# Period Index: A Learned 2D Hash Index for Range and Duration Queries


Andreas Behrend[1] Anton Dignös[2] Johann Gamper[2] Philip Schmiegelt[3] Hannes Voigt[4] Matthias Rottmann[5] Karsten Kahl[5]

[1]University of Bonn, Germany

[2]Free University of Bozen-Bolzano, Italy

[3]Fraunhofer FKIE Bonn, Germany

[4]TU Dresden, Germany

[5]University of Wuppertal, Germany



SSTD' 19, Vienna, Austria

# Background and Motivation /1

Temporal period data is produced in many application domains:

- Personnel data (work contract periods, project assignment periods)
- Financial data (insurance policy periods, rental contract periods)
- Medical data (hospital stay periods)

...or derived via state analysis:

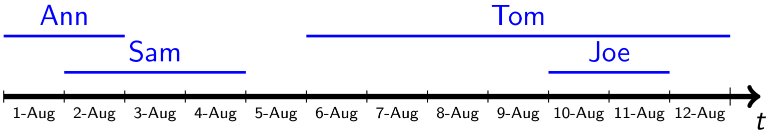
- Medical data (treatment periods, high fever periods)
- Air traffic data (maneuvering periods, landing periods)

# Background and Motivation /2

- Example: Fever periods of patients

| **Patient** | **Period** |
|---|---|
| Ann | [01-Aug-2019, 03-Aug-2019) |
| Sam | [02-Aug-2019, 05-Aug-2019) |
| Tom | [06-Aug-2019, 13-Aug-2019) |
| Joe | [10-Aug-2019, 12-Aug-2019) |

- Graphical illustration



# Background and Motivation /3

- Heavy research efforts since decades
  - Temporal data storage
  - Temporal keys
  - Temporal indexing
  - Time travel queries
  - Temporal aggregation queries
  - Temporal join queries

- Some works found their way into commercial DBMSs

# Background and Motivation /4

- Temporal features in the SQL:2011 standard [7]
  - Period specification, application time and system time
  - Predicates like overlap, before, during
  - Implementation in IBM D2, Oracle, Teradata, SQLServer [4]

- Range types in PostgreSQL (2012) [9]
  - Range datatype, predicates, and functions
  - Advanced temporal query processing prototype [5]

# Background and Motivation /5

- An important primitive is time travel or range queries
  - Find all active insurance policies **as of yesterday**
  - Find all air planes maneuvering **yesterday between 08:00 and 10:00**

- An often neglected attribute is the duration of periods
  - Find all active insurance policies **longer than a year**
  - Find all air planes maneuvering for **5 to 6 hours**

Position **and** duration are key properties for querying data with intervals!

# Three Types of Queries

## Range Query

Find all patients that had fever **last week**

## Duration Query

Find all patients that had fever **for more than two days**

## Range-Duration Query

Find all patients that had fever **last week for more than two days**

# Challenges

## Range Predicate

- Corresponds to the overlap of two intervals
- Inequality among both endpoints (difficult to index and optimize)
- `overlap(I, [start, end)) = Ie > start AND Is < end`

## Duration Predicate

- Rarely interested in a precise duration, i.e., 2h 5min 32 sec
- Needs to be expressible as a range, i.e., duration between 2h and 4h
- `duration(I) > min AND duration(I) < max`

We want to use **both** at the same time!

Background and Motivation

Contributions

Period Index

Experiments

Conclusion and Future Work

# Contributions

- *Period Index*: novel index for intervals according to **duration and position** on the timeline
- Grid-based data structure with **constant time lookup** – well suited for parallelization
- Space-efficient implementation using arrays and linked lists to **avoid storage of empty cells**
- Adaptive bucket length for different **data distribution** (learn)

# Related Work

## Range Queries

- Interval tree [3, 6]
- Segment tree [2]

## Duration Queries

- Any sorted index, most notably B-trees

## Range-Duration Queries

- Multidimensional indices, most notably quad-trees, R-trees [1] and Grid file [8]

# Intuition of the Period Index

## Idea

- Index position of intervals using **buckets** and cells
- Index duration of intervals using **levels**

