An Adaptive Index for Hierarchical Database Systems

Rafael Kallis

BSc Thesis

February 12, 2018

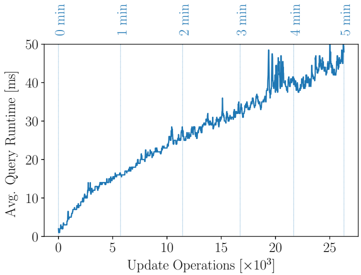
University of Zurich[UZH]
Department of Informatics

The Workload-Aware Property Index (WAPI):

- Detects volatile, i.e. frequently updated, nodes
- Stops pruning such 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

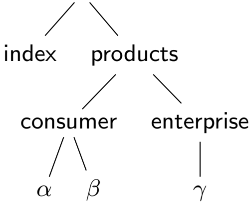


In this thesis we:

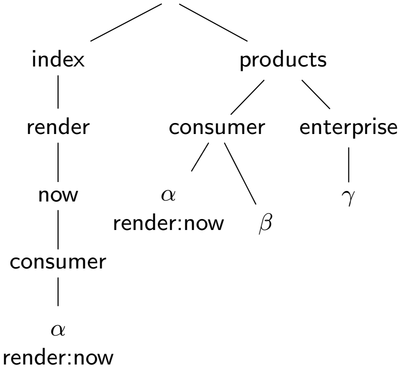
- Design and implement two solutions in order to mitigate unproductive nodes
- Analyze factors impacting the production of unproductive nodes
- Empirically evaluate and compare our two solutions

Background

Hierarchical database for an e-commerce platform

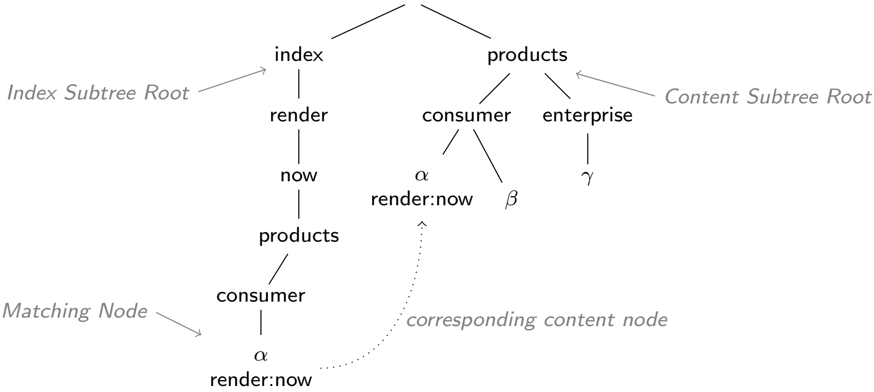


HTML pre-rendering



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.,

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

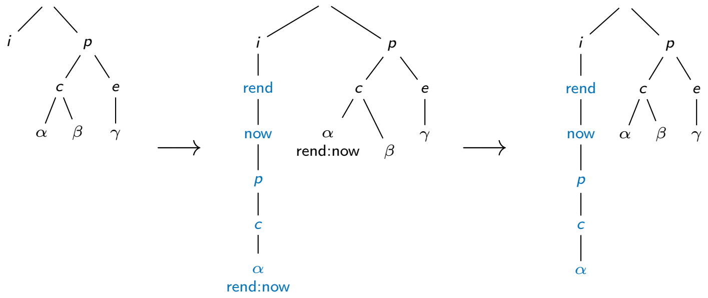
Definition (Volatile Node)

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

$$volatile(n) \iff vol(n) \geq \tau$$

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

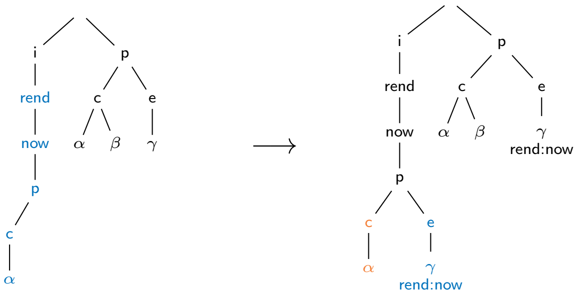
Index nodes becoming volatile



volatile

Unproductive Nodes

Index nodes becoming unproductive

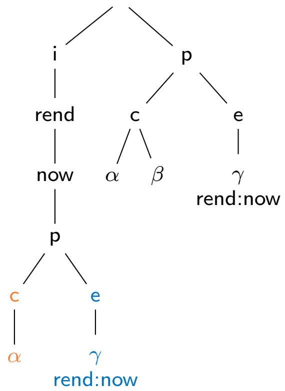


volatile
unproductive

Definition (Unproductive Node)

