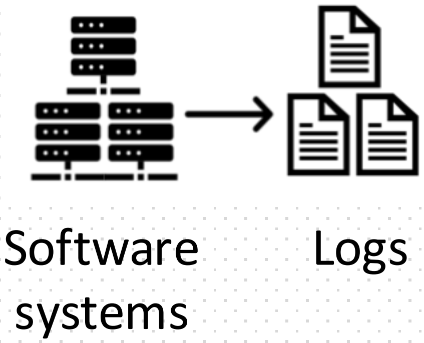
μSlope: High Compression and Fast Search on Semi-Structured Logs

Rui Wang, Devin Gibson, Kirk Rodrigues, Yu Luo,
Yun Zhang, Kaibo Wang, Yupeng Fu, Ting Chen, Ding Yuan

Presenter: **Yuming Xu**, Hengyu Liang

2024.9.24

Explosive growth of log data



- Error detection
- Performance debugging
- Operation management
- Usage analysis
- Security

2022-04-25T00:00:01.000 INFO Task task_12 assigned to container, operation took 0.335 seconds.

Unstructured log


```
{
  "timestamp": "2022-04-25T00:00:01.000",
  "level": "INFO",
  "task": "task_12",
  "message": "Task assigned to container",
  “operationDuration": 0.335
}
```


Semi-structured log

Log Compression in cloud systems

USTC, CHINA ADSLAB

- Logs are widely used in cloud systems
  - Lots of logs are produced every day
  - need to save for months
  - Compression is desirable to save storage cost
    - **1 PB** logs result in annual storage cost at **$35,019,817**

1PB/day at AliCloud
(LogReducer[FAST’21])

10PB/day at Uber
(uSlope[OSDI’24])

Existing work for log compression

USTC, CHINA ADSLAB

- What works have been published in the past 3 years?
  - FAST'21 — LogReducer (Variable compression) —— THU & AliCloud

2022-04-25T00:00:01.000 INFO Task task_12 assigned to container, operation took 21 seconds, error

Origin Log

\Time INFO Task **\Val** assigned to container, operation took **\Num** seconds, **\Val**

Template

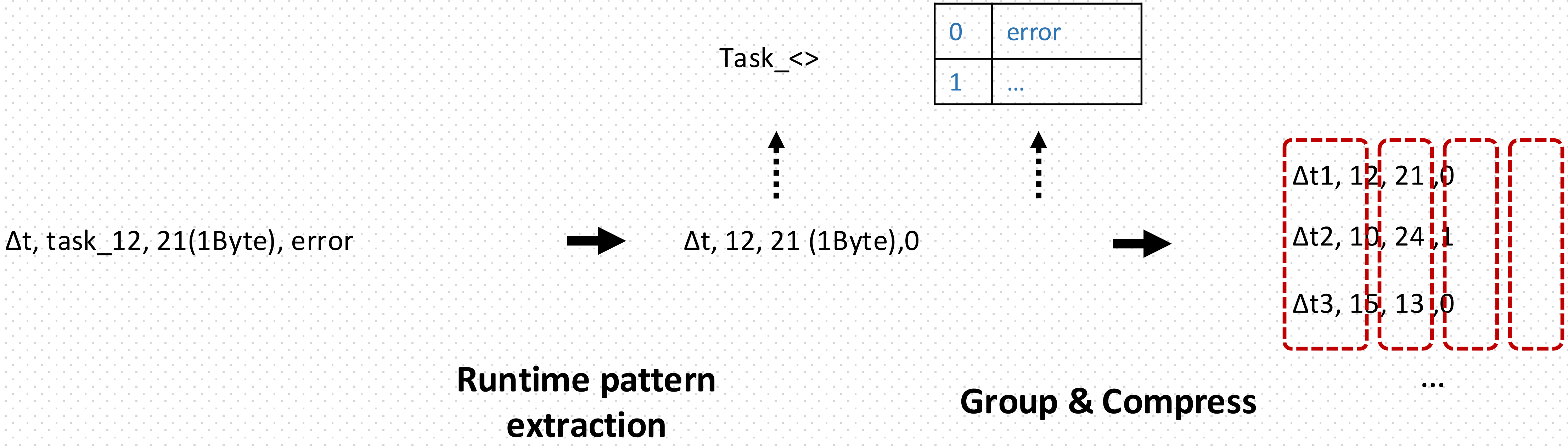
2022-04-25T00:00:01.000, task_12, 21(int), error

Variables

Δt, task_12, **21 (1Byte),** error

Existing work for log compression

- What works have been published in the past 3 years?
  - FAST’21 — LogReducer (Variable compression) —— THU & AliCloud
  - EuroSys’23 — LogGrep (Pattern extraction & Compression) —— THU & AliCloud



Existing work for log compression

- What works have been published in the past 3 years?
  - FAST'21 — LogReducer (Variable compression) —— THU & AliCloud
  - EuroSys'23 — LogGrep (Pattern extraction & Compression) —— THU & AliCloud

Past works mostly focus on unstructured log

2022-04-25T00:00:01.000 INFO Task task_12 assigned to container, operation took 21 seconds, error

→

Origin Log

0	INFO Task **\Val** assigned to container, operation took **\DVal** seconds, **\Val**

Log Type Dictionary

0	task_12
1	error

Variable Dictionary

Timestamp	Log Type	Variable values
2022-04-25T00:00:01.000	0	0 21 1

Encoded Messages

Semi-structured Logs

USTC, CHINA ADSLAB

2023-03-16T07:58:02.368 ERROR Can’t fetch flow 6, cell_32. TraceID abc-xyz, Error Error404, Request namespace_driver_onboarding.

KV pair


```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
     "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
     "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```


2023-03-16T07:58:06.246 ERROR Can't fetch flow 8, cell_32. TraceID def-uvw, Request vehicle_compliance.


```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
     "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```

Semi-structured Logs

2023-03-16T07:58:02.368 ERROR Can’t fetch flow 6, cell_32. TraceID abc-xyz, Error Error404, Request namespace_driver_onboarding.

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

Key can be dynamic

2023-03-16T07:58:06.246 ERROR Can't fetch flow 8, cell_32. TraceID def-uvw, Request vehicle_compliance.

```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```

Semi-structured Logs

USTC, CHINA ADSLAB

2023-03-16T07:58:02.368 ERROR Can’t fetch flow 6, cell_32. TraceID abc-xyz, Error Error404, Request namespace_driver_onboarding.

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

Value can be dynamic

2023-03-16T07:58:06.246 ERROR Can't fetch flow 8, cell_32. TraceID def-uvw, Request vehicle_compliance.

```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```

Semi-structured Logs

USTC, CHINA ADSLAB

2023-03-16T07:58:02.368 ERROR Can’t fetch flow 6, cell_32. TraceID abc-xyz, Error Error404, Request namespace_driver_onboarding.

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

value can be {"key": value} or {value, value,…}

2023-03-16T07:58:06.246 ERROR Can't fetch flow 8, cell_32. TraceID def-uvw, Request vehicle_compliance.

```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```

Traditional RDBMS cannot handle

- Predefined schema for each field

level	message	serviceA. traceId	request	timestamp	serviceB. traceId	serviceB. error
"warn"	"Could not fetch cell for flow 1."	"abc"	"vehicle_compliance"	"2022-04-14T07:58:02.3682"	NULL	NULL
"error"	"Error handling inbound request"	NULL	NULL	"2022-04-14T07:58:02.3722"	"xyz"	"application_error"

Traditional RDBMS cannot handle

USTC, CHINA ADSLAB

- Predefined schema for each field
- Sparse table

level	message	serviceA.traceId	request	timestamp	serviceB.traceId	serviceB.error
"warn"	"Could not fetch cell for flow 1."	"abc"	"vehicle_compliance"	"2022-04-14T07:58:02.3682"	**NULL**	**NULL**
"error"	"Error handling inbound request"	NULL	NULL	"2022-04-14T07:58:02.3722"	"xyz"	"application_error"

Traditional RDBMS cannot handle

USTC, CHINA ADSLAB

- Predefined schema for each field
- Sparse table
- Polymorphism limitation

```
{
  "request": {
    "namespace": "driver_onboarding"
  },
}
```

level	message	serviceA. traceId	**request**	timestamp	serviceB. traceId	serviceB. error
"warn"	"Could not fetch cell for flow 1."	"abc"	**"vehicle_compliance"**	"2022-04-14T07:58:02.3682"	NULL	NULL
"error"	"Error handling inbound request"	NULL	**NULL**	"2022-04-14T07:58:02.3722"	"xyz"	"application_error"

Traditional RDBMS cannot handle

USTC, CHINA ADSLAB

- Predefined schema for each field
- Sparse table
- Polymorphism limitation
- Uber (ClickHouse): only use selected keys as columns, use lots of index to improve query, results in **low compression ratio (less than 4:1)**

level	message	serviceA. traceId	request	timestamp	serviceB. traceId	serviceB. error
"warn"	"Could not fetch cell for flow 1."	"abc"	"vehicle_compliance"	"2022-04-14T07:58:02.3682"	NULL	NULL
"error"	"Error handling inbound request"	NULL	NULL	"2022-04-14T07:58:02.3722"	"xyz"	"application_error"

Limitation of Native JSON support

- E.g. BSON from MongoDB, jsonb from PostgreSQL, and OSON from Oracle

```
{
  "hello": "world"
}
```

JSON Log

```
\x16\x00\x00\x00                    // size (32-bit): 22 bytes
\x02                                // 0x02 = value type String
hello\x00                           // key name
\x06\x00\x00\x00world\x00           // size of value (6 bytes), value
\x00                                // 0x00 = 'end of object'
```

BSON

Limitation of Native JSON support

USTC, CHINA ADSLAB

- E.g. BSON from MongoDB, jsonb from PostgreSQL, and OSON from Oracle
- Low compression ratio
  - Extra metadata
  - Row-oriented format