## Observations / Assumptions

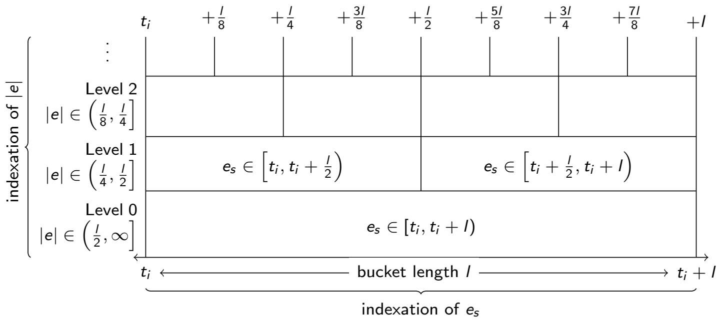
- Position of intervals may be arbitrarily skewed
- Duration of intervals follows ZIPF distribution
  (many short / few long)

## Desiderata

- Access to **relevant** cells should be **fast**
- Cells should be **equally filled**

# Data Structure

- Buckets of fixed length $l$ to index position (horizontal)
- Levels with smaller cells to index duration (vertical)




- Intervals are stored in all overlapping cells on the corresponding level
- Relevant cells are calculated arithmetically

# Construction / Interval Assignment

Given an interval $e = [e_s, e_e)$ with duration $|e|$.

1. Find corresponding level:

$$x = min\Big(max\big(\lfloor \log_2(\frac{l}{|e|}) \rfloor, 0\big), n_l\Big)$$

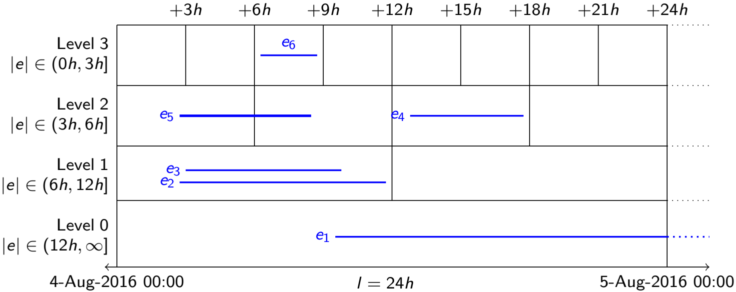
2. Place $e$ into cells $c_i$ s.t.:

$$\left\lfloor \frac{e_s - o}{l} \cdot 2^x \right\rfloor \leq i \leq \left\lfloor \frac{e_e - o}{l} \cdot 2^x \right\rfloor$$

Relevant cells for interval $e$ are calculated **arithmetically**
$f(e, d) \rightarrow \{c_i, c_j, \ldots\}$

# Example / Interval Assignment

- Buckets length $l = 24\text{h}$ and number of levels $n_l = 3$
- $e_5 = [02{:}45, 08{:}30)$ with $|e_5| = 5h\ 45m$




- $e_5 \rightarrow$ level $2$, cells $\{c_1$ and $c_2\}$

# Query Evaluation

Given a Range-Duration query $Q$ with range $e = [e_s, e_e)$ and duration restriction $d = [d_{min}, d_{max})$.

1. For each level $x$ s.t.:

$$min\Big(max\big(\lfloor \log_2(\frac{l}{d_{min}})\rfloor, 0\big), n_l\Big) \leq x \leq min\Big(max\big(\lfloor \log_2(\frac{l}{d_{max}})\rfloor, 0\big), n_l\Big)$$

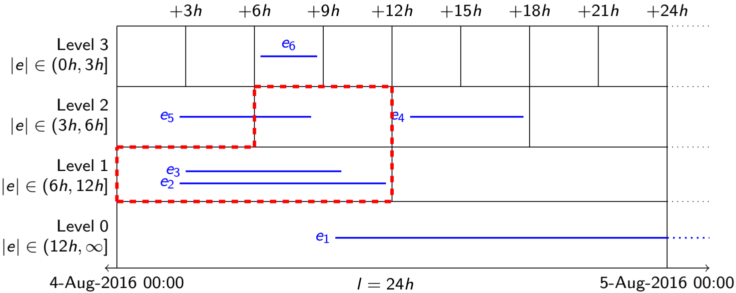
2. Scan all cells $c_i$ s.t.:

$$\left\lfloor \frac{e_s - o}{l} \cdot 2^x \right\rfloor \leq i \leq \left\lfloor \frac{e_e - o}{l} \cdot 2^x \right\rfloor$$

