GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Secure Elasticsearch Clusters on HPC Systems for Sensitive Data

Hendrik Nolte, Lars Quentin, Julian Kunkel

16.05.2024

HPC-IODC

Outline

Large Data Sets

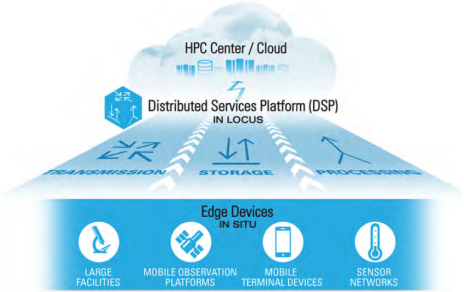


Figure: Source: https://www.hpcwire.com/2018/08/31/the-convergence-of-big-data-and-extreme-scale-hpc/

- New and inexpensive sensors create a wealth of information
- Processing these large data sets requires powerful infrastructure
  - E.g. HPC systems
- Data streams from multiple and potentially diverse sources
  - Requires data selection based on semantic metadata

Data Cataloging Examples

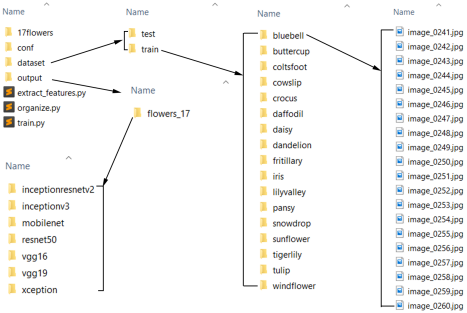


Figure: Source: `https://gogulilango.com/software/flower-recognition-deep-learning`

- One simple way is to encode semantic metadata within file paths/names
- This has several disadvantages:
  - each key-value pair needs two folders
  - Very nested structures
  - No indexing
  - no aggregation with deduplication

Data Cataloging

```
"Metadata": {
"city": "Pittsburgh",
"make": "Apple",
"model": "iPhone 14",
"state": "Pennsylvania",
"zip": "15210",
"processed": "false"
}
```

```
SELECT parent_path,name,size,user_tags
FROM "vast_big_catalog_table:/"
where user_tags_count > 0
AND element_at(user_tags, 'processed')='false';
parent_path | name | size | user_tags
----------------------+----------------------------------------------------
----
/photos/ | andy-coffin.jpg | 414403 | {processed=false}
/vfs/processed/false/ | PXL_20230115_082215604.jpg | 3143258
| {processed=false}
/photos/ | PXL_20230215_222637434.MP.jpg | 5779131 | {processed=false}
```

```
SELECT parent_path,name,size,uid FROM
"vast_big_catalog_table:/"
WHERE uid = 50006
AND size > 52428800
AND extension = 'pdf';
```

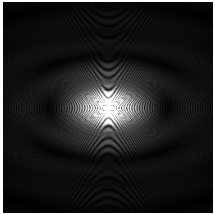
Figure: Source: https://www.vastdata.com/blog/vast-catalog-treat-your-file-system-like-

- This is no new observation/problem
- Filesystem providers have reacted on it by including indexing engines
  - E.g. VAST Catalog
- These can substitute `find` commands
- But also allow to select files/objects based on semantic metadata
- Other approaches by Kalray use an external ES cluster

Motivation of this Work

- There is a clear need for data catalogs, also on HPC systems
- There are different off-the-shelf solutions available
- But there are different disadvantages:
  - Entry costs
  - Lock-in effect
  - Missing baseline
  - Fixed size clusters
- Therefore, we wanted a simple setup with the following properties:
  - On-demand cluster spawner which works in normal Slurm job allocations
  - Clusters should be transparently and arbitrarily sizable
  - Generic throughput-oriented benchmarker to assess performance

Use Case: Integrating MRI Scans



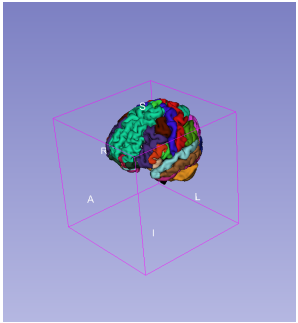
(a) K-Space

(b) Image Space

Figure: BART's Shepp Logan phantom.