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

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



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

volatile
unproductive

The number of unproductive nodes depends on:

- Volatility threshold τ
- 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)

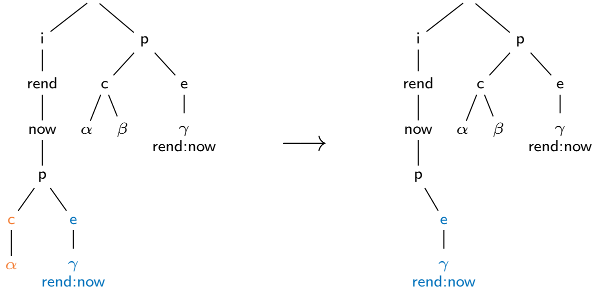
Periodic Garbage Collection (GC)

Periodic GC

Main idea:

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

Periodic GC



volatile
unproductive

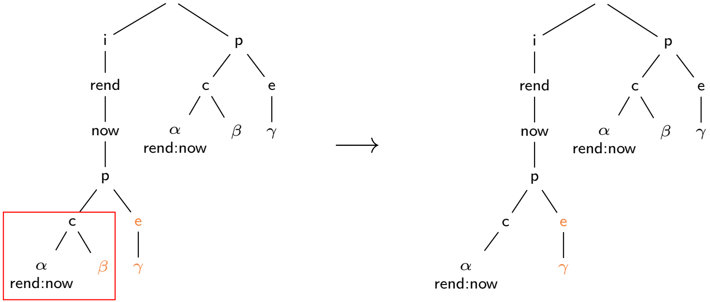
Query-Time Pruning (QTP)

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



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

unproductive

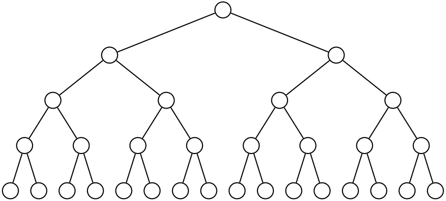
Experimental Evaluation

Experiments are run on the hierarchical database system Apache Jackrabbit Oak

Datasets

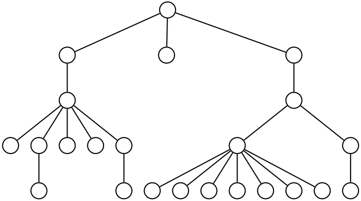
- Datasets resemble the content subtree

(a) Synthetic



$2^{20} \approx 10^6$ nodes

(b) Dell



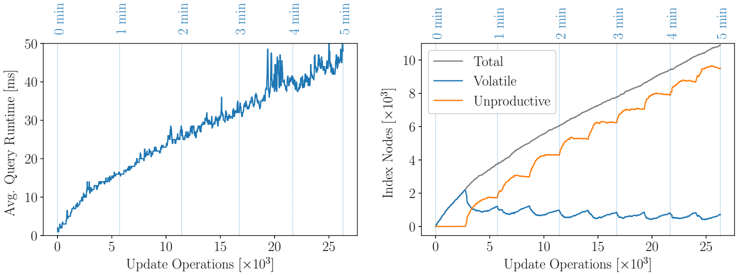
$13 \cdot 10^6$ nodes

Workload simulation

- Zipf distribution
- Workload changes every 30 seconds
- 10 update operations per query operation
- Update operation: add and remove a property which triggers an index update

Impact of Unproductive Nodes on Query Runtime

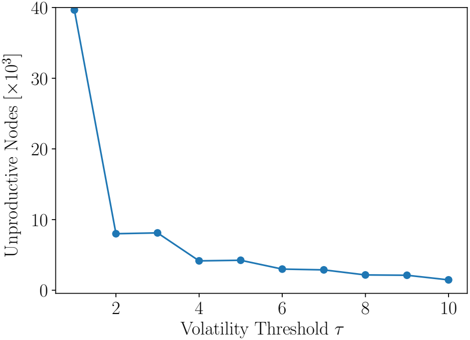
Impact of Unproductive Nodes on Query Runtime