```
{
  "hello": "world"
}
```

```
\x16\x00\x00\x00                    // size (32-bit): 22 bytes
\x02                                // 0x02 = value type String
hello\x00                           // key name
\x06\x00\x00\x00world\x00           // size of value (6 bytes), value
\x00                                // 0x00 = 'end of object'
```

JSON Log **BSON**

Inefficient search engines

- Low compression ratio
  - Require extra indices
  - The indices size is at the same order of magnitude as the raw data
- Low ingestion speed & High resource usage
  - Parsing, tokenization, updating indices
  - Ingestion involves complex processing

splunk>

elastic

Challenge of using CLP

Search taking lot of time using clg binary #154

Closed bb-rajakarthik opened this issue on Aug 29, 2023 · 4 comments

bb-rajakarthik commented on Aug 29, 2023

Bug

We are using CLP for compressing logs generated by our Kubernetes cluster which are in JSON format. A sample log is given below:

{
"log_time": "2023-08-29T13:55:09.477456Z",
"stream": "stdout",
"time": "2023-08-29T13:55:09.477456564Z",
"@timestamp": "2023-08-29T19:25:09.477+05:30",
"@Version": "1",
"message": " Method: POST;Root=1-64edf8bd-5c762a676349ee71616bb687 , Request Body : {"orders":
[{"order_type":"normal","external_reference_id":"69426","items":
[{"offset_in_minutes":"721","quantity":"1","external_product_id":"225090"}]}]}",
"level": "INFO",
"level_value": 20000,
"request_id": "6f3f3651-a22b-42a0-b5fe-412d2167c5ca",
"kubernetes_docker_id": "caa9102a169a1495e5790cb2c17cb21d0a279ffc50d802d413938870ba59c7c0",
}

Challenge of using CLP

USTC, CHINA ADSLAB

ID	Log Type
0	{"level": "warn","message": "Could not fetch cell for flow \DICTVAR","serviceA": {"traceID": "\DICTVAR"},"request": "\DICTVAR","timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}
1	{"level": "error","message": "Error handling inbound request.","serviceB": {"traceID": "xyz","error": "application_error"},"timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}

Log Type Dictionary

ID	Format
0	1.
1	abc
2	1_vehicle_compliance
3	2022-04-14T07
4	02.368Z
5	02.372Z

Variable Dictionary

Timestamp	Log Type	Variable values
NULL	0	0 1 2 3 58 4
NULL	1	3 58 5

Encoded Messages

request: 1_vehicle_compliance

Challenge of using CLP

ID	Log Type
0	{"level": "warn","message": "Could not fetch cell for flow \DICTVAR","serviceA": {"traceID": "\DICTVAR"},"request": "\DICTVAR","timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}
1	{"level": "error","message": "Error handling inbound request.","serviceB": {"traceID": "xyz","error": "application_error"},"timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}

Log Type Dictionary

ID	Format
0	1.
1	abc
2	1_vehicle_compliance
3	2022-04-14T07
4	02.368Z
5	02.372Z

Variable Dictionary

Timestamp	Log Type	Variable values
NULL	0	0 1 2 3 58 4
NULL	1	3 58 5

Encoded Messages

request: 1_vehicle_compliance

Challenge of using CLP

USTC, CHINA ADSLAB

ID	Log Type
0	{"level": "warn","message": "Could not fetch cell for flow \DICTVAR","serviceA": {"traceID": "\DICTVAR"},"request": "\DICTVAR","timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}
1	{"level": "error","message": "Error handling inbound request.","serviceB": {"traceID": "xyz","error": "application_error"},"timestamp": "\DICTVAR:\NDVAR:\DICTVAR"}

Log Type Dictionary

ID	Format
0	1.
1	abc
2	1_vehicle_compliance
3	2022-04-14T07
4	02.368Z
5	02.372Z

Variable Dictionary

Timestamp	Log Type	Variable values
NULL	0	0 1 2 3 58 4
NULL	1	3 58 5

Encoded Messages

.traceId: abc AND request: 1

{"traceID"*:*"abc"*}*"request"*:*"1*"*
"request":*"1*"*{*"traceID"*:*"abc"*}*

- ❌ Incompatible query interface
- ❌ Poor query performance

NOT, OR operators and numeric comparison ?

μSlope Goals

- Automatic Schema inference
- More complex query support
- High compression ratio and fast search

Characterizing Semi-structured Log

- **Key idea**: Semi-structured logs are highly repetitive
- Dataset analyzed:
  - 21 machine-generated log datasets with 1 million log records each from
    - Frequently used data in Uber
    - Public software: Spark, MongoDB, CockroachDB, ElasticSearch, PostgreSQL
  - 23091 real-world queries spanning twenty days from Uber
    - 7665 of them are unique

Schema Variation

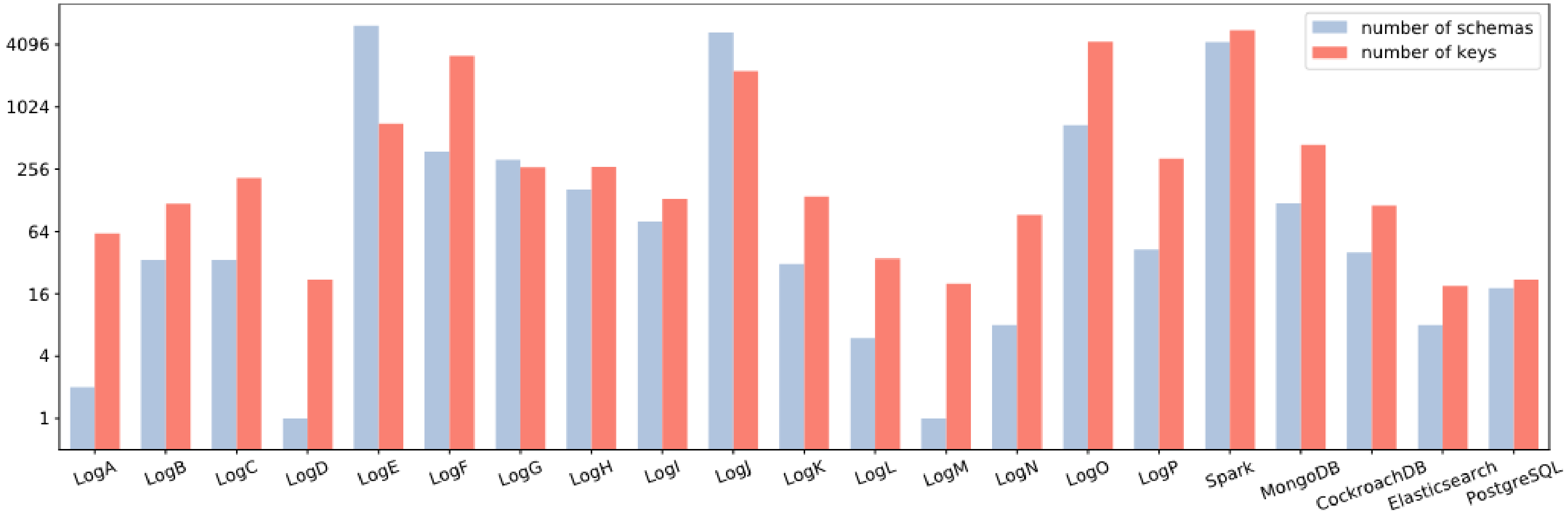
- Schema & Key definition
  - Two schemas are the same if and only if their keys are all the same
  - Two keys are the same if and only if their full name & value types are the same
    - A key’s full name includes all the nested keys


```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```

Schema Variation

- Schemas are dynamic, but also **repetitive**
  - All except two datasets have more than 1 unique schemas
  - On average, 25000 records get the same schema
    - Repetition will be larger when increasing the sample size
  - Unique keys varies greatly (from 20 (LogM) to 5627 (Spark))
    - This result suggests that the variation in schemas is likely due to the variation of keys

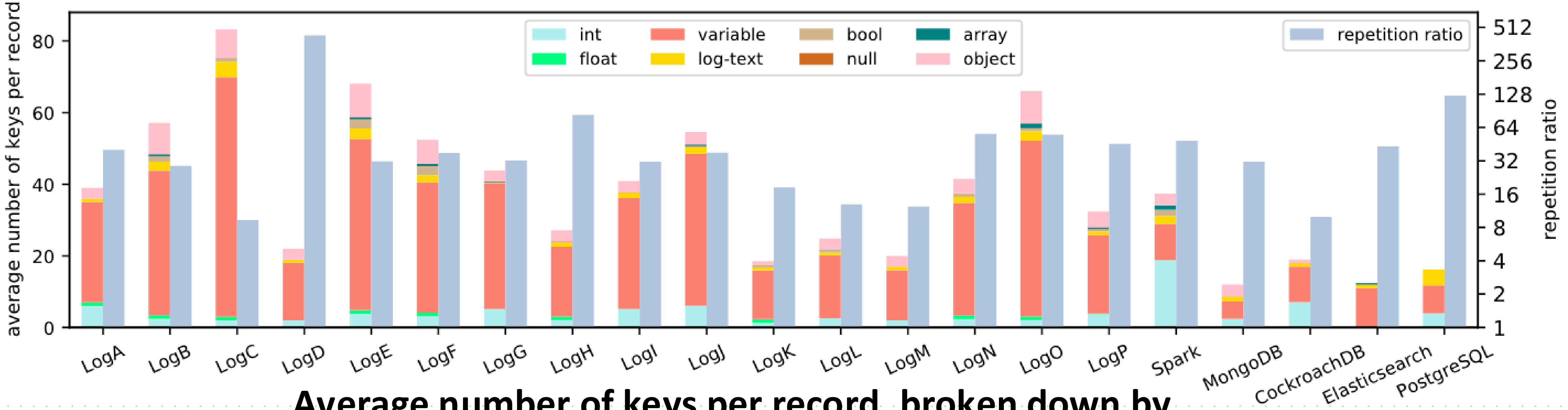


Number of unique schemas and keys for each dataset

Type Composition & Repetitive Values

USTC, CHINA ADSLAB

- **71%** of the values are variables (single-word strings) and highly **repetitive**
  - High repetition ratio means dictionary deduplication can be effective
    - What’s more, dictionary can speed up common wildcard keys (nearly 30%) filter queries.
  - Array fields are low at 0.79%, and only 0.4% of the queries search on array fields



Average number of keys per record, broken down by value types & repetition ratio of variables

Importance of Schema Search

USTC, CHINA ADSLAB

- **29%** of the queries can be completed by querying the schema structure
  - They do not match any of the schema structure
    - Example: error detect
      - Regularly verify the nonexistence of certain error events
  - But existing systems waste this opportunity since the schema structure is interspersed with the values

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

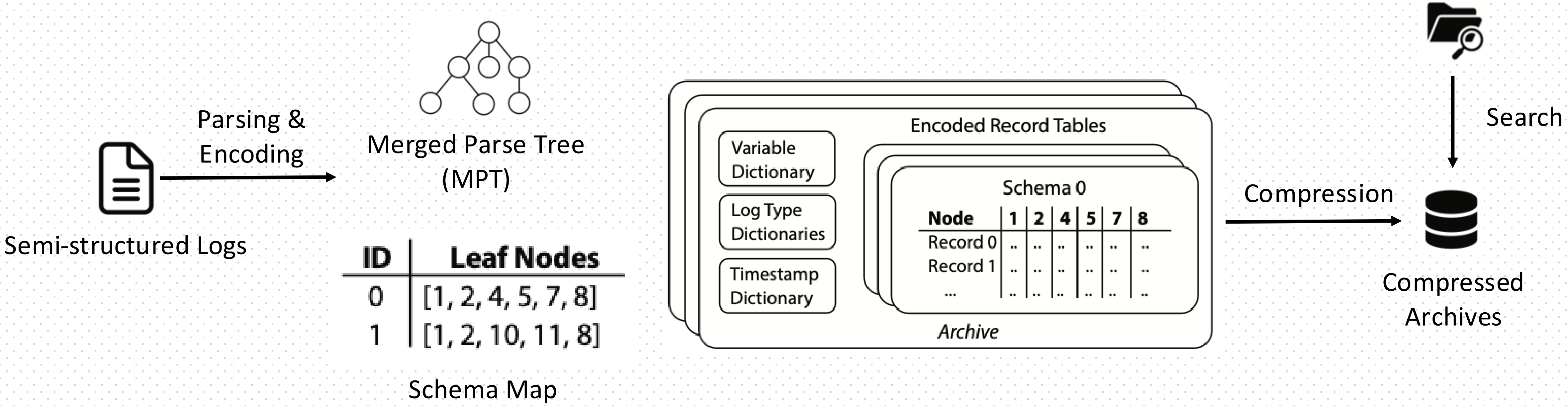
Error:*

Key Takeaways

- **Schemas are dynamic, but also repetitive**
  - Need to precisely track the schema of semi-structured data
  - Repetition shows opportunities to group records in well-structured form.
- **71% of the values are variables (single-word strings) and highly repetitive, and commonly queried on**
  - Dictionary can help effectively deduplication and speed up search
- **29% of the queries can be completed by querying the schema structure**
  - Decouple schema structure and records can speed up search

μSlope Design

- Overview
  - Logs are parsed, partitioned by schema, encoded, then compressed into archives
  - Generate multiple archives
    - Once the memory buffer is full, write all the data into archives
    - Each archive can be searched independently, resulting in high parallel performance

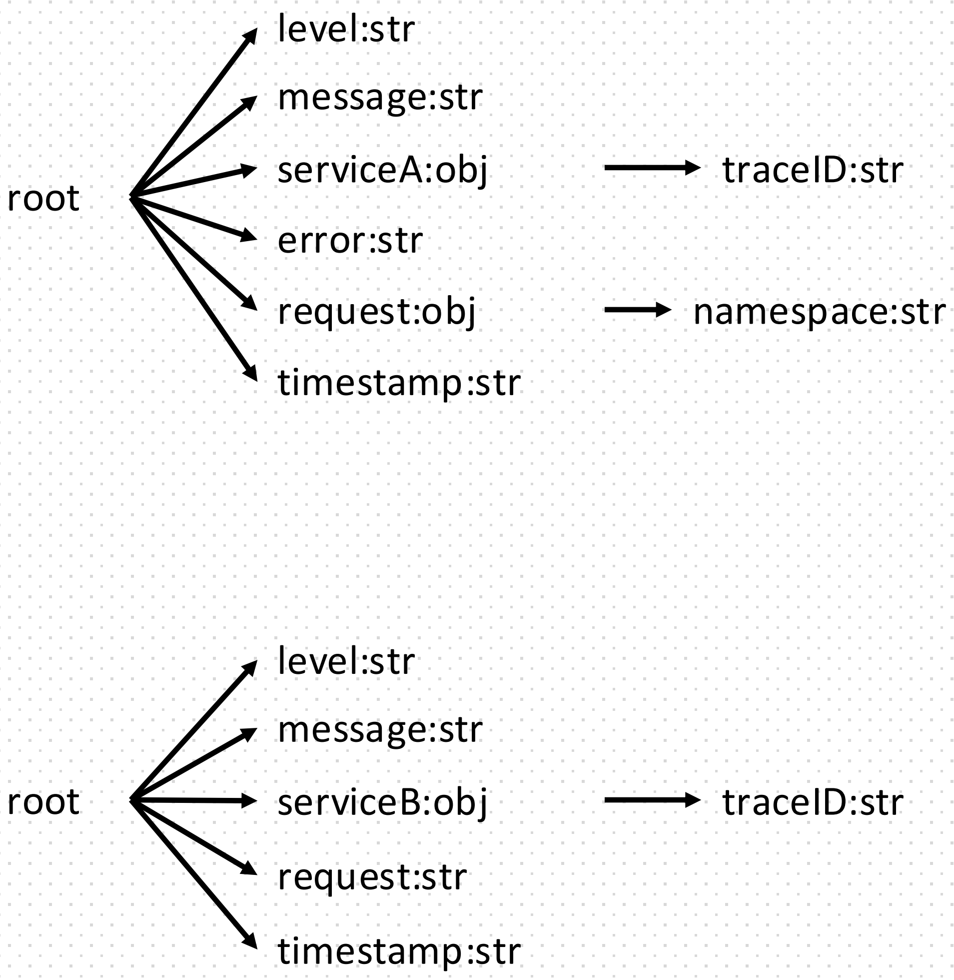


Schema Tree

```
{
   "level": "error",
   "message": "Can’t fetch flow 6, cell_32",
   "serviceA": {
      "traceID": "abc-xyz"
   },
   "error": "Error404",
   "request": {
      "namespace": "driver_onboarding"
   },
   "timestamp": "2023-03-16T07:58:02.368"
}

{
   "level": "error",
   "message": "Can't fetch flow 8, cell_32",
   "serviceB": {
      "traceID": "def-uvw"
   },
   "request": "vehicle_compliance",
   "timestamp": "2023-03-16T07:58:06.246”
}
```