- Original scope: Index k-space MR images
- Goal: Select only specific files based on domain-specific metadata
  - Recording protocol (T1, T2)
  - Body part (head, heart, knee,..)
  - Capture provenance
    - Software version (git commit hash)
    - Acceleration factor

Extending the Original Scope



- Extension requested to incorporate sequential processing steps
- Contain quantized markers about brain health:
  - Volumetric thickness analysis
  - Segmentations
  - Derived attributes like brain age
- Integrate with other measurements
  - E.g. EEG data
- Or solely select MRI scans based on properties

Modeling

Excerpt from the Custom MRI Model

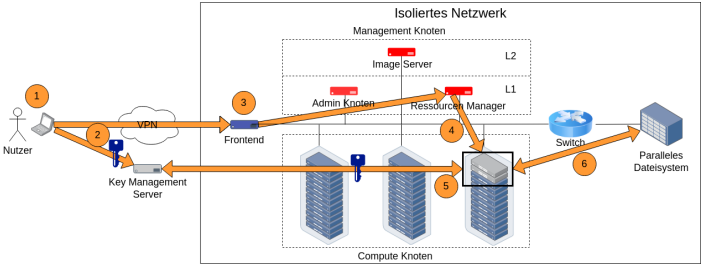
```
"SystemVendor" : "Text('Siemens', 'Philips', 'GE')",
"SystemFieldStrength" : "Float(1.5,3,7)",
"NumberReceiverChannels" : "Integer(8,12,16,32,64)",
"Age": "Integer(1..100)",
"BrainAge" : "Integer(Age + NormDist(5))",
"BodyPart" : "Text('Head','Heart','Knee')",
```

- For simplicity merging both models
- In total 61 keys are defined
- modeling done together with doctors
  - *brain age* variance of 5 years
- Simulating 1 million documents

Encryption these Documents

- Sensitive data should be processed
  - Requires additional security measures, i.e., encryption
- Therefore, encryption support is added as 4th requirement
- Two different encryption techniques are compared:
  - One uses GoCryptFS on an isolated HPC partition
  - The other uses a newly developed attribute-based encryption (ABE)

Setup GoCryptFS



- Slurm job runs on an isolated SecureHPC Partition
  - Access requires 2FA, only one user per session
- The filesystem is mounted with GoCryptFS
  - A FUSE-based stacked cryptographic file system

Attribute-based Encryption

Excerpt from the Custom MRI Model

```
AES256("SystemVendor") : AES256("Text(‘Siemens’, ’Philips’, ’GE’)"),
AES256("SystemFieldStrength") : OPE(Float(1.5,3,7)),
AES256("NumberReceiverChannels") : OPE(Integer(8,12,16,32,64)),
AES256("Age"): OPE(Integer(1..100)),
AES256("BrainAge") : OPE(Integer(Age + NormDist(5))),
AES256("BodyPart") : AES256("Text(‘Head’,’Heart’,’Knee’)"),
```

- Documents are encrypted on the client-side
- AES256 is used for
  - All keys
  - Text and keywords
- Therefore, no fuzzy search is possible

Attribute-based Encryption

Excerpt from the Custom MRI Model

```
AES256("SystemVendor") : AES256("Text('Siemens', 'Philips', 'GE')"),
AES256("SystemFieldStrength") : OPE(Float(1.5,3,7)),
AES256("NumberReceiverChannels") : OPE(Integer(8,12,16,32,64)),
AES256("Age"): OPE(Integer(1..100)),
AES256("BrainAge") : OPE(Integer(Age + NormDist(5))),
AES256("BodyPart") : AES256("Text('Head','Heart','Knee')"),
```

- Documents are encrypted on the client-side
- OPE is used for
  - All numeric values
  - E.g. Dates, numbers