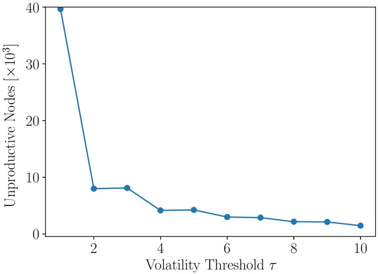
- Query runtime increases by an order of magnitude after five minutes

Volatility threshold τ

Volatility threshold τ



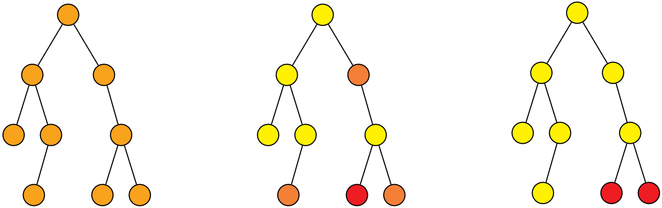
Volatility threshold τ



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

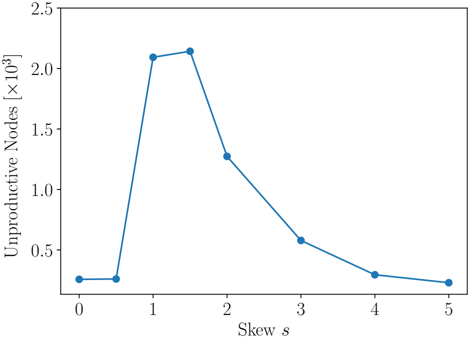
Workload skew s

Workload skew s

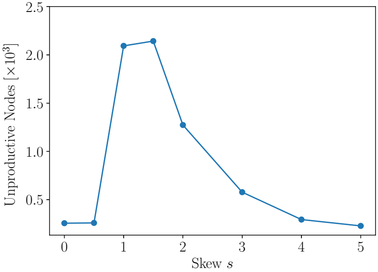


No skew Normal skew High skew

Workload skew s



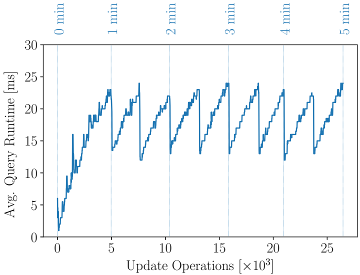
Workload skew s

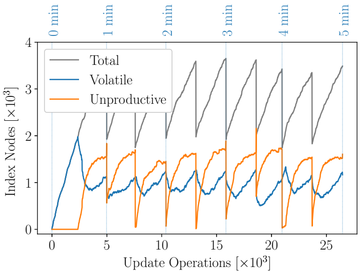


- $s > 1 \implies$ small hotspot $\implies$ few unproductive nodes
- $s < 1$ (uniform) $\implies$ no hotspot $\implies$ no unproductive nodes

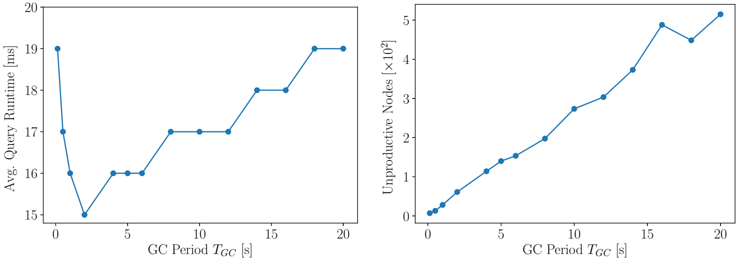
Periodic Garbage Collection

Periodic GC


0 min
1 min
2 min
3 min
4 min
5 min
Avg. Query Runtime [ms]
Update Operations [×10^3]


0 min
1 min
2 min
3 min
4 min
5 min
Total
Volatile
Unproductive
Index Nodes [×10^3]
Update Operations [×10^3]

