Efficient Adaptive Radix Tree for CXL-based Memory System

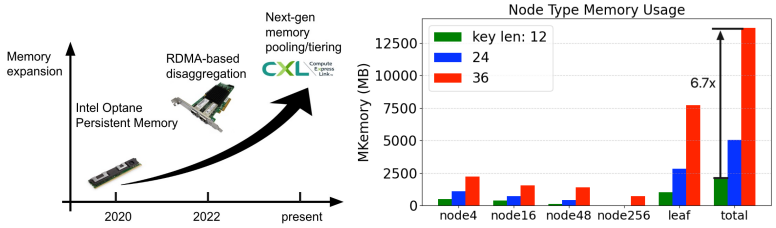
Yuze Li, Sumit Monga, Kirshanthan Sundararajah, Ali R. Butt, Huaicheng Li

VIRGINIA TECH

Large Memory Needs for Database Indexing

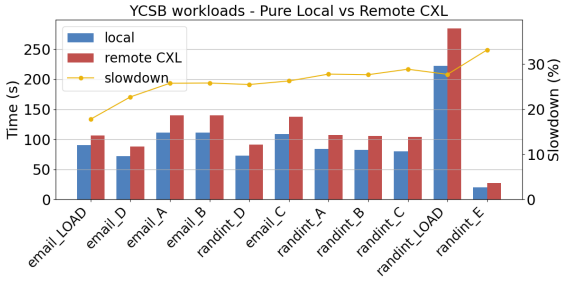
(1). **Indexes** are a major contributor to **memory footprint** in main-memory OLTP systems (over 50% of the total memory used in DB).

(2). **CXL** is the next generation memory expansion technology.



Naïve Tiering for ART is Inefficient

(1). Access latency for ART is due to **pointer chasing** (up to **28%** slowdown).

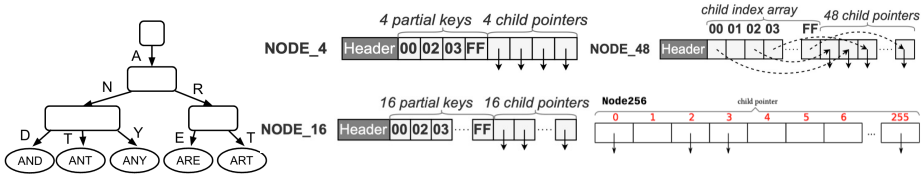


Adaptive Radix Tree (ART)

(1). ART indexing **widely used** in DuckDB, PostgreSQL, Hyper.

(2). ART inner nodes are categorized into four different types.

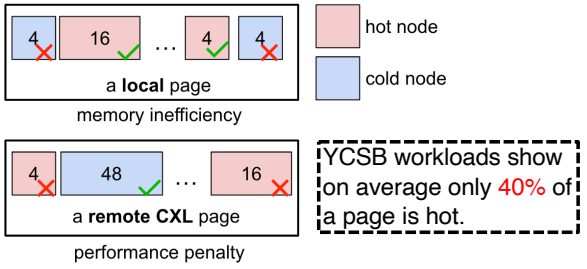
(3). Memory usage scales with average ken length.



Problem of state-of-the-art OS page-based tiering solutions:

(1). **Agnostic** to ART structural **semantics**

(2). **Coarse grained** on **page** level (**node** level hotness not captured)

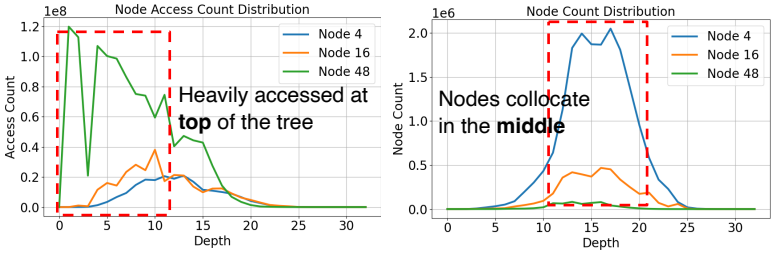


YCSB workloads show on average only 40% of a page is hot.

Exploiting ART Structural Characteristics for Better Placement

Insights: Inner node **types** show **distinct** and **stable** access **profiles**

- **High** fanout nodes show skewed distribution & higher access.
- **Low** fanout nodes show normal distribution & lower access.



Design 1. Historical information-guided static allocation

(1) Based on short-term access history, calculate slowdown density

l : level in the tree

t : node type (4, 16, 48, 256)

ϕ_t : **slowdown sensitivity** for node type t

$h_{l,t}$: total **hit count** for node type t at level l

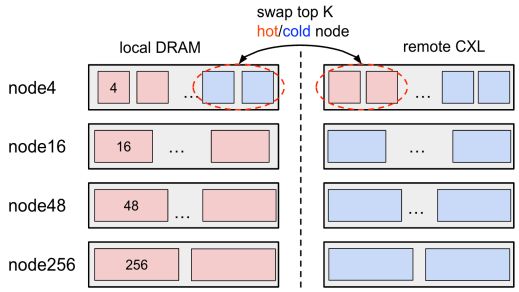
$n_{l,t}$: **number of nodes** of type t at level l

s_t : **bytes per node** for type t

$$\text{density}l, t = \frac{\text{slowdown cost avoided}}{\text{memory bytes required}} = \frac{\phi_t \cdot h_{l,t}}{n_{l,t} \cdot s_t}$$

(2) Node static allocation based on **local DRAM budget**

Design 2. Online node swapping



Design 3. Profile-guided prefetching

(1) Leverage techniques in previous work like APT-GET (EuroSys '22), RPG² (ASPLOS '24) to find candidate **prefetch distance** and **prefetch injection sites**

(2) Dynamically **tune** prefetch instructions in real time by detecting hot path

(3) Rely on Linux **ptrace** to perform actual code insertion

Progress & Future Work

(1) Tuning static placement and online node swapping performance.

(2) Measuring state-of-the-art prefetching strategy performance on ART.

Future work:

(1) Exploit node replication on both local / remote to reduce migration overhead.

(2) Extend to other index data structure like B+-tree and skip list.