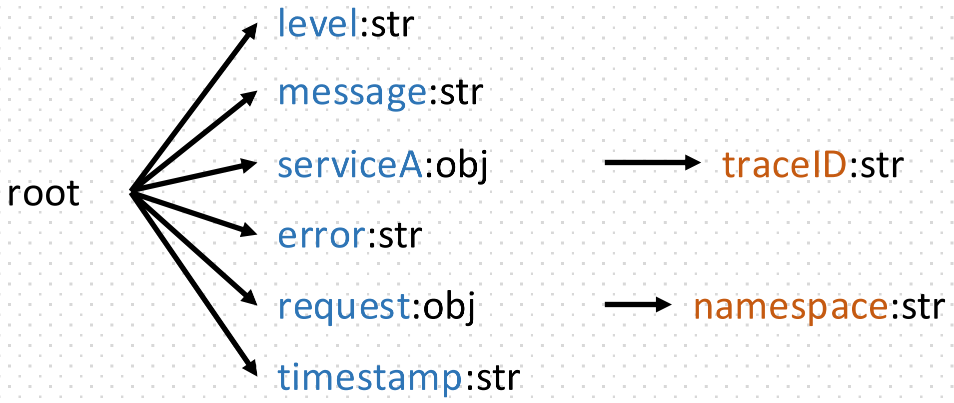
Log



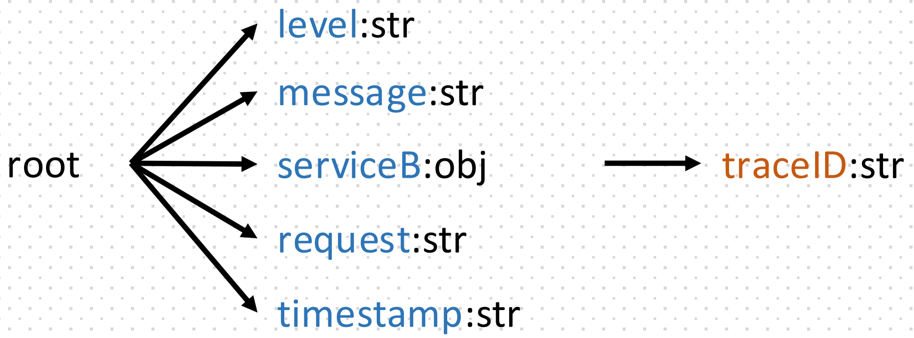
Schema Tree

Schema Tree

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

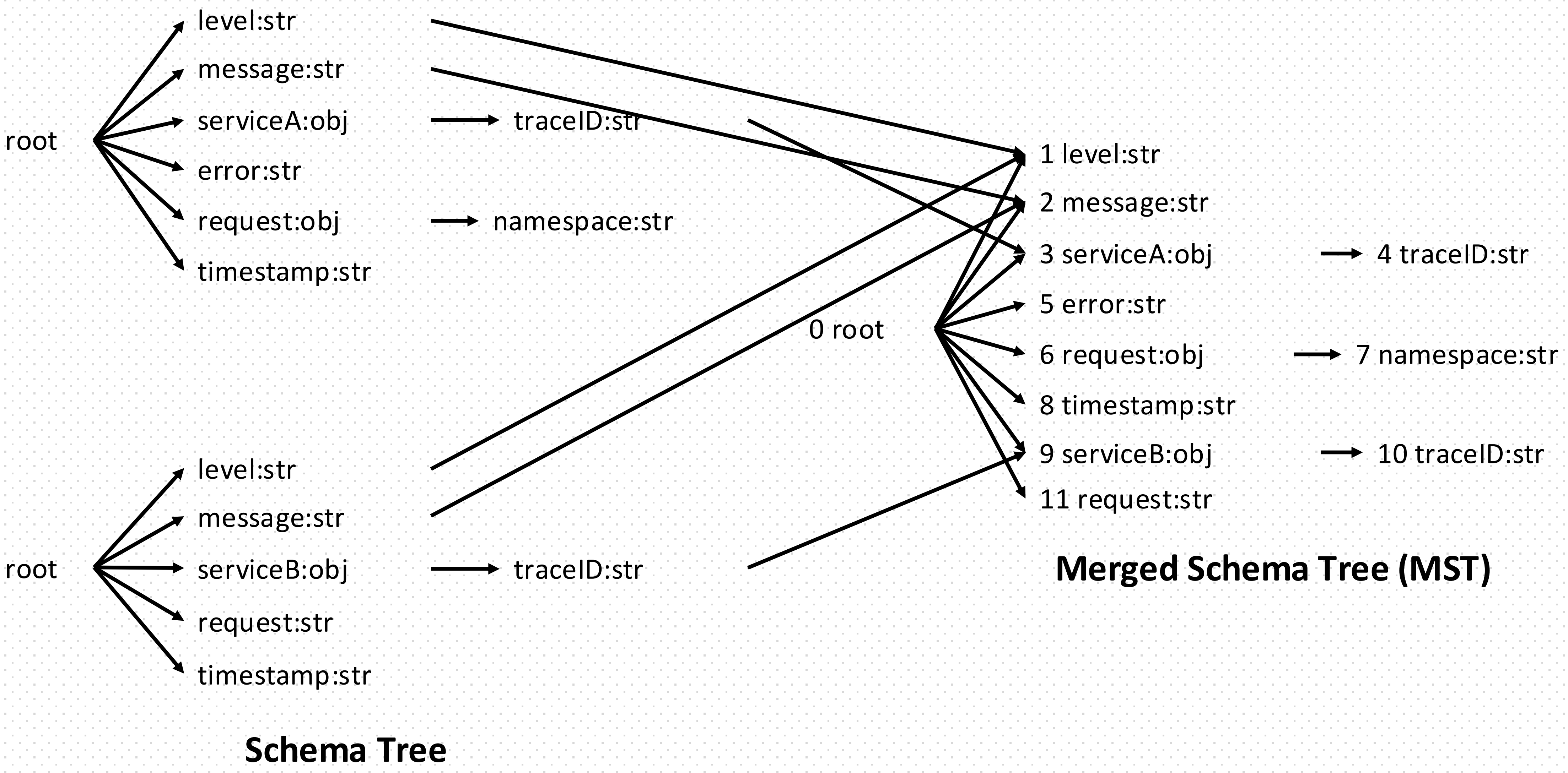



```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```



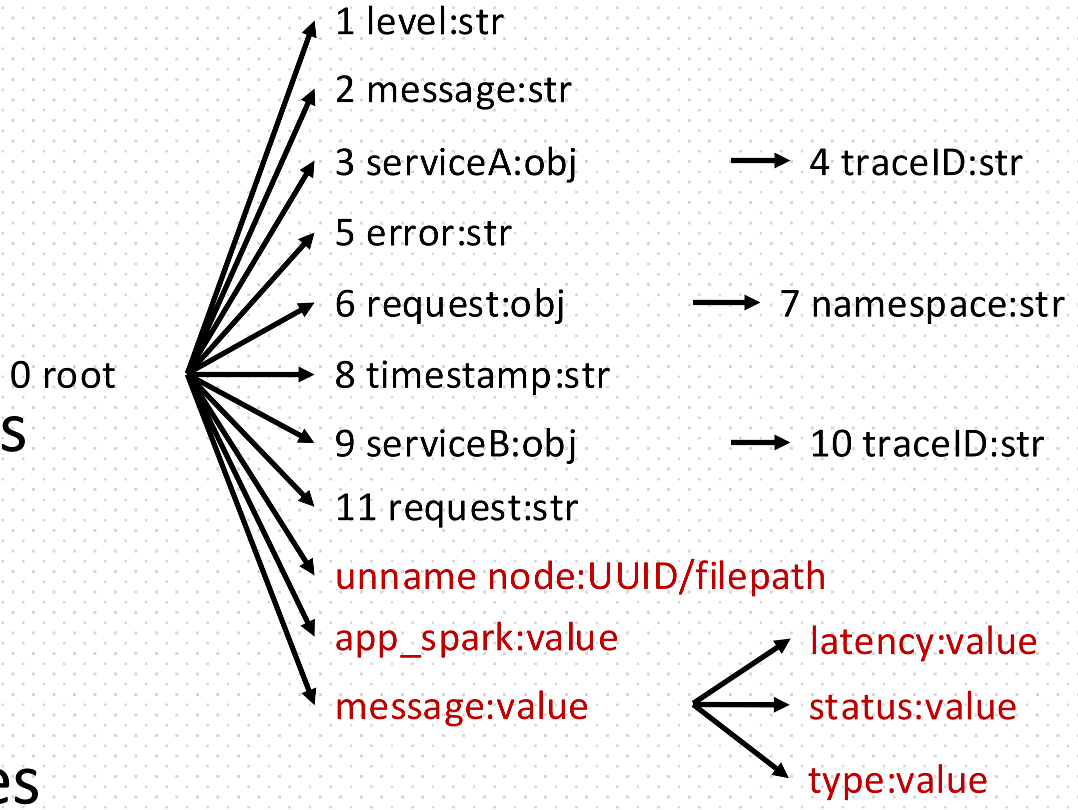
Log **Schema Tree**

Merged Schema Tree



Merged Parse Tree

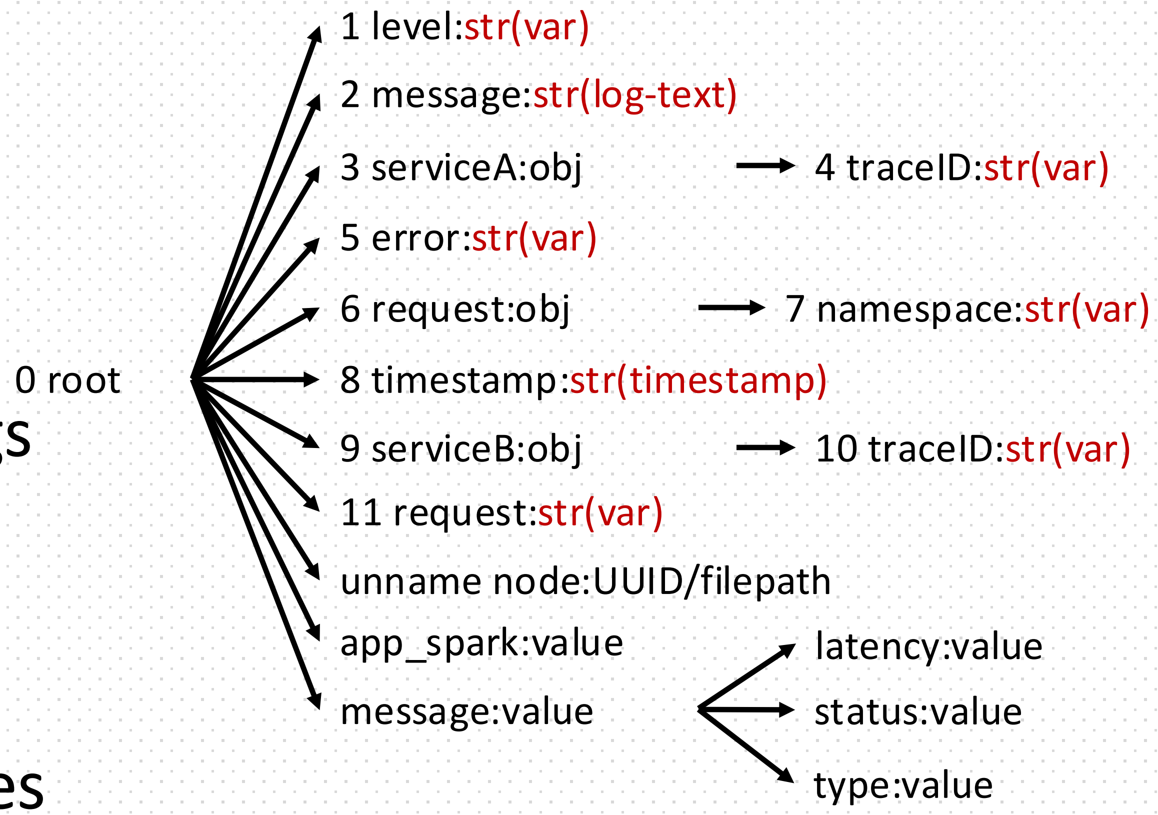
1. Key name contains random data
   - UUID, filepath, …
2. The value of a key could be highly repetitive
   - "app": spark
3. Encode the structure of strings with key-value pairs
   - "message": "latency=35, status=OK, type=READ"
4. Stores fine-grained string types



Merged Parse Tree (MPT)

Merged Parse Tree

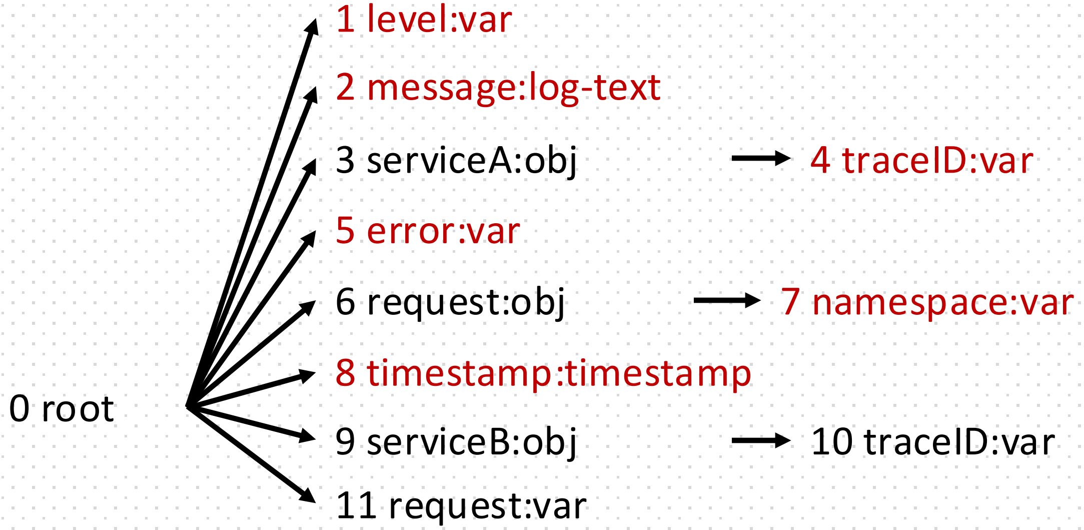
1. Key name contains random data
   - UUID, filepath, …
2. The value of a key could be highly repetitive
   - "app": spark
3. Encode the structure of strings with key-value pairs
   - "message": "latency=35, status=OK, type=READ"
4. Stores fine-grained string types



Merged Parse Tree (MPT)

Merged Parse Tree

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```



Merged Parse Tree (MPT)

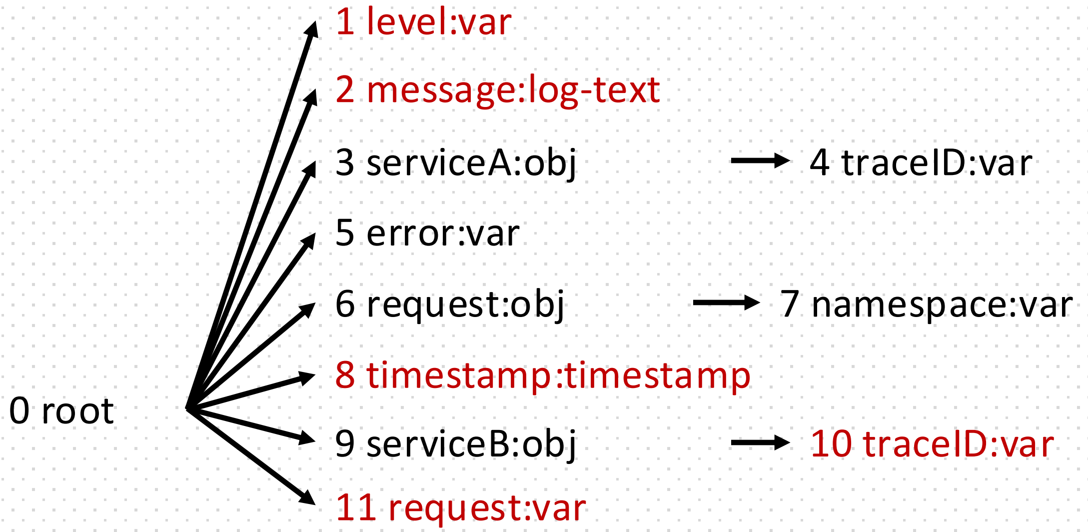
Schema ID	Node IDs
0	1 2 4 5 7 8

Schema Map

Merged Parse Tree