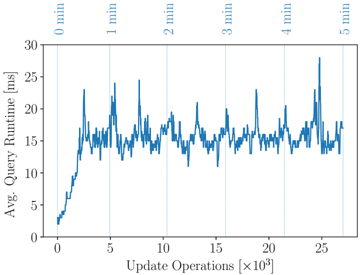
GC period T

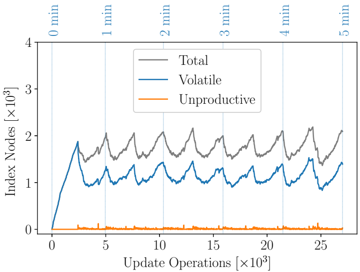


- Optimal GC period T^*_{GC}: period with the smallest query runtime
- Too small $T_{GC} \implies$ GC steals resources from query executor

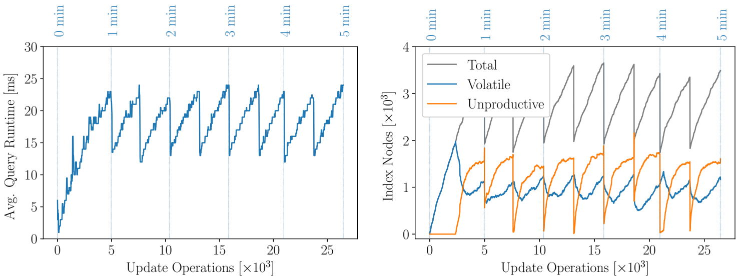
Query-Time Pruning

QTP

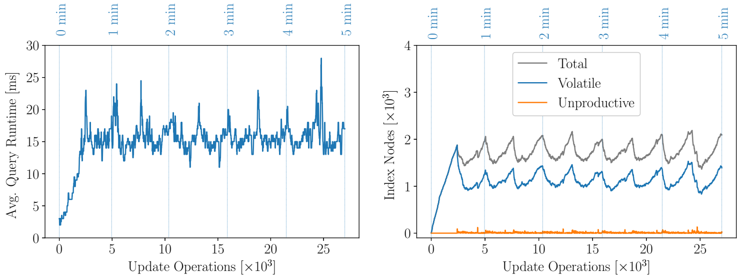

0 min
1 min
2 min
3 min
4 min
5 min
Avg. Query Runtime [ms]
Update Operations [$\times 10^3$]


0 min
1 min
2 min
3 min
4 min
5 min
Total
Volatile
Unproductive
Index Nodes [$\times 10^3$]
Update Operations [$\times 10^3$]

GC


0 min
1 min
2 min
3 min
4 min
5 min
Avg. Query Runtime [ms]
Update Operations [×10³]
Total
Volatile
Unproductive
Index Nodes [×10³]

QTP


0 min
1 min
2 min
3 min
4 min
5 min
Avg. Query Runtime [ms]
Update Operations [×10³]
Total
Volatile
Unproductive
Index Nodes [×10³]

QTP

Algorithm: QueryQTP

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

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

$r \longleftarrow \emptyset$

for *node* $n \in desc(/i/k/v/\lambda_1/\ldots/\lambda_d)$ *in postorder tree walk* **do**

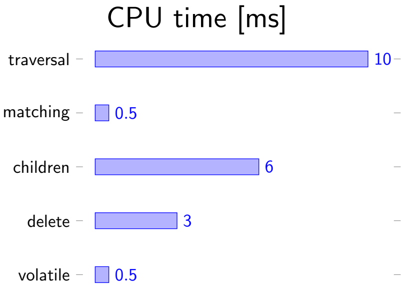
  if *matching*(n) **then**

    $r \longleftarrow r \cup \{*n\}$

  else if *children*$(n) = \emptyset \wedge \neg$*volatile*$(n)$ **then**

    delete node n

return r



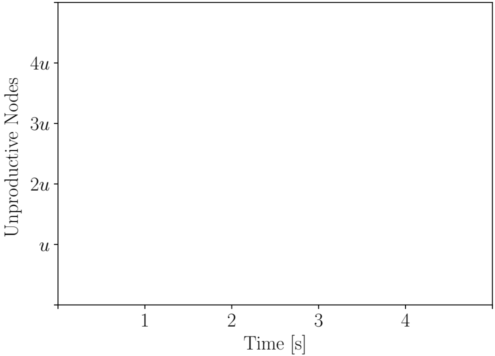
Periodic GC vs. QTP

Simple model:

