

# An Adaptive Index for Hierarchical Database Systems

Rafael Kallis

BSc Thesis

February 17, 2018



University of  
Zurich UZH

Department of Informatics

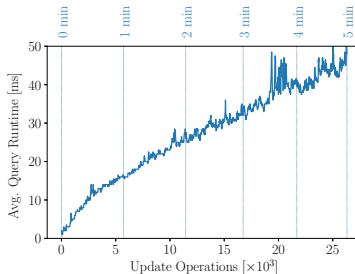
- Skewed and update-heavy workloads trigger repeated structural index updates over a small subset of nodes to the index
- Informally, a frequently added or removed node is called volatile
- Volatile nodes decrease index update performance

## The Workload-Aware Property Index (WAPI):

- Detects such volatile nodes
- Stops pruning volatile nodes
- Significantly improves update throughput

## Unproductive nodes are an unwanted byproduct:

- When the workload changes, volatile nodes cease to be volatile
- They do not contribute to a query match and contain no data
- They waste space and slow down queries over time

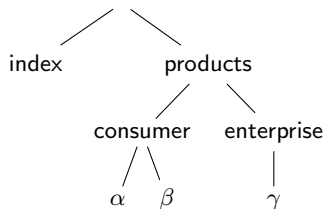


In this thesis we:

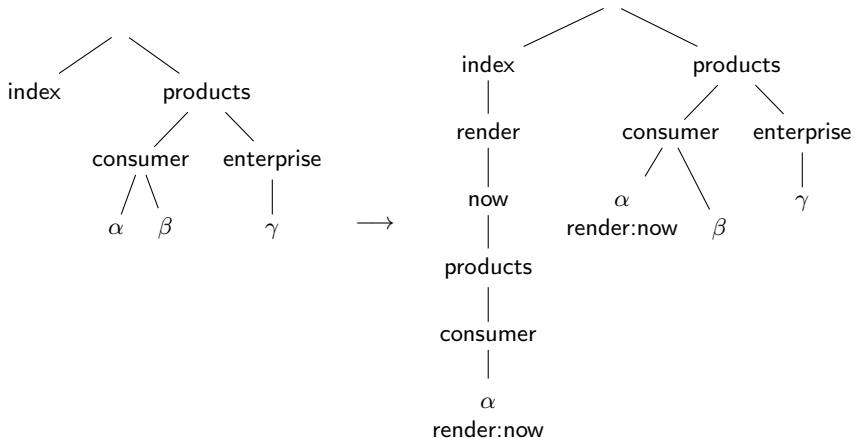
- Design and implement two solutions in order to mitigate unproductive nodes
- Analyze factors impacting the production of unproductive nodes
- Empirically evaluate and compare our two solutions

## Example: e-commerce platform

## Hierarchical database for an e-commerce platform



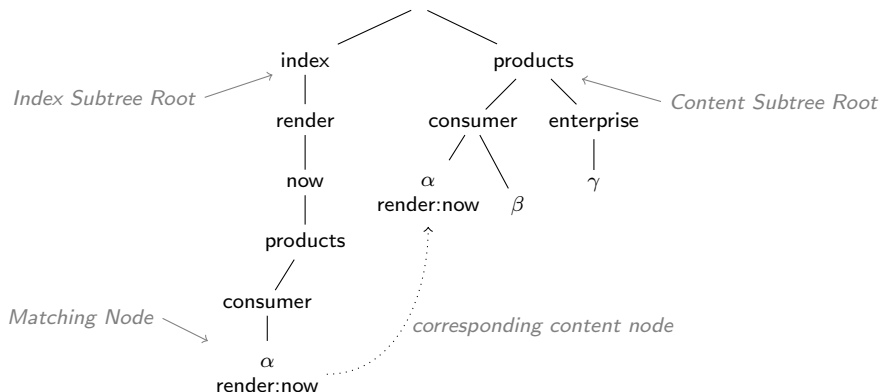
## HTML pre-rendering





# Workload-Aware Property Index

# Hierarchical Database with WAPI



We denote node  $n$ 's property  $k$  as  $n[k]$  and node  $n$ 's descendants as  $desc(n)$ .