Order Preserving Encryption

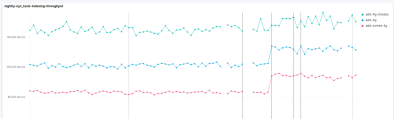
- Implementation used: PyOPE[1]
  - Based on on Boldyreva et al. “Order- preserving symmetric encryption”
- PyOPE maps a range of Integers from [1,M] to [1,N]
- The achieved entropy depends on the ratio of $\frac{M}{N}$
- The following mappings are applied:
  - **Date:** YYYY-MM-DD -> YYYYMMDD yields input range (0,30000000)
  - **Data-Time:** hh:mm:ss are appended using 6 more digits
  - **Float:** Requires expected input range $[l, r]$ and a step size s:

$$m(x) := \left\lfloor \frac{x - l}{s} \right\rfloor$$

Cluster Spawner

- For the first implementation we are using Elasticsearch
- An MPI-job is spawned with a single process per node
- Using a MPI_Gather all hostnames and ranks are sent to MPI root
- The MPI root creates an ES config for each sending process
  - Setting the data path
  - Providing the network interface to use
  - The first ranks are master-eligible
- The storage path is then abstracted using the MPI rank
  - This allows that different nodes can be used from job to job
  - And even switching from Ethernet to OPA
- Software is fully packaged within a Singularity Container
- Data path and configs (etc.) get bind-mounted into the container

Distributed Throughput-Benchmarker I



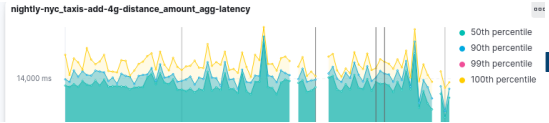


Figure: Source: `https://elasticsearch-benchmarks.elastic.co/#tracks/nyc_taxis/nightly/default/90d`

- The canonical tool for ES benchmarking is Rally[a]
- It can run distributed benchmarks
- It uses a fork-join model to orchestrate and synchronize
- It is sensitive to latencies and scales down when these increase
- Mainly used for regression testing for new ES versions

[a] https://github.com/elastic/rally

Distributed Throughput-Benchmarker II

- Dynamically scaling requests based on response latencies is sophisticated
  - Might not model real workloads very good
  - By allowing higher latencies a higher overall throughput might be possible
- Thus, we implemented a new throughput-oriented benchmark
  - it is relatively insensitive to latency increases due to overloading
  - Its NDJSON format for corpora is compatible with Rally
  - Tracks, i.e., benchmark operations, are simple to port
  - It currently only supports write and read operations
  - It can be arbitrarily scaled using MPI

Distributed Throughput-Benchmarker III

- Ingest:
  - Corpus defined in an NDJSON
  - It is split up into equally large chunks by the MPI root
  - The offset is distributed to each process which use *lseek* on the corpus
- Query:
  - Uses its own JSON-based DSL, but can be translated to Rally tracks
  - Each query runs independently within a fork-join model
  - Warm-up runs to optimze JVM JIT
  - Disabled caches
  - *Session Objects* for persistent TCP connections
- Measurements:
  - The host is randomly chosen by each process at the beginning
  - Only the time from request to response is measured
  - Total runtime configurable and measured in wall-clock time
    - Thus, payload verification and timeouts covered here and do not dilute the result

System

- System:
  - Benchmarks done on Emmy
  - Double socket nodes, Xeon Platinum 9242 CLX-AP, 368 GiB RAM
  - Lustre: 2 ES14KX with 500 12TB SAS HDD, 1 SFA7700X with 16 1TB SAS SSD
- Data Sets
  - Used a standard NYC Taxis dataset
    - Comprised of 165 million docs with all yellow taxi rides in 2015
    - Has an uncompressed size of 74.3 GB
  - The custom MRI dataset
    - 1 million docs
    - Uncompressed size of 1.5 GB