Relevant cells for query $Q$ are calculated **arithmetically** $f(e, d) \to \{\dots, c_i, \dots\}$

# Example / Query Evaluation

$Q(e, d)$: $e = [08{:}00, 10{:}00)$ and $d = [5\text{h}, 10\text{h})$




- $Q \rightarrow \{(\text{level } 1, \text{cells } c_1), (\text{level } 2, \text{cells } c_2)\}$

# Analysis

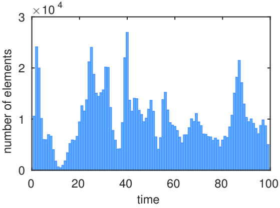
- The number of cells proportionally determines the number of collisions in the index; Large dataset $\rightarrow$ many cells
- The maximum duration of intervals limits the maximum bucket length $l$
- The minimum duration of intervals limits the maximum number of levels $n_l$
- Given $l$ and the number of levels $n_l$ we can control the total number of cells.
  $\#cells = (2 \cdot 2^{n_l} - 1) \cdot \frac{|D|}{l}$ adjustable in the range of 1 to $2 \cdot |D|$[1]

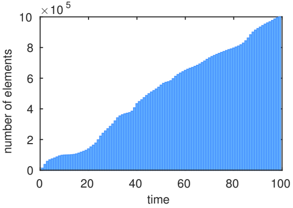
[1] $|D|$ is the domain size

# Adaptive Bucket Length

- What if start times are not uniformly distributed?

- Use Histogram of starting points to model the distribution
- Replace regular time division with weight from cumulative histogram



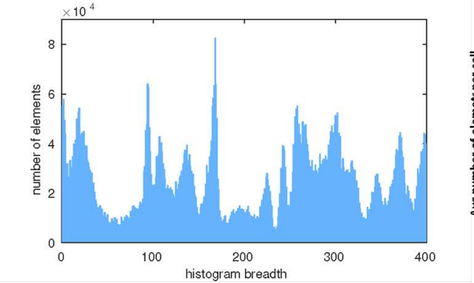

- Relevant Cells are calculated **arithmetically**
  $f(e, d) + H \rightarrow \{c_i, c_j, \ldots\}$

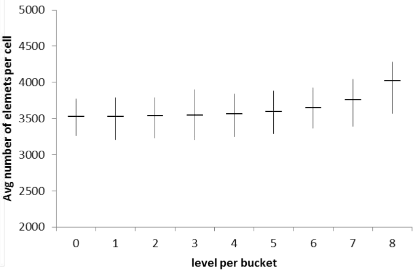
# Experiments - Setup

- Adaptive bucket lengths
- Competitors
  - Grid file
  - Interval tree (Range Queries only)
  - B+-tree (Duration Queries only)
- Runtime in Query/sec
  - Range Queries only
  - Duration Queries only
  - Range-Duration Queries
  - Different duration distributions

# Experiments - Adaptive bucket lengths

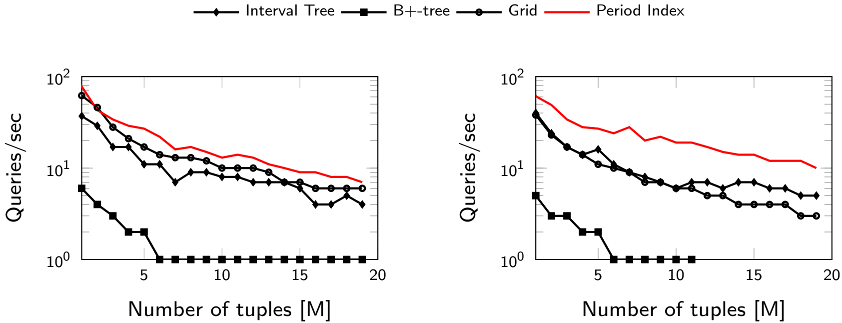
- Impact of start time point distribution on cell fill factor




- Adaptive bucket length is able to counteract (learn) data skew and distribute load among cells