### Definition (CAS Query)

Given node  $m$ , property  $k$  and value  $v$ , a CAS query  $Q(k, v, m)$  returns all descendants of  $m$  which have  $k$  set to  $v$ , i.e.,

$$Q(k, v, m) = \{n \mid n \in desc(m) \wedge n[k] = v\}$$

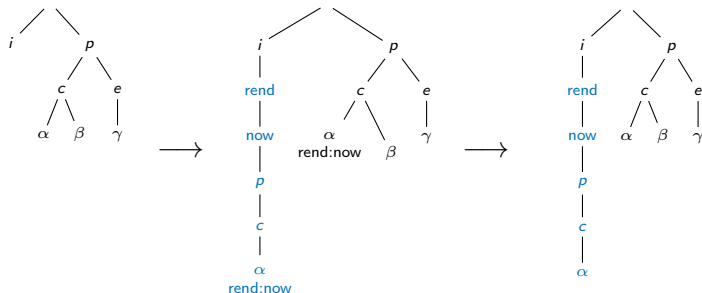
## Definition (Volatile Node)

Index node  $n$  is volatile iff  $n$ 's volatility count is greater or equal than the volatility threshold  $\tau$ , i.e.,

$$\text{volatile}(n) \iff \text{vol}(n) \geq \tau$$

- volatility count  $\text{vol}(n)$  is the number of insertions and deletions of node  $n$  inside a sliding window.

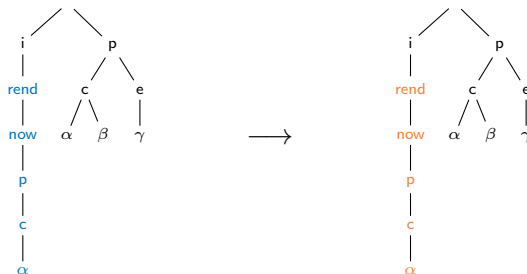
# Index nodes becoming volatile



volatile

# Unproductive Nodes

# Index nodes becoming unproductive



Volatile nodes might cease to be volatile and become unproductive

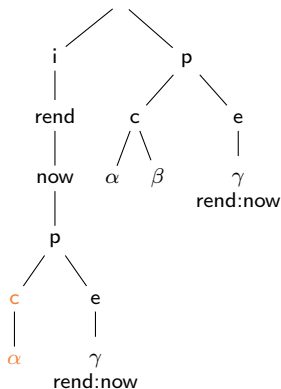
volatile  
unproductive

## Definition (Unproductive Node)

Index node  $n$  is unproductive iff  $n$ , and any descendant of  $n$ , is neither matching, nor volatile, i.e.,

$$\text{unproductive}(n) \iff \forall m (m \in (\{n\} \cup \text{desc}(n)) \implies (\neg \text{matching}(m) \wedge \neg \text{volatile}(m)))$$





$$\text{unproductive}(n) \iff \forall m(m \in (\{n\} \cup \text{desc}(n))) \implies (\neg \text{matching}(m) \wedge \neg \text{volatile}(m))$$

volatile  
 unproductive

The number of unproductive nodes depends on:

- Volatility threshold  $\tau$
- Sliding window length  $L$
- Workload skew  $s$
- Update operations per second

Unproductive index node cleaning, we propose:

- Periodic Garbage Collection (GC)
- Query-Time Pruning (QTP)

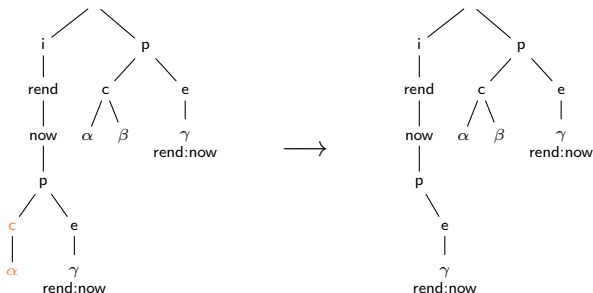
## Periodic Garbage Collection (GC)