```
{
  "level": "error",
  "message": "Can’t fetch flow 6, cell_32",
  "serviceA": {
    "traceID": "abc-xyz"
  },
  "error": "Error404",
  "request": {
    "namespace": "driver_onboarding"
  },
  "timestamp": "2023-03-16T07:58:02.368"
}
```

```
{
  "level": "error",
  "message": "Can't fetch flow 8, cell_32",
  "serviceB": {
    "traceID": "def-uvw"
  },
  "request": "vehicle_compliance",
  "timestamp": "2023-03-16T07:58:06.246”
}
```



Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Encoded Record Tables (ERT)

USTC, CHINA ADSLAB

- Records are stored in tables partitioned by schemas

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Node	**1**	**2**		**4**	**5**	**7**	**8**	
Values	V0	L0	6,V1	V2	V3	V4	T0	1..8

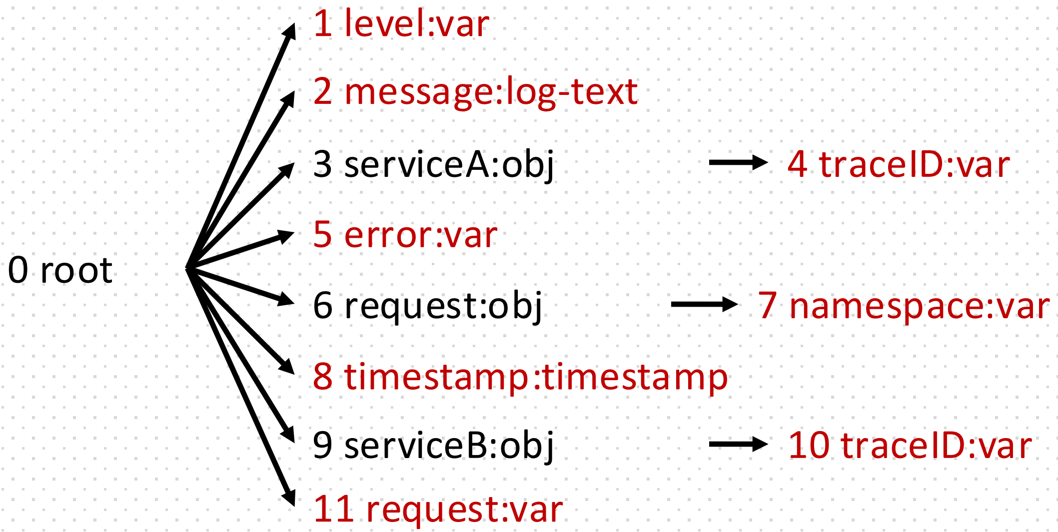
Schema 0 Encoded Record Table

Node	**1**	**2**		**10**	**11**	**8**	
Values	V0	L0	8,V1	V5	V6	T0	1..6

Schema 1 Encoded Record Table

Encoded Record Tables (ERT)

- Records are stored in tables partitioned by schemas



Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Node	1	2		4	5	7	8	
Values	V0	L0	6,V1	V2	V3	V4	T0	1..8

Schema 0 Encoded Record Table

Node	1	2		10	11	8	
Values	V0	L0	8,V1	V5	V6	T0	1..6

Schema 1 Encoded Record Table

Encoded Record Tables (ERT)

USTC, CHINA ADSLAB

- Records are stored in tables partitioned by schemas

ID	Format	ID	Format
V0	error	V4	driver_onboarding
V1	cell_32	V5	def-uvw
V2	abc-xyz	V6	Vehicle_compliance
V3	Error404		

Variable Dictionary

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Node	1	2		4	5	7	8	
Values	V0	L0	6,V1	V2	V3	V4	T0	1..8

Schema 0 Encoded Record Table

Node	1	2		10	11	8	
Values	V0	L0	8,V1	V5	V6	T0	1..6

Schema 1 Encoded Record Table

Encoded Record Tables (ERT)

USTC, CHINA ADSLAB

- Records are stored in tables partitioned by schemas

ID	Format	ID	Format
V0	error	V4	driver_onboarding
V1	cell_32	V5	def-uvw
V2	abc-xyz	V6	Vehicle_compliance
V3	Error404		

Variable Dictionary

ID	Log Type
L0	Can't fetch flow \INT, cell \DICTVAR

Log Type Dictionary

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Node	1	2		4	5	7	8	
Values	V0	L0	6,V1	V2	V3	V4	T0	1..8

Schema 0 Encoded Record Table

Node	1	2		10	11	8	
Values	V0	L0	8,V1	V5	V6	T0	1..6

Schema 1 Encoded Record Table

Encoded Record Tables (ERT)

- Records are stored in tables partitioned by schemas

ID	Format	ID	Format
V0	error	V4	driver_onboarding
V1	cell_32	V5	def-uvw
V2	abc-xyz	V6	Vehicle_compliance
V3	Error404		

Variable Dictionary

ID	Log Type
L0	Can’t fetch flow \INT, cell \DICTVAR

Log Type Dictionary

ID	Format
T0	yyyy-MM-dd’T’HH:mm:ss’.’SSS

Timestamp Dictionary

Schema ID	Node IDs
0	1 2 4 5 7 8
1	1 2 8 10 11

Schema Map

Node	1	2		4	5	7	8	
Values	V0	L0	6,V1	V2	V3	V4	T0	1..8

Schema 0 Encoded Record Table

Node	1	2		10	11	8	
Values	V0	L0	8,V1	V5	V6	T0	1..6

Schema 1 Encoded Record Table

Compression: encoded choice

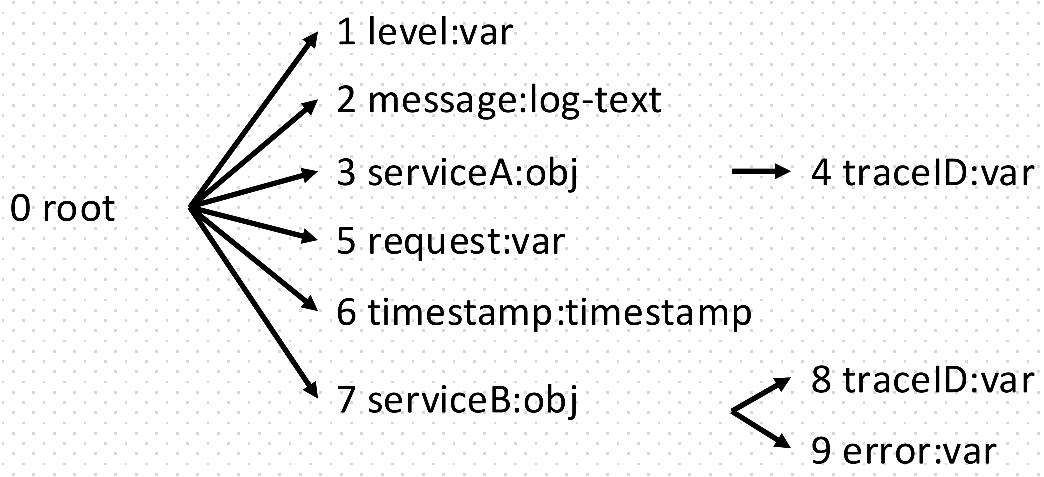
- Strings: divided into timestamp, variable, log-text
  - Timestamp: heuristics to detect if the string is a timestamp
  - Comment: highly rely on its parser