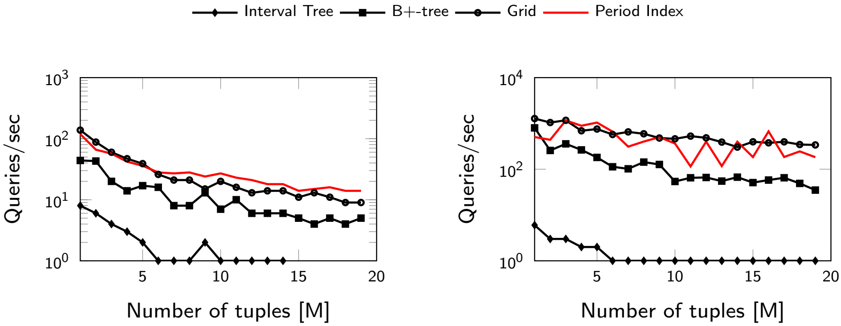
# Experiments - Range Query




Uniform distribution Zipf distribution

- Period index applies efficient calculation of relevant cells (compared to trees)
- Period index has smaller cells for small intervals (compared to Grid)

# Experiments - Duration Query

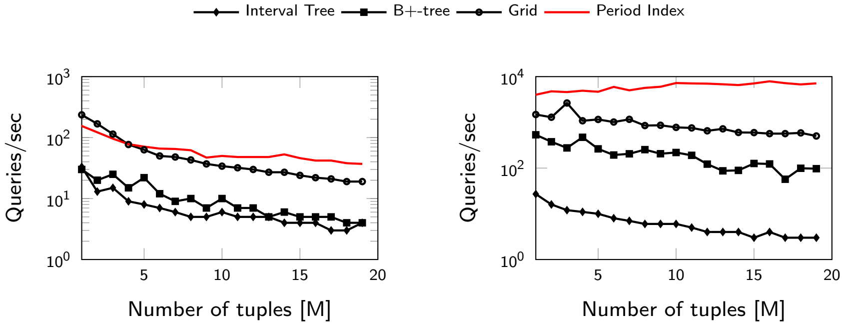



Uniform distribution

Zipf distribution

- Period index applies efficient calculation of relevant cells (compared to trees)
- Regular grid may in some cases have less relevant cell accesses

# Experiments - Range-Duration Query




Uniform distribution

Zipf distribution

- Period index applies both range and duration restriction (compared to trees)
- Period index has smaller cells for small intervals (compared to Grid)

# References


Norbert Beckmann et al. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: *SIGMOD Conference*. ACM Press, 1990, pp. 322–331.

Jon Louis Bentley. "Solutions to Klee's rectangle problems". In: *Technical report* (1977).

Mark de Berg et al. "Interval Trees". In: *Computational Geometry*. Springer-Verlag, 2000. Chap. 10.1, pp. 212–217.

Michael H. Böhlen et al. "Temporal Data Management - An Overview". In: *eBISS*. Vol. 324. Lecture Notes in Business Information Processing. Springer, 2018, pp. 51–83.

Anton Dignös et al. "Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries". In: *ACM Trans. Database Syst.* 41.4 (2016), 26:1–26:46.

Hans-Peter Kriegel, Marco Pötke, and Thomas Seidl. "Managing Intervals Efficiently in Object-Relational Databases". In: *VLDB*. Morgan Kaufmann, 2000, pp. 407–418.

Krishna G. Kulkarni and Jan-Eike Michels. "Temporal features in SQL: 2011". In: *SIGMOD Record* 41.3 (2012), pp. 34–43.

Jürg Nievergelt, Hans Hinterberger, and Kenneth C. Sevcik. "The Grid File: An Adaptable, Symmetric Multikey File Structure". In: *ACM Trans. Database Syst.* 9.1 (1984), pp. 38–71.

PostgreSQL. *Documentation manual PostgreSQL - range types.* `https://www.postgresql.org/docs/9.2/rangetypes.html`. 2012.

# Conclusion and Future Work

- Period index: efficient access method for period-duration queries
- Constant time access to relevant cells
- Adaptive to different start time distributions
- Well suited for parallelization

Future Work

- Extend to more fine grained duration distribution among cells
- Extensive experiments on parallelization and NUMA
- Adaptability to dynamic scenarios similar to extendible hashing

Thank you for your attention!