# Periodic GC

Main idea:

- Background process
- Periodically traverse the whole index subtree
- Prune any visited unproductive node

# Periodic GC



volatile

## Query-Time Pruning (QTP)

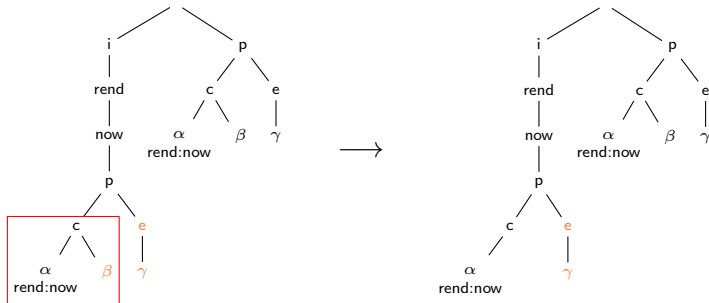
# Query-Time Pruning

Main idea:

- Prune unproductive nodes during query execution
- Piggybacking on query execution
- Adds overhead on query runtime
- Avoids unnecessary full index traversals
- We traverse a part of the index subtree and prune any visited unproductive node



# Query-Time Pruning



$Q(\text{render}, \text{now}, \text{/products/consumer})$

unproductive

# Experimental Evaluation

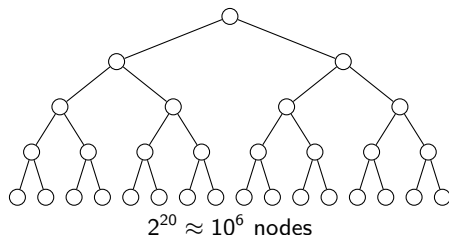
Experiments are run on the hierarchical database system  
Apache Jackrabbit Oak



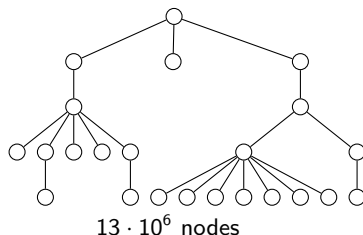
# Datasets

- Datasets resemble the content subtree

(a) Synthetic



(b) Dell

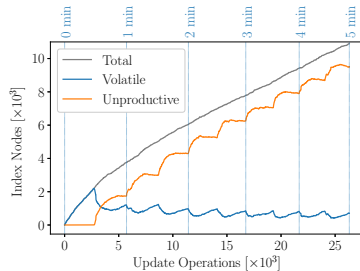
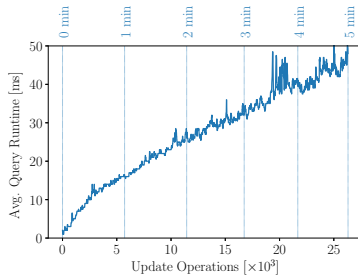


# Workload simulation

- Zipf distribution
- Workload changes every 30 seconds
- 10 update operations per query operation
- Update operation: add or remove a property triggering index updates