```
// Timestamps (using the `timestamp` keyword)
// E.g. 2015-01-31T15:50:45.392
timestamp:\d{4}\-\d{2}\-\d{2}T\d{2}:\d{2}:\d{2}.\d{3}
// E.g. 2015-01-31T15:50:45,392
timestamp:\d{4}\-\d{2}\-\d{2}T\d{2}:\d{2}:\d{2},\d{3}
// E.g. [2015-01-31T15:50:45
timestamp:\[\d{4}\-\d{2}\-\d{2}T\d{2}:\d{2}:\d{2}
// E.g. [20170106-16:56:41]
timestamp:\[\d{4}\d{2}\d{2}\-\d{2}:\d{2}:\d{2}\]
// E.g. 2015-01-31 15:50:45,392
// E.g. INFO [main] 2015-01-31 15:50:45,085
timestamp:\d{4}\-\d{2}\-\d{2} \d{2}:\d{2}:\d{2},\d{3}
// E.g. 2015-01-31 15:50:45.392
timestamp:\d{4}\-\d{2}\-\d{2} \d{2}:\d{2}:\d{2}.\d{3}
// E.g. [2015-01-31 15:50:45,085]
timestamp:\[\d{4}\-\d{2}\-\d{2} \d{2}:\d{2}:\d{2},\d{3}\]
// E.g. 2015-01-31 15:50:45
// E.g. Started POST /api/v3/internal/allowed for 127.0.0.1 at 2017-06-18 00:20:44
// E.g. update-alternatives 2015-01-31 15:50:45
```

Compression: encoded choice

- Strings: divided into timestamp, variable, log-text
  - Timestamp: heuristics to detect if the string is a timestamp
  - Comment: highly rely on its parser
- Integers/floating point/Boolean: directly encoded in binary form
- Array: treat as log-text, but in different dictionary to avoid pollution
- Random key and invariant values
  - Random key: if a key does **not appear in more than 1%** of the records of the archive (when writing to disk), it will be truncated
  - Invariant values: if a value **never changes**, it will be included in the MPT
  - Truncated/included time: write all the buffered data into disk

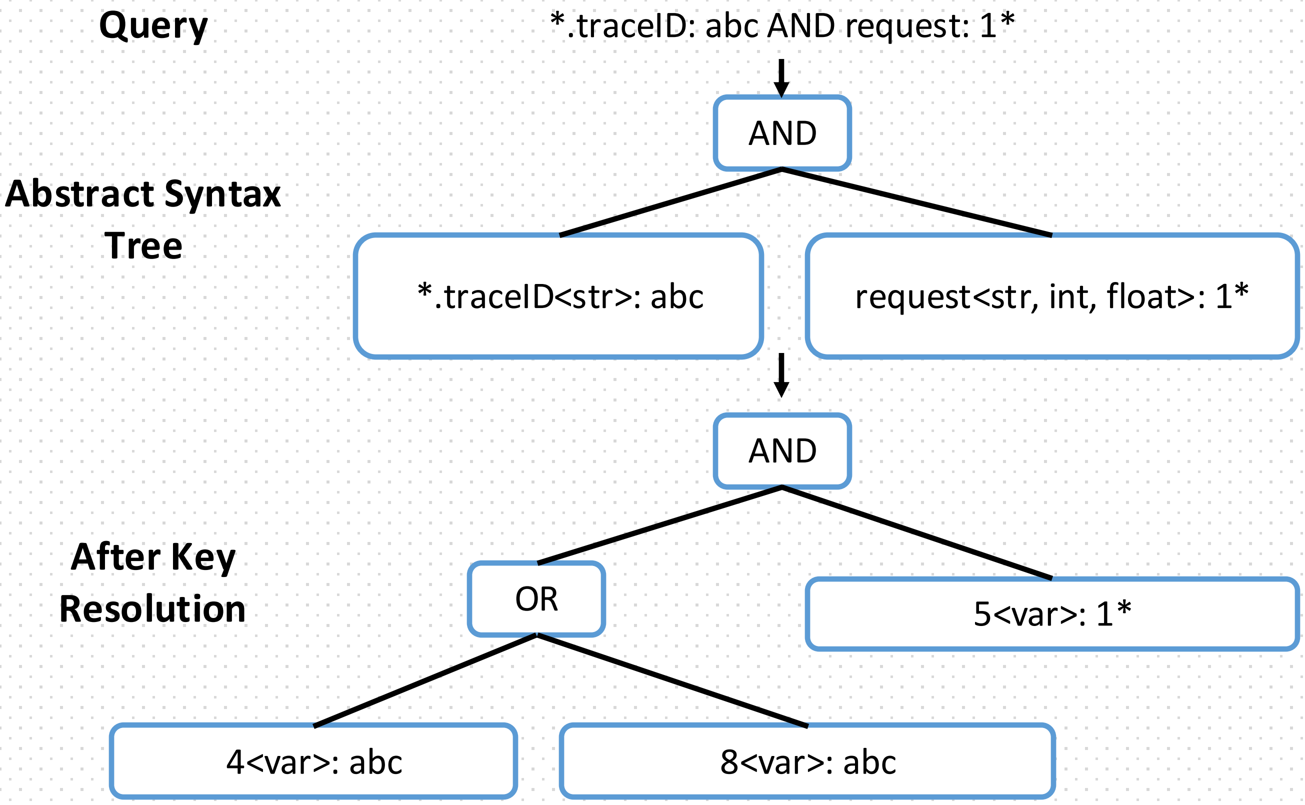
Search through μSlope



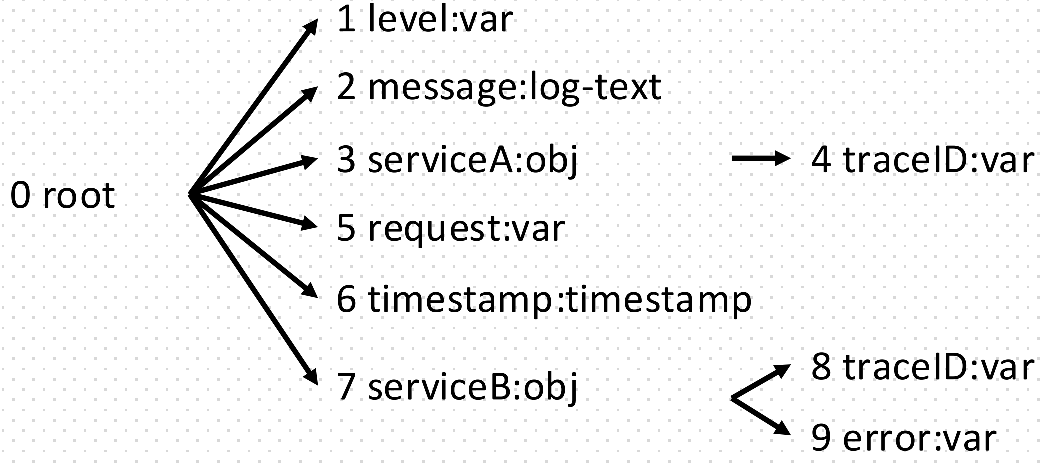
Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 6
1	1 2 6 8 9

Schema Map



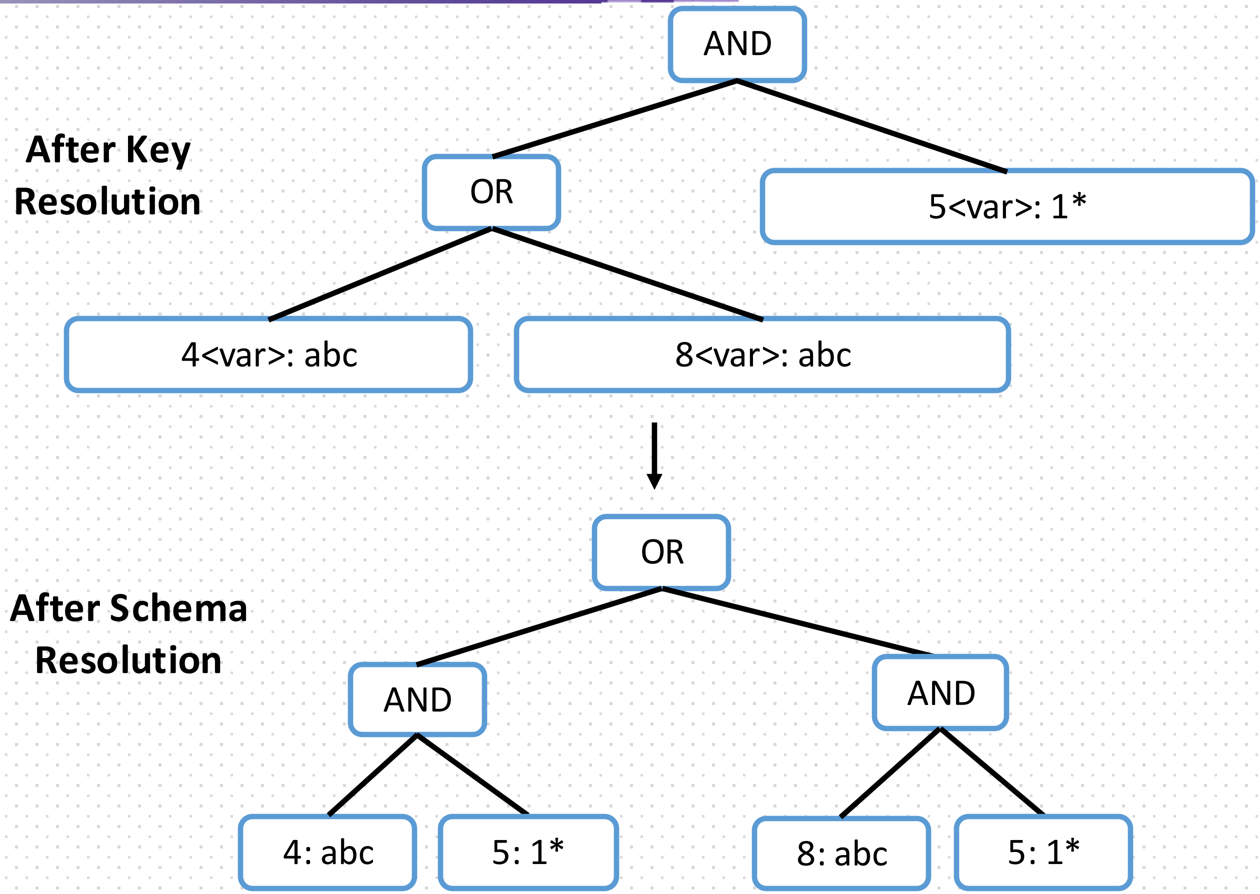
Search through μSlope



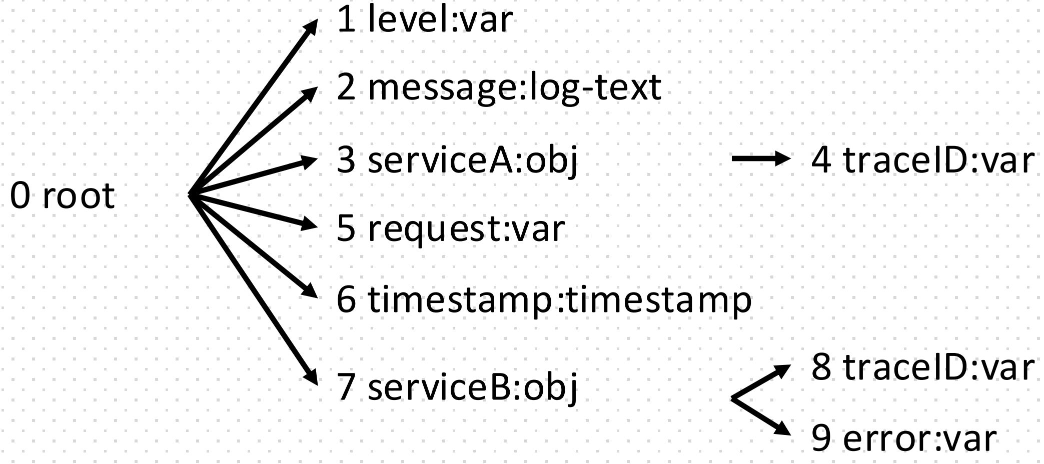
Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 6
1	1 2 6 8 9

Schema Map



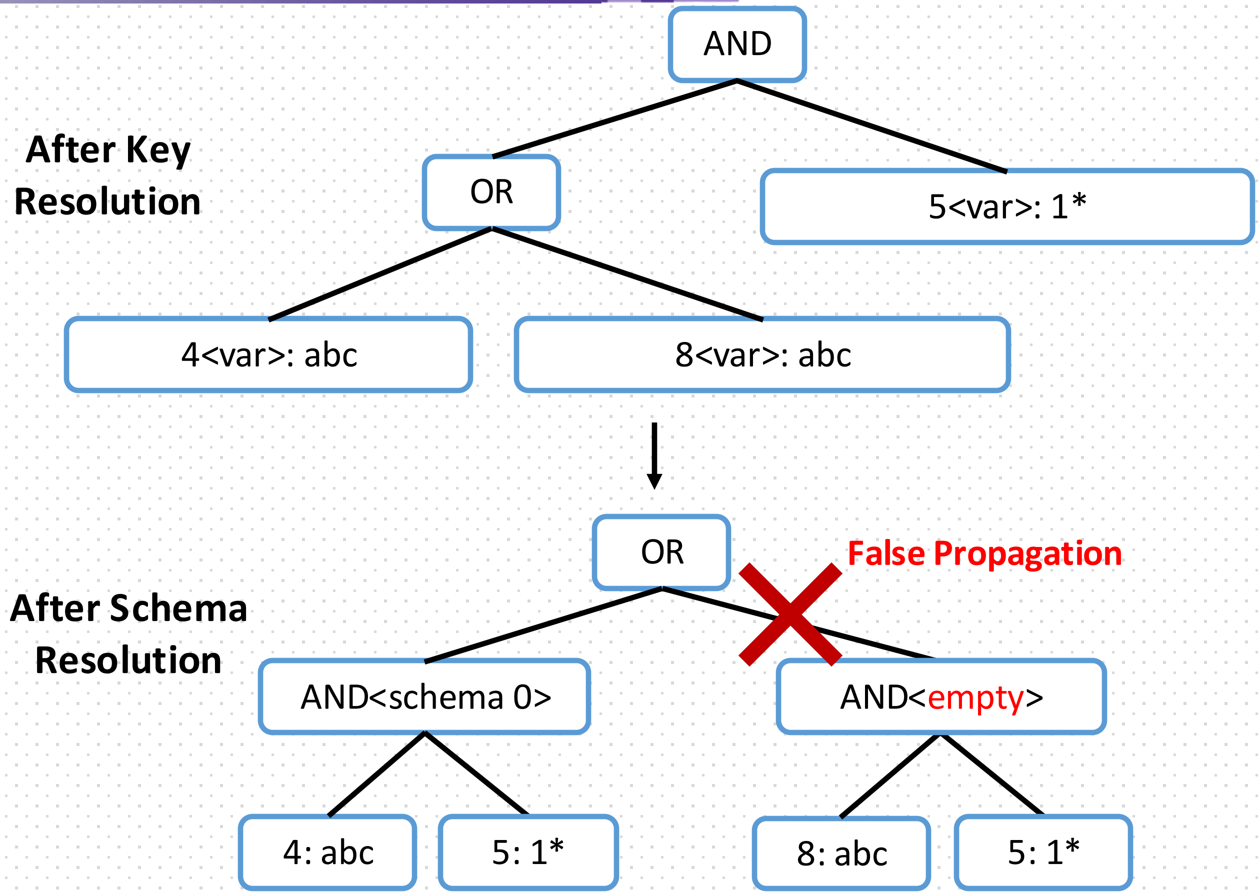
Search through μSlope



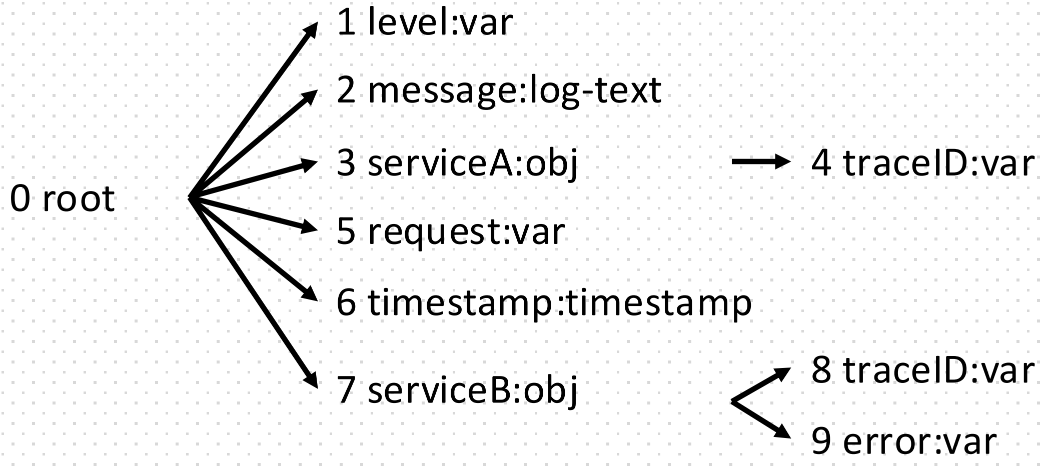
Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 6
1	1 2 6 8 9

Schema Map



Search through μSlope



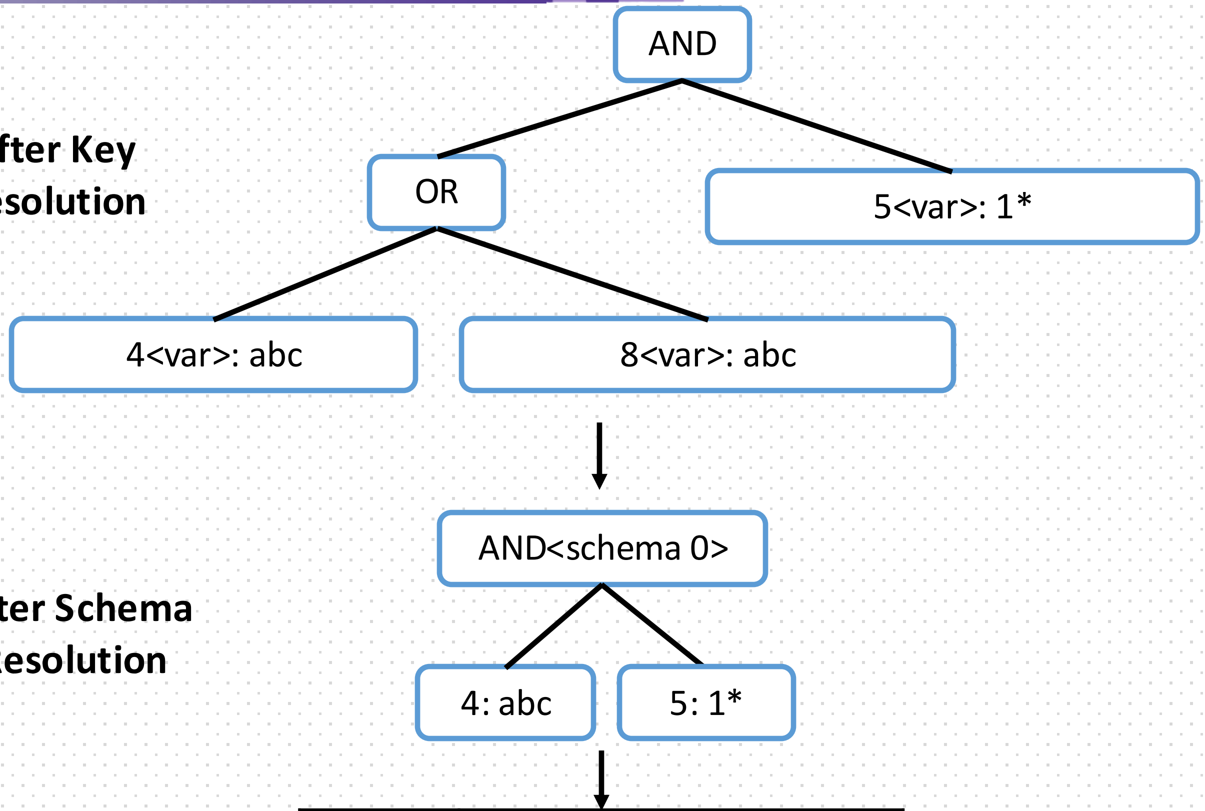
Merged Parse Tree (MPT)

Schema ID	Node IDs
0	1 2 4 5 6
1	1 2 6 8 9

Schema Map

After Key Resolution

After Schema Resolution



Search on Strings

ID	Format
…	…
V1	abc
V2	1_vehicle_compliance
…	…

Evaluation

