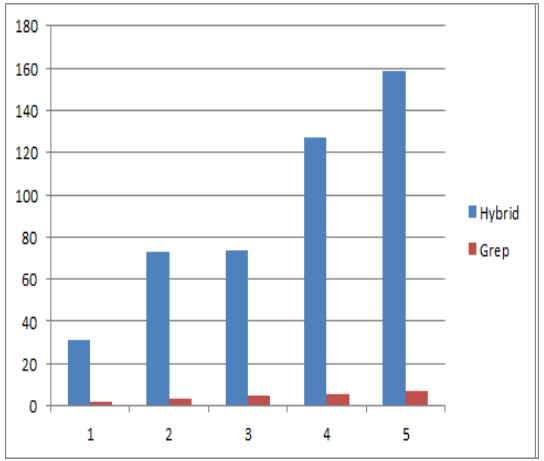
Log Retrieval:

Large number of reads from S3

Hybrid System - Measurements

# of terms	Rows returned	Counts		Logs	
		Hybrid Sys	Grep	Hybrid Sys	Grep
1	~360	.80	1.82	31.23	1.81
2	~800	.82	3.51	72.7	3.27
3	~1150	.88	4.96	73.7	4.82
4	~1550	.81	6.25	127.2	5.91
5	~1900	.88	7.23	158.6	6.97
	Average:	.838	4.75	92.6	4.55





Input log file:

~83MB
1 Million logs
10 words/log

Tests:

5 sets of tests
Terms with OR operator

```
grep -w -c "chore\|bland\|out"
search.py "chore|bland |out"
```

Results (hybrid system):

Counts: 5.6 times faster than grep
Logs: ~20 times slower than grep (Network and S3 latency)

Additional Measurements:

Hybrid System(1,4 large) Indexing time:
1 hr 17 minutes- Indexer job
8 minutes- mongoDB upserts

Hybrid System - Improvements/Future Work

- AWS
  - EBS vs S3
  - Search performance on EC2

- Right sizing Key-Value file

- MongoDB
  - mmap v1 vs wiredtiger
  - GridFS
  - Indexing and Sharding

Challenges and Conclusion

Challenges

Unique Key generation for S3

Every mapper works independently

Keys have to be unique across mappers

Our solution is to use MongoDB as the Unique ID generator

Document size in MongoDB

Inverted Index = term paired with set of pList

MongoDB provides the above structure

Hit 16 MB document size limit

Solution = Term and location pair per document

{Term: term1, postingsList: pList1}
{Term: term1, postingsList: pList2}
...

Conclusion

Inverted Index

A simple concept like Inverted Index can be used to create fast databases

Cloud technologies to scale out

AWS cloud technologies such as EMR and S3 can be leveraged easily to create huge databases

Our hybrid system is slower than grep for an 83 MB log file, but will be faster if the log size is bigger

Multi-word search

The solution can be extended to support multi-term queries such as - `("term1 term2 term3" & term4)`

This can be achieved by adding the location of a term to the postings list