Ingestion: Single Run

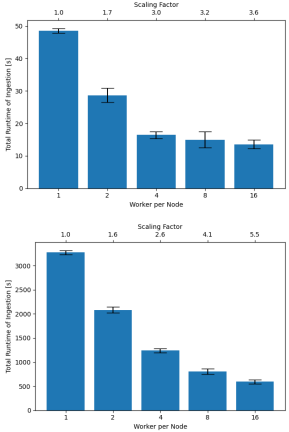
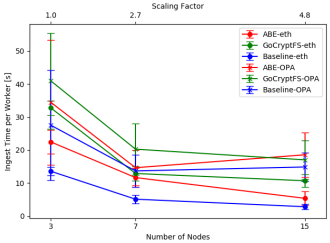
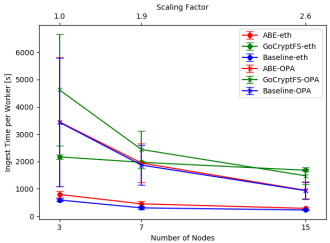


Figure: 3 node cluster via Eth baseline

- Number of workers are scaled from 1,2,4,8,16 ppn
- For the example a reduction is achieved:
  - **MRI:** (49 ± 1)s to (14 ± 1)s
  - approx 70k docs/s
  - **NYC:** (3270 ± 44)s to (590 ± 42)s
  - approx 280k docs/s
- NYC shows a much better scalability beyond 4 ppn

Ingestion: Multiple Cluster Sizes





- From the single run analysis only choosing the best ones
  - Reduces data points from 180 to 36
- Baseline is always fastest
- GoCryptFS is slowest
- MRI shows better scalability with the number of cluster nodes

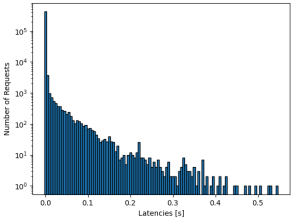
Encryption Cost ABE

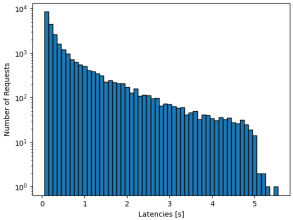
- However, the encryption cost for ABE was excluded
  - Client-side encryption is embarassingly parallel
  - Thus it would have diluted the measurements
- ABE extremly slow due to the use of PyOPE
  - Completely written in Python from Scratch
  - Recursive patterns
- Results:
  - **NYC:** (0.456 ± 0.001) s/doc
  - **MRI:** (0.079 ± 0.002) s/doc
- Different runtimes due to different amount of numbers within docs
  - Shows how unperformant the current OPE implementation is
- Future work!

Match All Query: Introduction

- Simple filter which is always true
  - Thus, no logic has to be executed
- ES dynamically allocates threads, e.g. based on segment count
- On an unoptimized index the match all query is not necessarily the baseline
- Other filters can get treated differently, e.g. range filter
- Can be used for fine-tuning and network optimizations
- In addition: To answer **any** query ES always communicates with all active data nodes holding an relevant shard

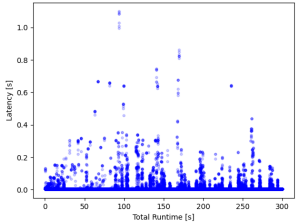
Match All Query: Latency Distribution

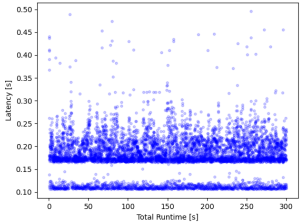




- 7 node cluster with Ethernet Baseline
  - **Top:** 10 docs per response; 1ppn
  - **Lower:** 10k docs per response; 16 ppn
- In both cases super-exponential decay is observed
- One can see how the cluster gives in to increasing load