- The evaluation mainly focus on ：
    - Compression Ratio and Speed
    - Search Performance
    - Worst Case On Synthetic Dataset
    - ~~Scalability~~ (Large-scale evaluation)

Evaluation

USTC, CHINA ADSLAB

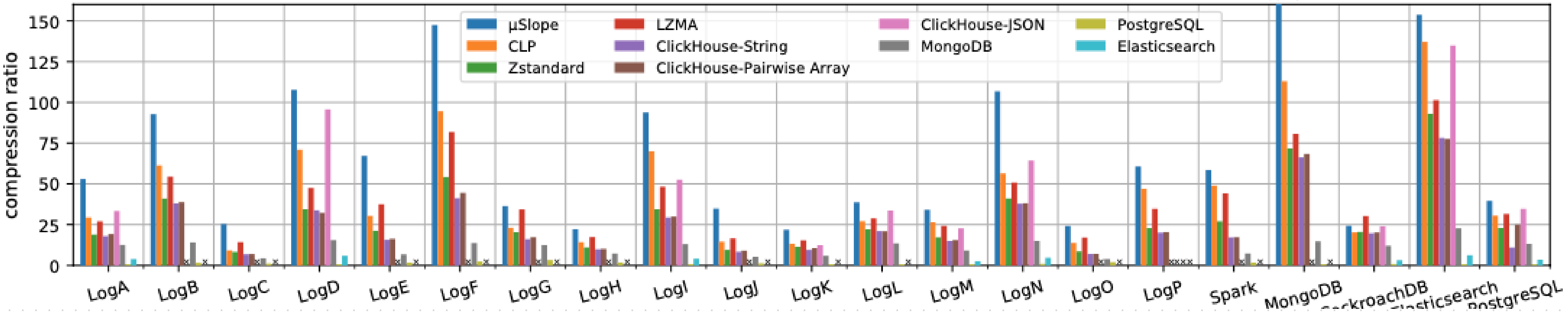
- **Experiment setup**
  - **Hardware**
    - CPU: Intel Xeon E5-2630v3
    - Memory: 128GB DDR4
    - Storage: Distributed file system (MooseFS) running on multiple 7200RPM SATA HDDs.
  - **Datasets**
    - 16 from Uber (30.0GB to 102.9GB)
    - 5 from public software (392.8MB to 64.8GB)
  - **Baselines**
    - **Systems:** CLP, MongoDB, PostgreSQL, ClickHouse, Elasticsearch
    - **Compressors:** Zstandard, LZMA

	Name	Uncompressed Size	Number of Records
Uber Logs	LogA	30.0GB	22,996,492
	LogB	47.1GB	16,606,964
	LogC	60.4GB	15,306,125
	LogD	50.7GB	58,309,754
	LogE	91.8GB	22,345,071
	LogF	102.9GB	17,251,752
	LogG	30.9GB	3,046,845
	LogH	30.8GB	11,461,221
	LogI	39.7GB	27,209,375
	LogJ	36.0GB	13,605,274
	LogK	30.2GB	57,919,224
	LogL	37.1GB	45,827,554
	LogM	36.5GB	42,206,452
	LogN	38.0GB	22,307,407
	LogO	38.6GB	4,438,786
	LogP	38.3GB	34,840,347
Public Logs	Spark	2.0GB	1,011,651
	MongoDB	64.8GB	186,287,600
	CockroachDB	9.8GB	16,520,377
	elasticsearch	8.0GB	140,012,234
	PostgreSQL	392.8MB	1,000,000

Compression ratio

- **µSlope outperforms all other baselines**
  - The average compression ratio of µSlope is 68.1:1

CLP	Zstandard	LZMA	ClickHouse-String	ClickHouse-Pairwise Array
1.5X	2.3X	1.7X	2.75X	2.62X
ClickHouse-JSON	**MongoDB**	**PostgreSQL**	**Elasticsearch**	
1.34X	6.10X	16.5X	15.71X	

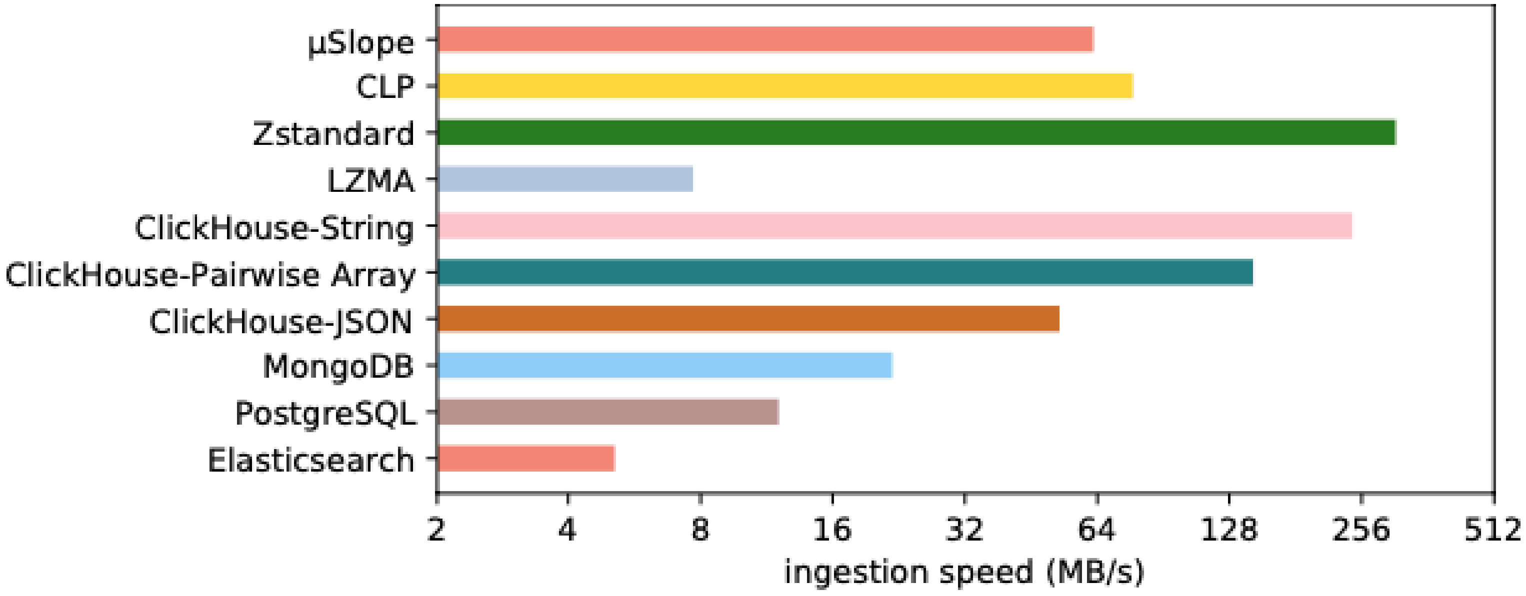


Compression Ratio on different dataset

Compression Ingestion Speed

USTC, CHINA ADSLAB

- **µSlope is faster than all fully-parsed JSON tools except CLP**
  - Outperforming ClickHouse-JSON, MongoDB, PostgreSQL and Elasticsearch by 19.3%, 186.7%, 419.8%, 1127.3%



Average ingestion speed (log scale)

Search Performance

USTC, CHINA ADSLAB

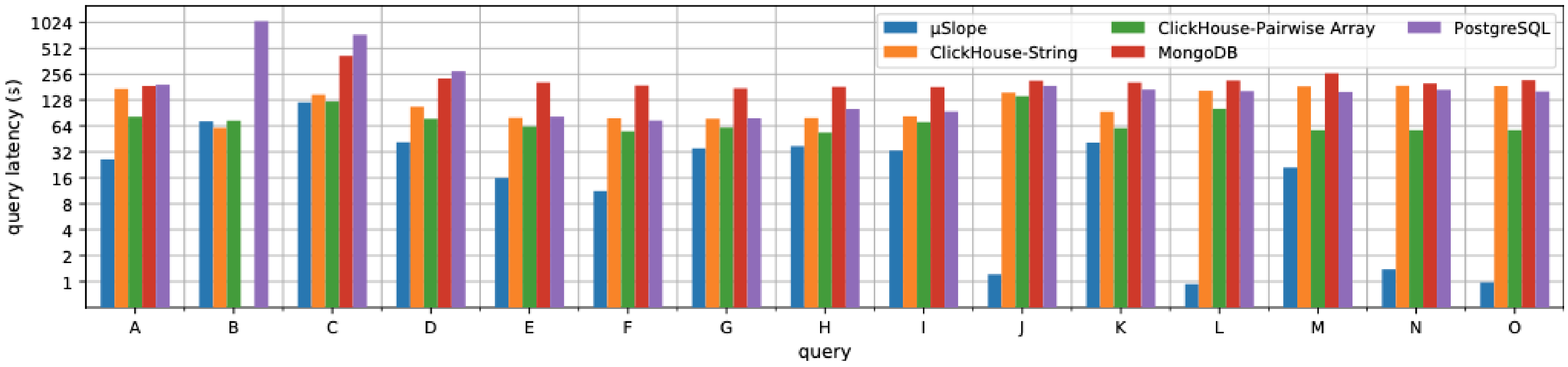
- **Baselines**
  - ClickHouse, MongoDB, PostgreSQL
- **Dataset & query**
  - 15 queries on Uber LogF, LogO and MongoDB queries

	Queries for LogF
A	zone:... AND NOT @reserved.collector.filename:stdout AND runtime_env:staging
B	*: "d...-...-...-...-...9"
C	level: error AND message: d...*
D	timestamp >date("2022-04-14T08:00:00.000") AND timestamp <date("2022-04-14T08:15:00.000")