- assume constant rate of growth of unproductive nodes

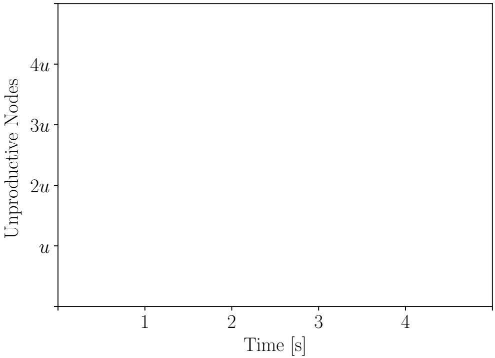
Simple Model

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



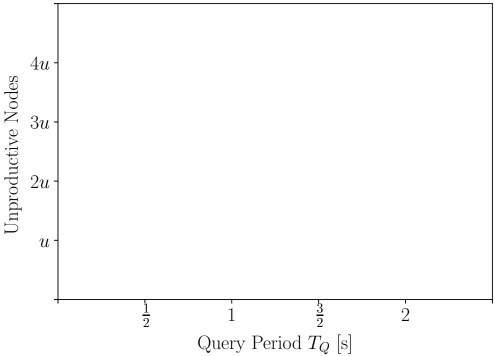
Simple Model, GC

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



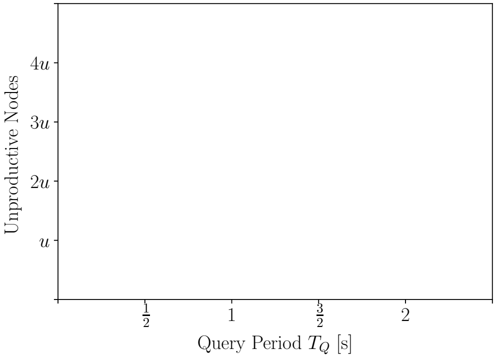
Simple Model, GC vs. QTP

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

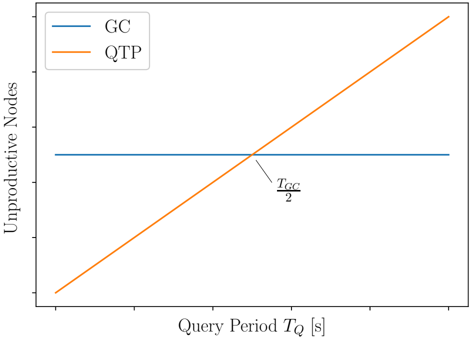


Simple Model, GC vs. QTP

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



Simple Model, GC vs. QTP



Queries traverse on average the same number of unproductive nodes under GC or QTP when the query period T_Q is 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 $s \uparrow\downarrow \implies$ unproductive nodes $\searrow$
- update operations per second $\nearrow \implies$ unproductive nodes $\nearrow$

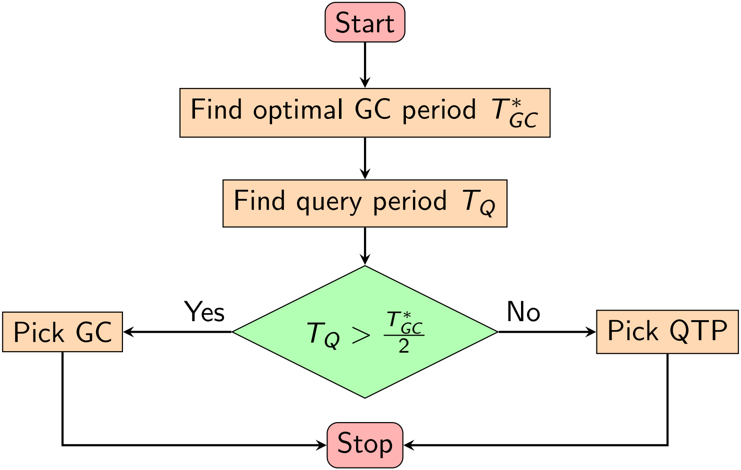
GC & QTP

GC

- Periodically cleans 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


Start
Find optimal GC period T^*_{GC}
Find query period T_Q
$T_Q > \frac{T^*_{GC}}{2}$
Yes
No
Pick GC
Pick QTP
Stop

Future Work

Future Work

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