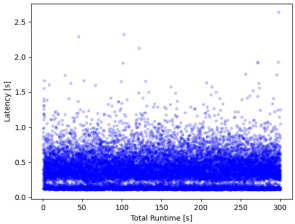
Match All Query: Latency Time Series I

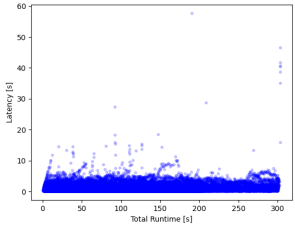




- 7 node cluster with Ethernet Baseline
  - **Top:** 10 docs per response; 1 ppn
  - **Lower:** 10k docs per response; 1 ppn
- **Top:** Beating pattern, parallel requests
- **Lower:** Observe 2 bands
  - 10k docs cause a much higher network load
  - Some workers get randomly localhost assigned
  - These have a lower network overhead

Match All Query: Latency Time Series II



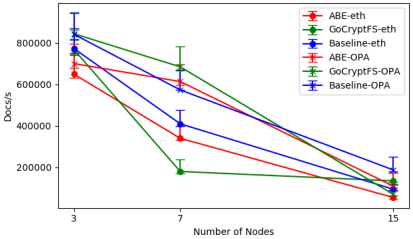


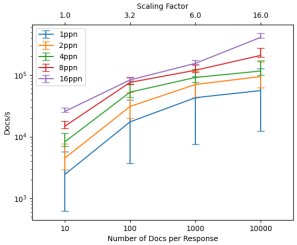
- 7 node cluster with Ethernet Baseline
  - **Top:** 10k docs per response; 4 ppn
  - **Lower:** 10k docs per response; 16 ppn
- Cluster gives continously in
- Dispersion in latencies increases
- The two bands smear out
  - Waiting time in queues and
  - Intra-cluster communication are dominating

Match All Query: Distribution Modeling

- Different Distributions can be observed:
  - Aggregation for latency yields a super-exponential decay
  - Time resolution shows
    - Beating patterns
    - Bands
    - Increasing smearing
- Makes it hard to model the distribution accurately
  - Use the geometric mean to account for the multiplicative nature
  - Use either gstd or std depending on plots or tables

Match All Query: Scalability





- With increasing cluster size performance decreases
  - Increasing overhead due to intra-cluster communication
  - Diminishing return for threading and parallelization
- Increasing docs per response maximizes throughput
- More workers increase overall throughput
  - Although latency increases
  - Justifies new throughput oriented benchmark

Distributed Throughput-Benchmarker II

- Match all has shown risks when scaling out
  - Parallelization overhead which could not be weighted out by compute intensity
- Thus: More compute-itensive query is required
  - Histogramm aggregation of all tripes
    - Larger then 0 miles and smaller then 50
    - Aggregated with respect to trip distance and cost
- Observe almost linear scalablity
  - GoCryptFS Speedup: Measurement uncertainty or additional caching by FUSE

Cluster-Size	Baseline [s]						GoCryptFS [s]					
	Ethernet			OPA			Ethernet			OPA		
	gmean	1st	99th	gmean	1st	99th	gmean	1st	99th	gmean	1st	99th
3	7.4	6.8	65.5	8.5	7.9	57.7	7.1	6.9	19.2	7.1	6.9	16.3
7	2.9	2.9	3.7	3.5	3.4	3.6	2.9	2.9	3.1	3.0	2.9	3.6
15	1.6	1.6	1.7	1.6	1.6	1.8	1.4	1.4	1.5	1.6	1.6	1.8
Speedup	4.6			4.6			5.1			4.4		

Custom MRI Benchmark: Intro

- Until now I have (hopefully) motivated:
  - Proper query modeling is important since they scale differently
  - Modeling the latency distributions is very difficult
    - Will just use the geometric mean and the std for its additive nature

Custom MRI Benchmark: Queries

- **Body Part:** *term* query. Corresponds to "return all MRI scans of the head".
- **Systemvendor:** *match_phrase* query. Corresponds to "return all MRI scans from a specific system vendor". (e.g. Siemens, GE,..)
- **Age:** *range* query. Corresponds to "return all MRI scans of patients with an age between 50 and 60 years".
- **Body Part and Resolution:** Boolean-chained *term* queries. Corresponds to "return all MRI scans of the head with a resolution, of 256 times 256".
- **Brain-Age:** script filter. Brain age larger than 5 years of patient age.
- **Vendor and Body Part:** *term* and *match* queries.Corresponds to "return all MRI scans of heads done on either a Siemens scanner with an MP2RAGE or a General Electric scanner with a BRAVO sequence".
- **Age-Weight Distribution:** A chained *match*, *range* query and histogram aggregation. Corresponds to "select all men with an age lower than 70 and do a histogram of the patient weight with a binning size of 5".

