하나고등학교
Hana Academy Seoul

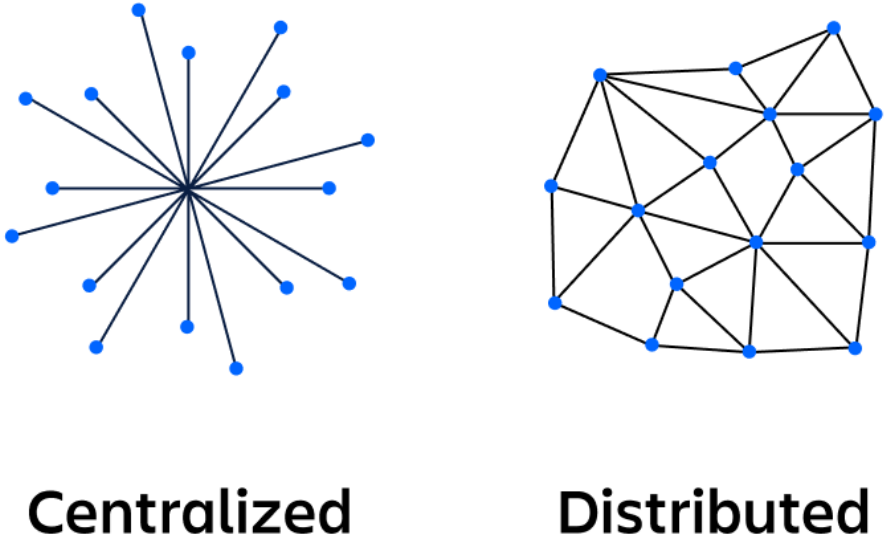
분산컴퓨팅 환경에서 범용 고유 식별자를 대체하기 위한 사전순 정렬 가능한 고유식별자의 제안

Proposal of Lexicographically Sortable Unique Identifier to replace Universal Unique Identifier (UUID) in Distributed Computing System

하나고등학교 권동한. Ryan Donghan Kwon, Hana Academy Seoul. - kznm.develop@gmail.com

분산컴퓨팅 환경에서의 고유 식별자
Unique identifiers in a distributed computing environment

- 중앙 집중식(Centralized) 시스템에선 Ticket Server의 오버헤드 및 SPoF 발생
- UUID 등과 같은 고유 식별자가 널리 사용되고 있음



Centralized Distributed

기존 고유 식별자의 분석
Analysis of existing unique identifiers - UUID, ULID, Snowflake

- **UUID** (1) 100ns 단위 60-bit Timestamp
  (2) 4-bit Version
  (3) 16-bit Clock Sequence
  (4) 48-bit Node Identifier (Mac Address)

  `2acf084b-07a3-4568-a097-4114d00a98fd`

- **ULID** (1) 1ms 단위 48-bit Timestamp
  (2) 80-bit Randomness

  `01ARZ3NDEKTSV4RRFFQ69G5FAV`

- **Snowflake** (1) 1ms 단위 41-bit Timestamp
  (2) 10-bit Machine ID
  (3) 12-bit Machine Sequence

  `1584097399939268608`

사전순 정렬 가능한 고유 식별자의 제안
Proposal of Lexicographically Sortable Unique Identifier - LSID

- UUID - **너무 길다!** 데이터베이스* 삽입 시 성능 저하 이슈 발생
- ULID - **너무 부실하다!** 시간을 제외하고 어느 의미있는 데이터도 담고 있지 않음
- Snowflake - **너무 복잡하다!** Ticket Server**의 필요로 인해 SPoF 발생
- 짧고 알차며 복잡하지 않은 고유식별자의 필요성

APACHE ZooKeeper™

MySQL®

* https://stackoverflow.com/questions/2365132/uuid-performance-in-mysql
** https://github.com/twitter-archive/snowflake

사전순 정렬 가능한 고유 식별자의 제안
Proposal of Lexicographically Sortable Unique Identifier - LSID