	Queries for LogO
E	headers.x-tenancy:testing* AND NOT headers.x-tenancy: testing/.../4...-...-...-...6d AND headers.caller-procedure:"fareEstimateV2" AND headers.x-source:public
F	headers.x-region-name:... AND headers.x-tenancy:"production" AND caller:*create*
G	level: error AND NOT @reserved.collector.filename: executor AND runtime_env:production AND partition: compute-... AND instance: 3...5 AND mesos_executor_id: t...5-6
H	level: error AND message: "Error handling inbound request."
I	glue.handler.method: get_ranked_products AND env: production AND level: error
	Queries for MongoDB logs (public dataset)
J	attr.tickets: *
K	id: 22419
L	attr.message.msg: log_release* AND attr.message.session_name: connection
M	ctx: initandlisten AND (NOT msg: "WiredTiger message" OR attr.message.msg: log_remove*)
N	c: WTWRTLOG AND attr.message.ts_sec >1679490000
O	ctx: FlowControlRefresher and attr.numTrimmed: 0

Queries used in experiments

Search Performance

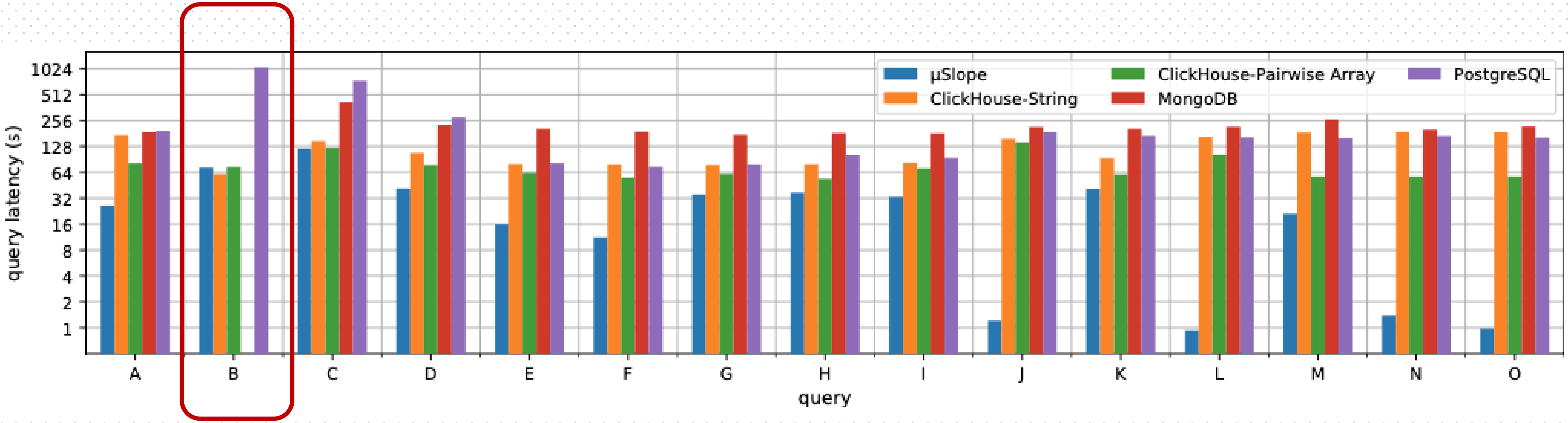
- **μSlope outperforms all other tools on 14 queries**
  - 2.5x faster than the fastest setup of ClickHouse, 6.7x faster than MongoDB, 8.1x faster than PostgreSQL



Query Latency

Search Performance

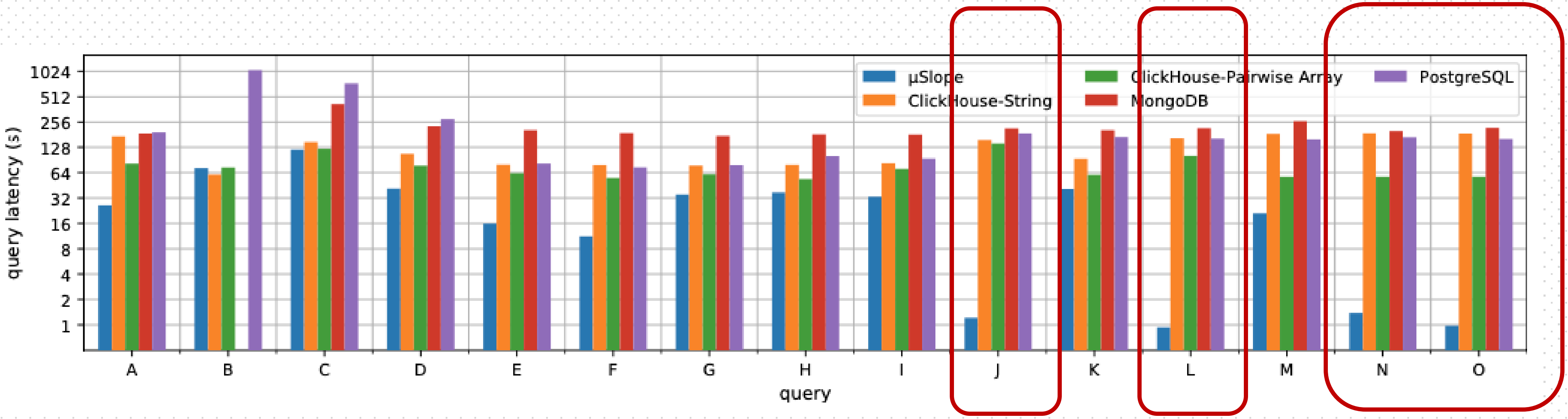
- ***µ*Slope outperforms all other tools on 14 queries**
  - 2.5x faster than the fastest setup of ClickHouse, 6.7x faster than MongoDB, 8.1x faster than PostgreSQL
  - For Query B, *µ*Slope needs to scan all the ERTs and decode them



Query Latency

Search Performance

- **μSlope outperforms all other tools on 14 queries**
  - 2.5x faster than the fastest setup of ClickHouse, 6.7x faster than MongoDB, 8.1x faster than PostgreSQL
  - For Query B, μSlope needs to scan all the ERTs and decode them
  - For Query J/L/N/O, there are only a small number of schemas matches



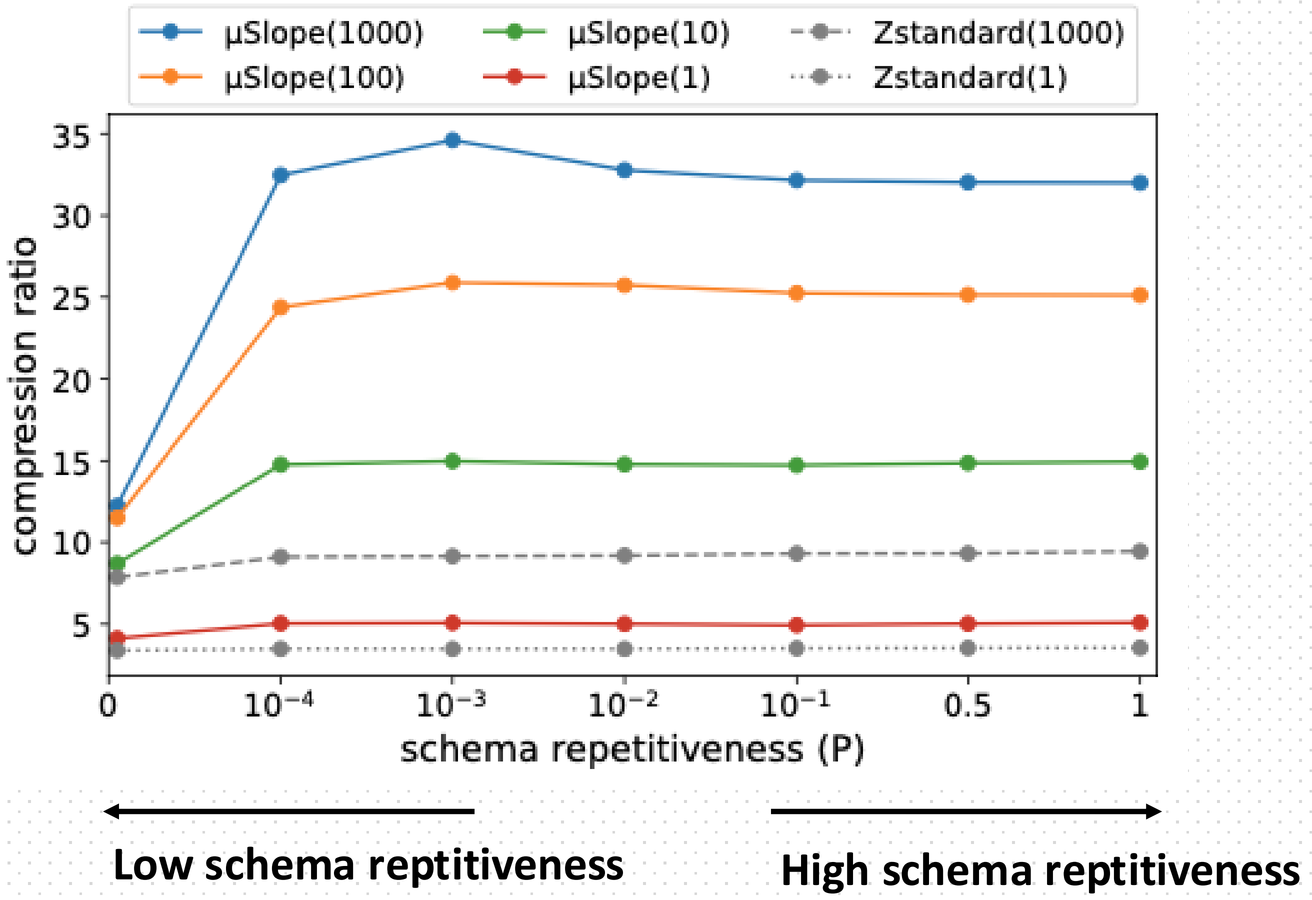
Query Latency

Synthetic Evaluation

- **Demonstrate the boundaries of μSlope's capabilities**
  - Since the efficiency of µSlope relies on the repetitiveness of schemas and variable values
  - All KV set to: "UUID : UUID", each record gets 20 keys, 670K records totally
- **Repetitiveness of variable values**
  - $repetation\ ratio = \frac{number\ of\ all\ variable\ values}{number\ of\ unique\ variable\ values}$
- **Repetitiveness of schemas**
  - the n-th most frequent schema appears in $P \times (1 - P)^n$ of the records (where n starts from 0)