Custom MRI Benchmark: Throughput

Setup	Cluster-Size	Body Part	System-vendor	Age	Body Part Resolution	Brain-Age	Vendor+ Body P.	Age-W. Distr.
Baseline	3 Node	190k±15k	180k±13k	190k±15k	124k±6k	94k±5k	96k±5k	4k±160
	7 Node	212k±10k	244k±4k	103k±5k	114k±6k	101k±4k	166k±4k	12k±700
	15 Node	39k±9k	37k±4k	49k±4k	48k±34k	37k±4k	72k±5k	5k±2k
GoCryptFS	3 Node	180k±13k	157k±10k	176k±13k	119k±4k	170k±13k	153k±5k	5k±100
	7 Node	71k±9k	100k±12k	51k±4k	89k±7k	50k±5k	103k±4k	6k±480
	15 Node	29k±6k	33k±3k	49k±3k	41k±19k	39k±3k	60k±4k	4k±670
ABE	3 Node	120k±15k	110k±11k	160k±6k	90k±80k	-	140k±7k	-
	7 Node	80k±20k	80k±7k	80k±7k	110k±20k	-	95k±7k	-
	15 Node	6k±8k	10k±8k	12k±8k	11k±9k	-	12k±8k	-

- For the MRI corpus and queries no scalability advantages could be identified
- Different queries exhibit a different scalability
- Queries requiring a fixed length/intervall not possible with OPE

Custom MRI Benchmark: Latencies

Setup	Cluster-Size	Body Part	System vendor	Age	Body Part Resolution	Brain-Age	Vendor and Body Part	Age-Weight Distribution
Baseline	3	0.6±0.1	0.7±0.2	0.6±0.1	0.02±0.02	0.6±0.1	0.7±0.1	0.020±0.005
	7	0.7±0.1	0.7±0.1	0.7±0.1	0.02±0.02	0.7±0.1	0.5±0.1	0.013±0.003
	15	0.9±0.3	0.9±0.2	0.7±0.2	0.03±0.05	0.9±0.2	0.5±0.1	0.009±0.004
GoCryptFS	3	0.6±0.1	0.6±0.1	0.6±0.1	0.02±0.01	0.6±0.1	0.5±0.1	0.022±0.004
	7	0.7±0.1	0.7±0.1	0.7±0.1	0.02±0.02	0.7±0.1	0.5±0.1	0.013±0.003
	15	1.0±0.3	0.9±0.3	0.8±0.2	0.03±0.05	0.9±0.3	0.5±0.1	0.010±0.005
ABE	3	1±0.2	1.2±0.3	1.2±0.3	0.04±0.001	-	1.1±0.2	-
	7	1.4±0.6	1.4±0.6	1.4±0.6	0.04±0.001	-	1.2±0.2	-
	15	2.5±3.7	2.6±3.6	1.8±2.5	0.05±0.09	-	1.2±1.3	-

- Varying impact of cluster size on latency
- Only Aggregations benefits
- Differences in overall latencies due to different response sizes

Conclusion

- Presented a workflow that spawns on-demand ES clusters
  - Integrates with Slurm; dynamically scalable
- Presented a throughput oriented benchmark
  - Corpora and Queries completetly customizable
  - Good interoperability with Rally
- Demonstrated importance of accurate data and query modeling
  - Neither more worker nor more ES nodes must help throughput
- Provided an example use case demonstrating a systematic analyzis
  - Can be ported to other other uses cases in a similar manner