00jtx-04fecrkm-0cgm-3n
① ② ③ ④

- (1) 25-bit Date
- (2) 100ns 단위 40-bit Timestamp
- (3) 20-bit Worker ID (Hashed Mac Address) 1,048,576
- (4) 10-bit Sequence ID 1,024

* http://www.crockford.com/base32.html

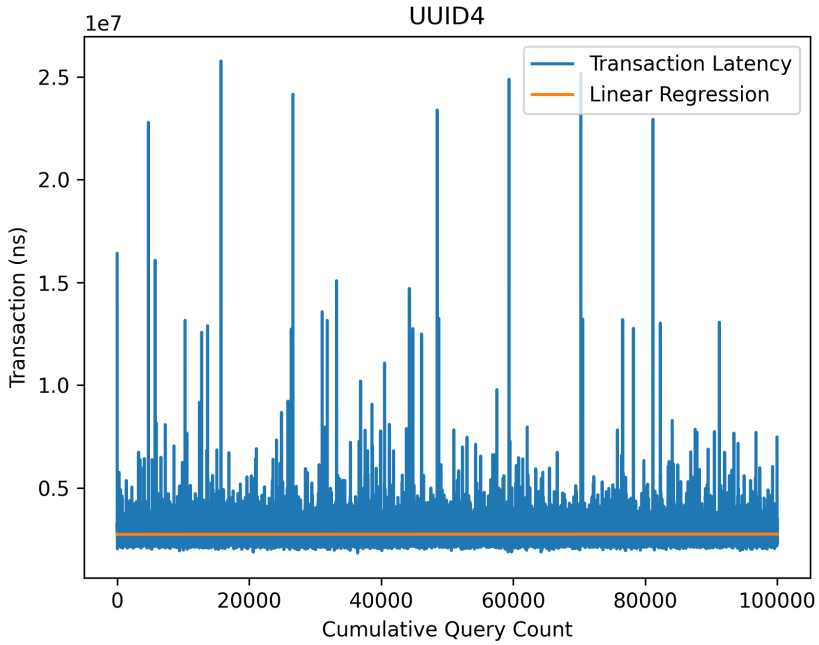
사전순 정렬 가능한 고유 식별자의 정의
Define of Lexicographically Sortable Unique Identifier - LSID

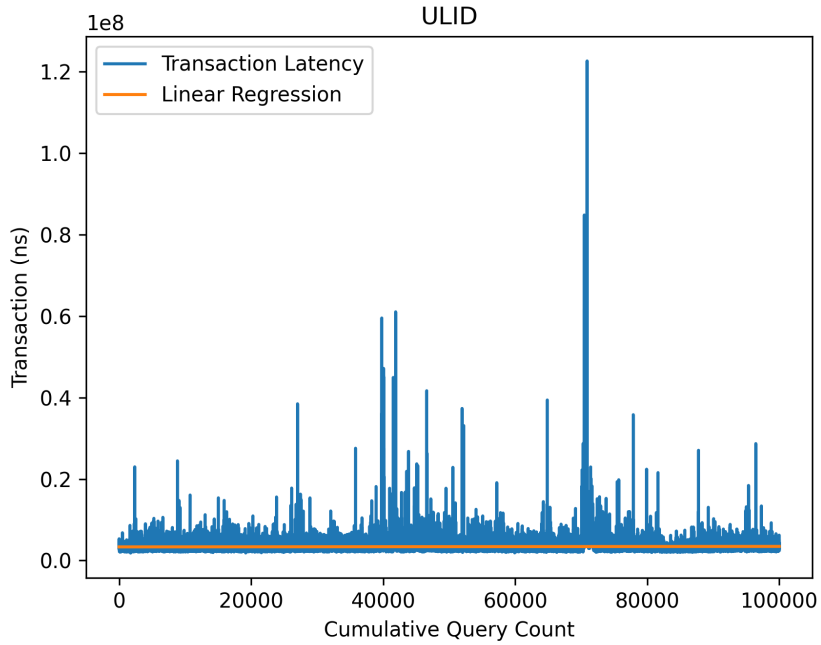
- Crockford’s Base32로 인코딩한 96-bit 22 Character String → 가독성 확보
- 100ns 단위 Timestamp를 채용한 고유식별자 중 가장 짧음
- 일(Day) 단위와, 해당 일 0시부터 계산되는 100ns Timestamp 분리 → 공간효율성 확보
- Epoch+91,929Y까지 표기 가능 → UUID 대비 1000배, ULID 대비 9배 이상의 효율