## Impact of Unproductive Nodes on Query Runtime

# Impact of Unproductive Nodes on Query Runtime

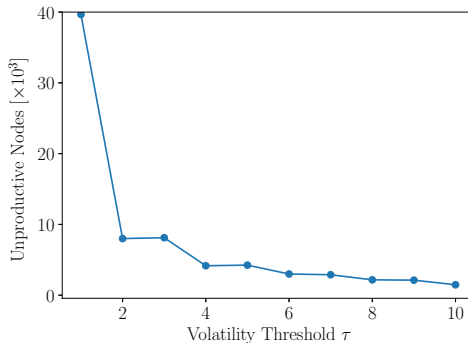


- Query runtime increases by an order of magnitude after five minutes

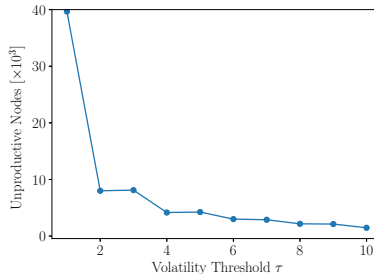
Volatility threshold  $\tau$



# Volatility threshold $\tau$



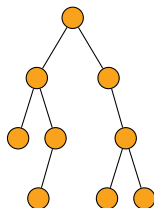
# Volatility threshold $\tau$



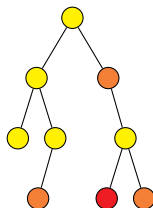
- $\tau \nearrow \implies$  volatile nodes  $\searrow$
- volatile nodes  $\searrow \implies$  unproductive nodes  $\searrow$
- Power law relationship between #unproductive nodes and  $\tau$

Workload skew s

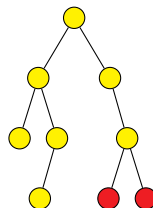
# Workload skew $s$



No skew  
 $s = 0$

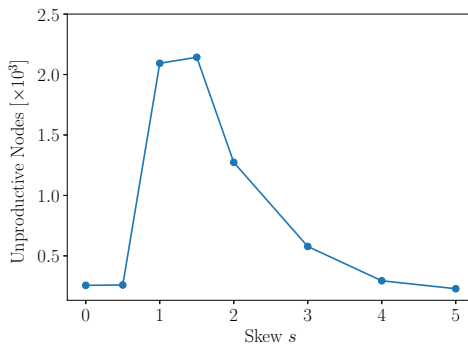


Normal skew  
 $s = 1$

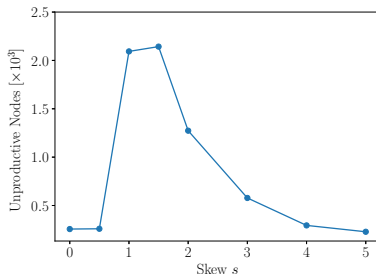


High skew  
 $s = 2$

# Workload skew $s$

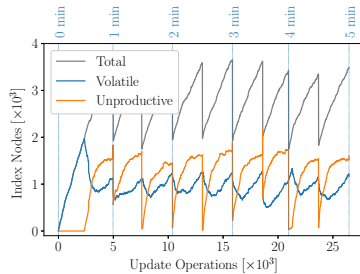
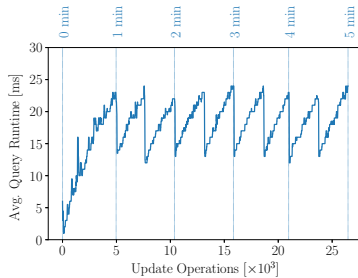


## Workload skew $s$



- $s > 1 \implies$  small hotspot  $\implies$  few unproductive nodes
- $s < 1$  (uniform)  $\implies$  no hotspot  $\implies$  few unproductive nodes

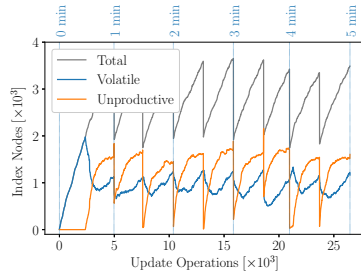
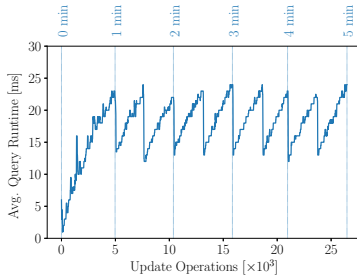
## Cleaning Unproductive Nodes



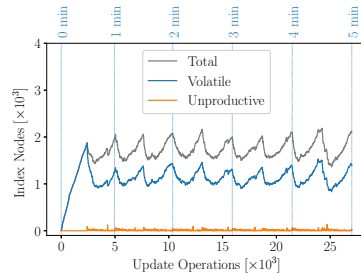
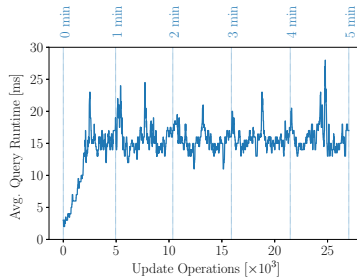
We run GC every 30 seconds