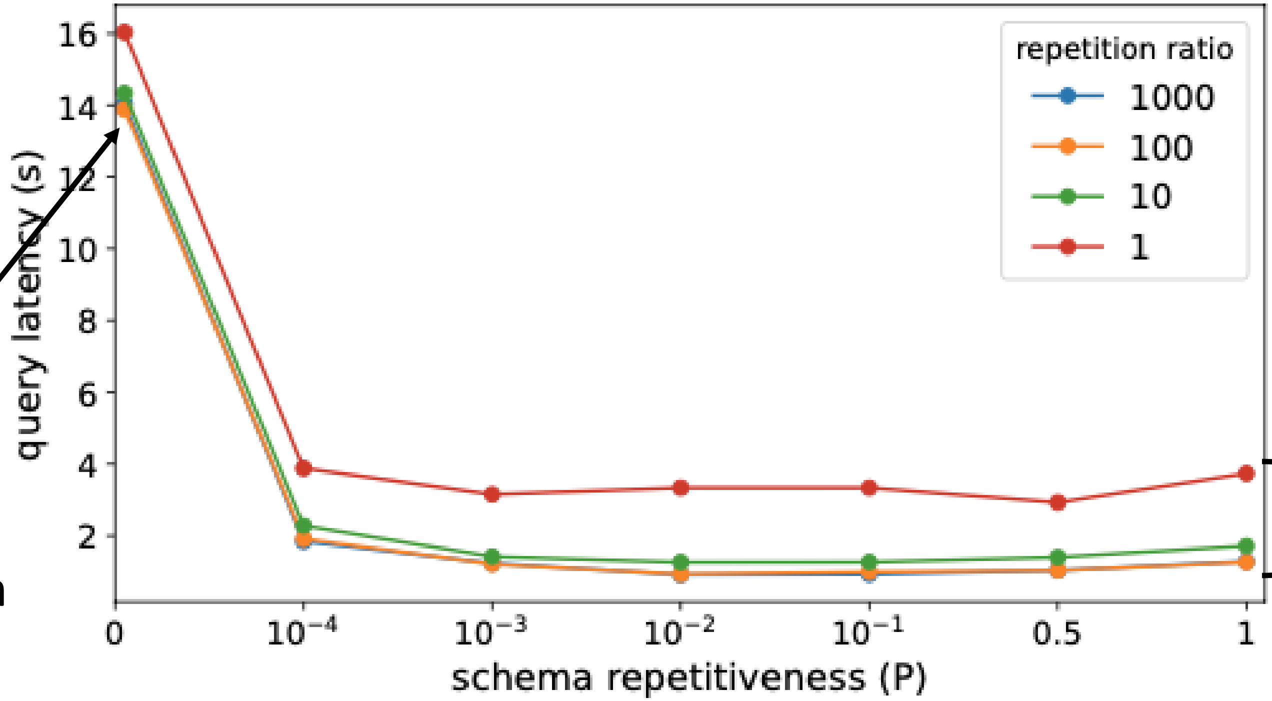
Compression ratio

- **µSlope outperforms Zstandard**
    - In the extreme case where P approaches 0, the compression ratio drops notably
    - The compression ratio quickly increases as P increases to the next smallest value (10^{-4}) and remains relatively stable



Search performance

- Query: "*: UUID"



we have a large number of small ERTs where each has only one record.

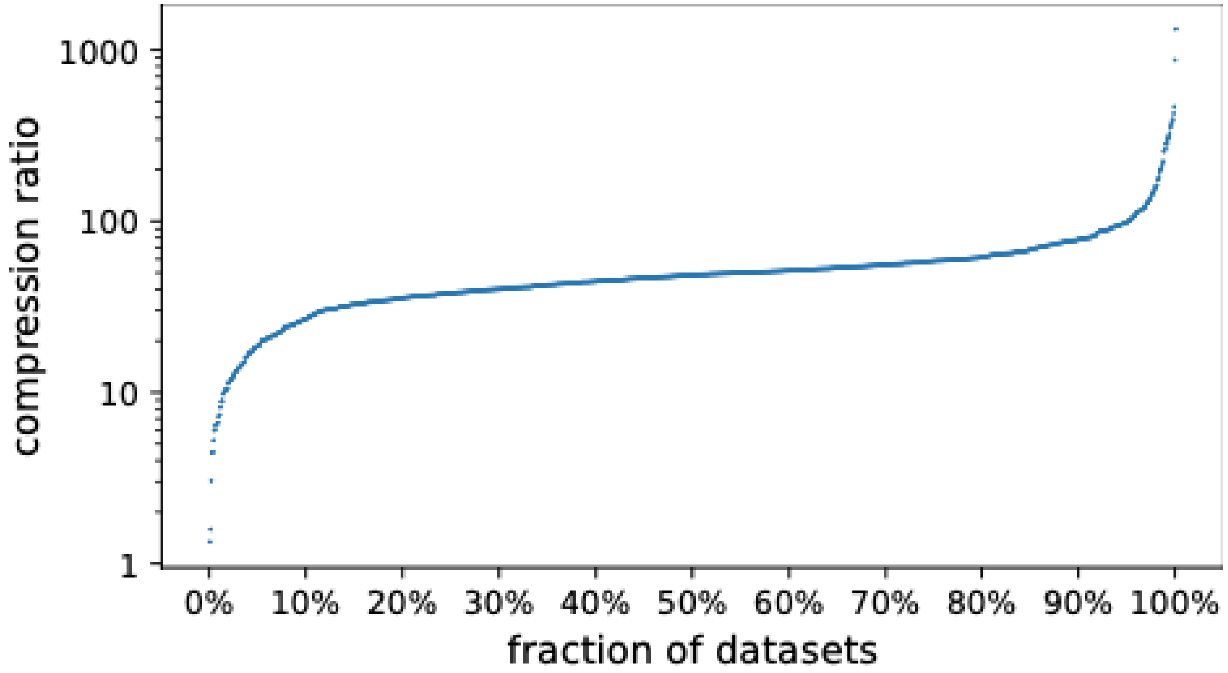
Lower repetition ratio, introducing a **constant overhead** before returned

Low schema reptitiveness

High schema reptitiveness

Large-scale compression

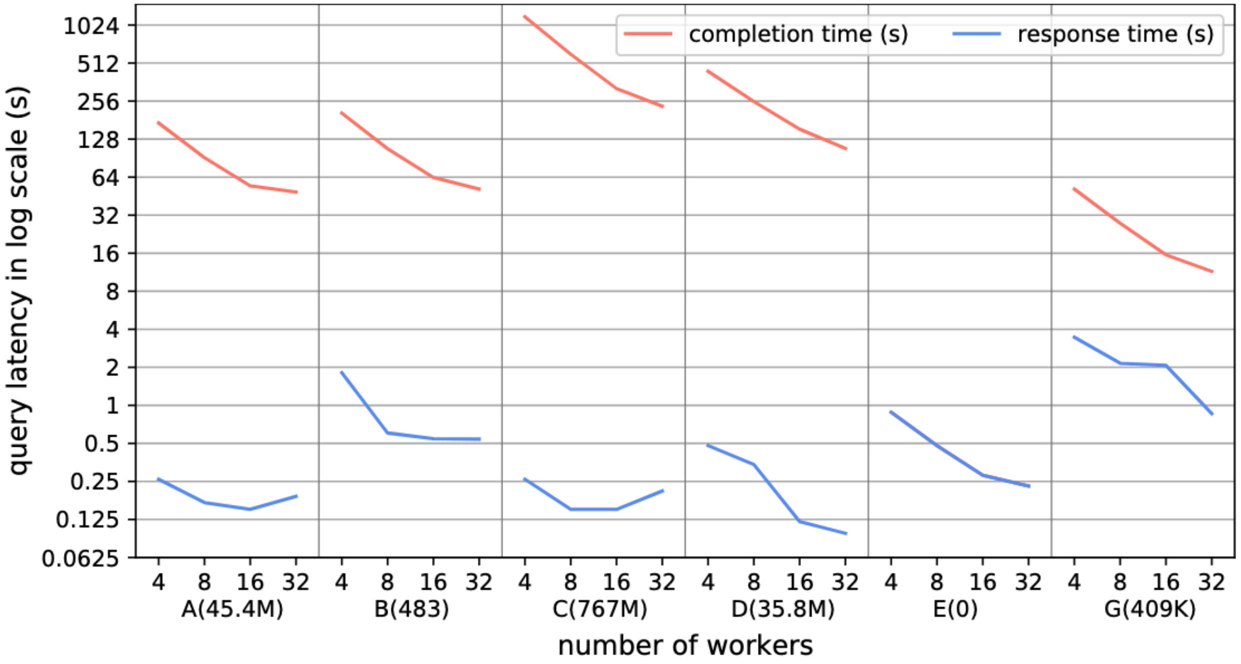
- **Dataset:** 434TB of production logs from Uber to evaluate compression.
  - Average compression ratio: 30.5:1
  - The outliers with low compression ratio contain large amounts of random non-repeating binary data such as base64 encoded binary data and UUIDs.



Compression ratio distribution

Search Scalability

- **Dataset:** 26.2TB subset from Uber’s LogF to evaluate search scalability.
  - Run on 8 containers, each has access to 96 cores, 2TB of network attached SSD, and 32GB of RAM, 2,155 archives are evenly distributed across these machine.
  - All queries scale well to 16 workers but have limited scalability to 32 workers.

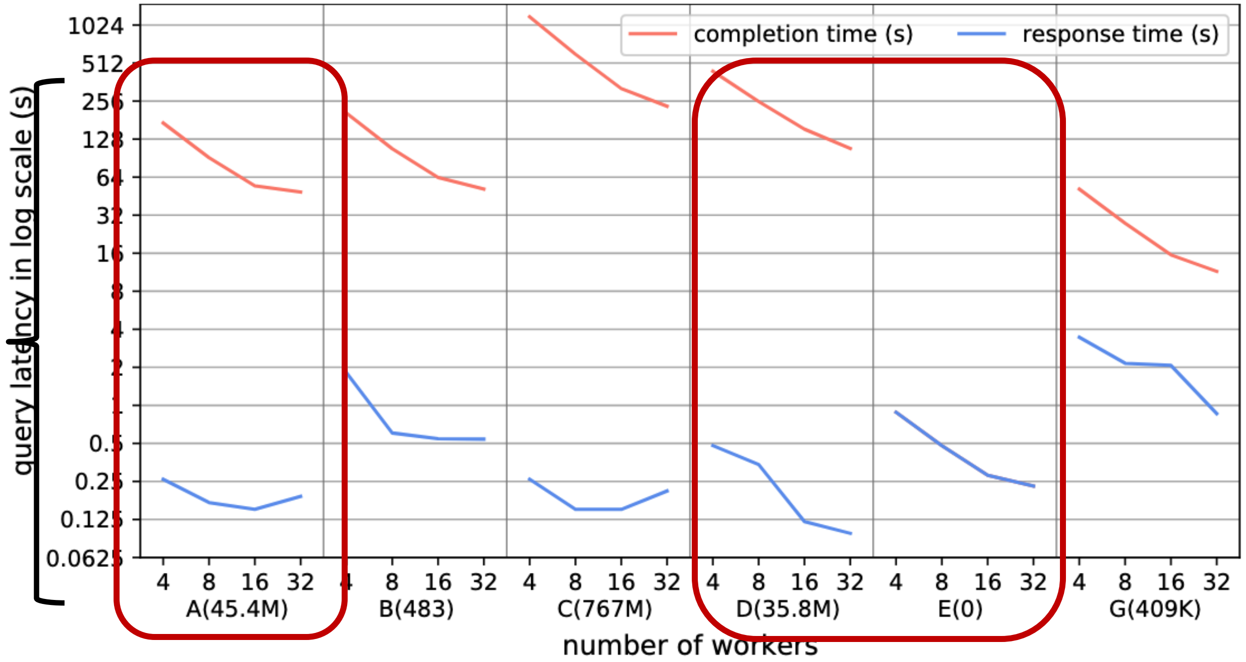


Query Completion Time & Response Time

Search Scalability

- **Dataset:** 26.2TB subset from Uber's LogF to evaluate search scalability.
  - Run on 8 containers, each has access to 96 cores, 2TB of network attached SSD, and 32GB of RAM, 2,155 archives are evenly distributed across these machine.
  - All queries scale well to 16 workers but have limited scalability to 32 workers.

Limited by different skewness of dataset



Query Completion Time & Response Time

Summary

- μSlope: An efficient semi-structured log management system that
  - Handles dynamic schema structures
  - Achieves unprecedented compression ratio
  - Allows search without full decompression
- Pros
  - Good & reasonable design motivated by complete analysis
  - Good writing
  - Lots of evaluation (but lack of explanation)
- Cons
  - Limited on read-only workload & highly rely on repetition ratio
  - Still lack of functionality to deal with other query operation
  - Lack of LogGrep Baseline