$$\log_{32} 86400 \times 10^7 = 7.9304480712299$$

기존 고유식별자 대비 LSID의 성능상 이점
Performance benefits of LSIDs over existing unique identifiers

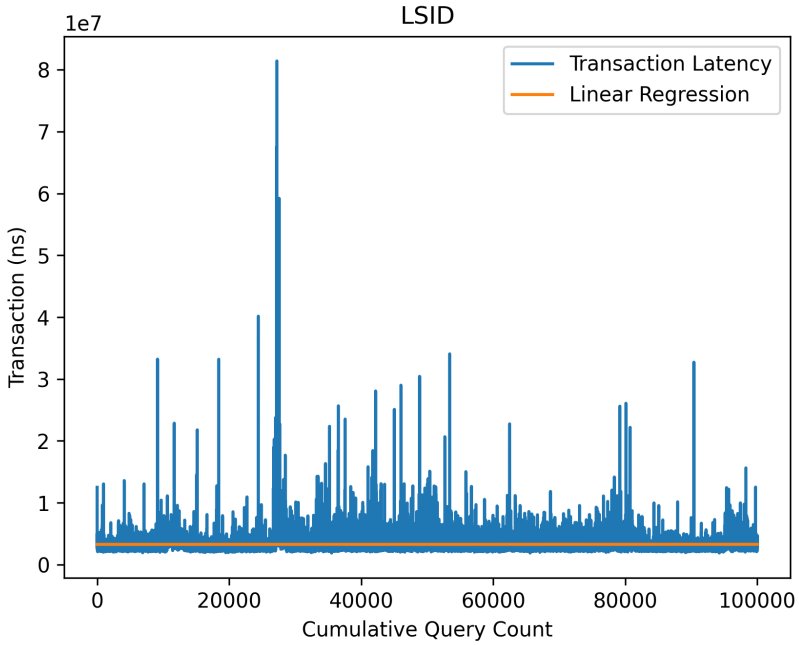
- Transaction Latency of UUID, ULID. (*Using MySQL, Set up on Localhost)





기존 고유식별자 대비 LSID의 성능상 이점
Performance benefits of LSIDs over existing unique identifiers

- Transaction Latency of LSID and Increase Rate(s)



	기울기	절편
UUID	0.136347408397646	3339880.93448226
ULID	0.185529453792534	3348060.9900751
LSID	0.0945419569594488	3273360.71942301

표 1. UUID, ULID, LSID의 100,000회
Insert Transaction Latency에 대한 선형 회귀 분석 결과

LSID의 이론적 한계 및 극복 방안
Theoretical Limitations and How to Overcome the LSID

- Mac Address 기반 Machine ID를 사용할 시, 48-bit Node ID를 20-bit로 Hash하며 해시 충돌의 가능성을 무시할 수 없음 → 분산 컴퓨팅 환경에서 사용되는 Kubernetes, Docker Swarm 등의 Orchestration Tool에서 제공하는 컨테이너 ID를 사용.

- Sequence ID의 Pool이 2^10 - 1,024로 크지 않아 충돌 가능성이 있음 → 100ns 단위 Timestamp 사용으로 Task 당 LSID를 생성할 시 동일한 Timestamp 및 Machine ID를 가지는 Key에서 중복으로 Seq ID가 생성될 가능성이 매우 낮음. getrandombits 표준 구현체를 기본적으로 사용하나, Collision-safe한 ID의 생성이 필요한 경우 CPU Clock Sequence를 사용함으로서 원천적으로 충돌을 방지할 수 있음.