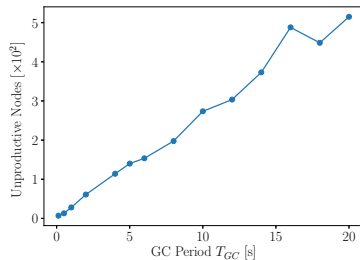
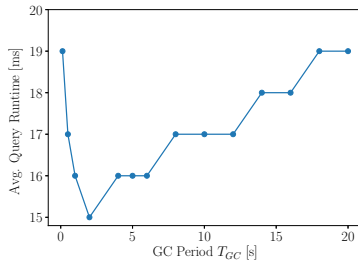
GC



QTP



# GC period $T_{GC}$



- Optimal GC period  $T_{GC}^*$ : period with the smallest query runtime
- Too small  $T_{GC} \implies$  GC steals resources from query executor

# QTP

## Algorithm: QueryQTP

**Data:** Query  $Q(k, v, m)$ , where  $k$  is a property,  $v$  a value and  $m (= / \lambda_1 / \dots / \lambda_d)$  a content node's path.

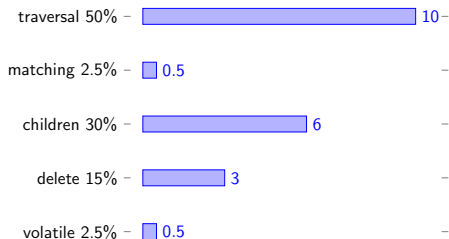
**Result:** A set of nodes satisfying  $Q(k, v, m)$

$r \leftarrow \emptyset$

```
for node  $n \in \text{desc}(/i/k/v/\lambda_1/\dots/\lambda_d)$  in
  postorder tree walk do
    if  $\text{matching}(n)$  then
      |  $r \leftarrow r \cup \{n\}$ 
    else if  $\text{children}(n) = \emptyset \wedge \neg \text{volatile}(n)$  then
      | delete node  $n$ 
```

return  $r$

## runtime [ms]



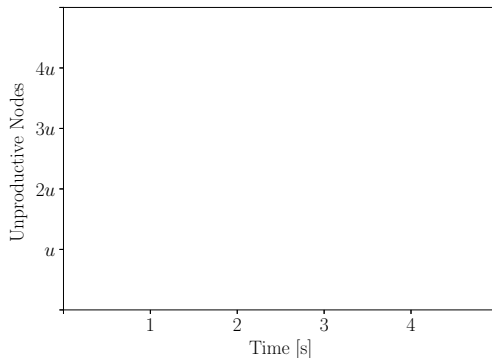
## Periodic GC vs. QTP

Simple model:

- assume constant rate of growth of unproductive nodes

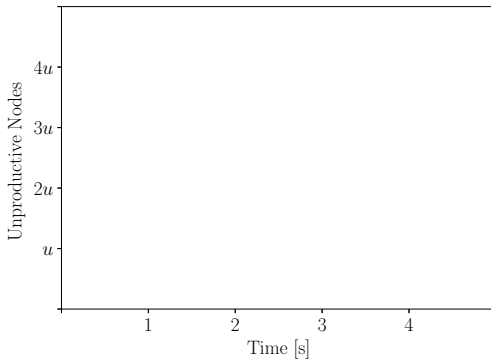
## Simple Model

- production rate of unproductive nodes  
 $r = \frac{u}{s}$  ( $u$  unproductive nodes per second)



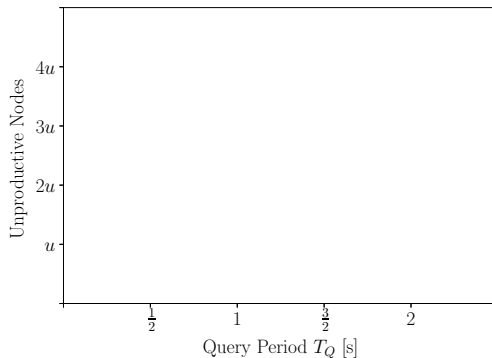
## Simple Model, GC

- production rate  $r = \frac{u}{s}$
- GC period  $T_{GC} = 2s$



## Simple Model, GC vs. QTP

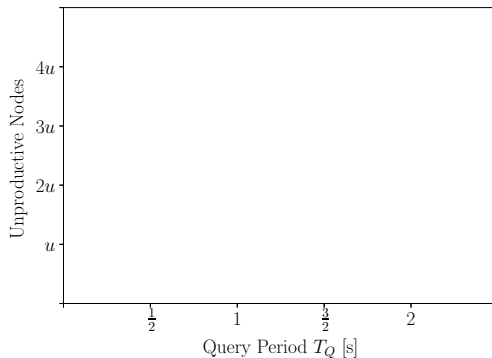
- production rate  $r = \frac{u}{s}$
- GC period  $T_{GC} = 2s$



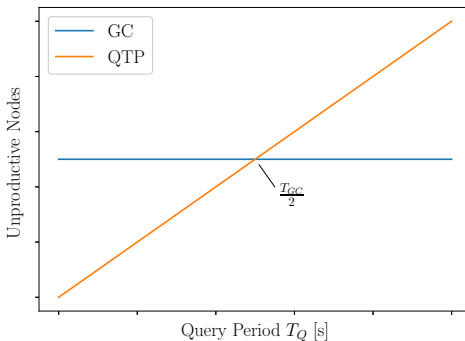


# Simple Model, GC vs. QTP

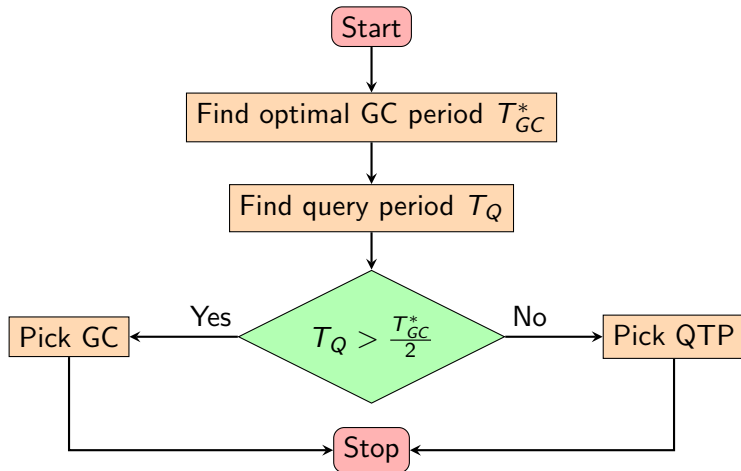
- production rate  $r = \frac{u}{s}, r' = \frac{2u}{s}$
- GC period  $T_{GC} = 2s$



# Simple Model, GC vs. QTP



Queries traverse on average fewer unproductive nodes under QTP when the query period  $T_Q$  is less than half the GC period  $T_{GC}$



# Conclusion

# Unproductive Nodes

- volatility threshold  $\tau \nearrow \implies$  unproductive nodes  $\searrow$
- sliding window length  $L \nearrow \implies$  unproductive nodes  $\nearrow$
- workload skew  $s \updownarrow \implies$  unproductive nodes  $\searrow$
- update operations per second  $\nearrow \implies$  unproductive nodes  $\nearrow$

# GC & QTP

## GC

- Periodically cleans unproductive index nodes
- Sawtooth pattern
- Slows down system if run too often

## QTP

- Faster and more stable than GC when queries are frequent
- Adds overhead to queries
- Overhead negligible in the long-term

# Future Work

# Future Work

- Concurrency control
- Frequently changing query filter
- Unproductive node production rate