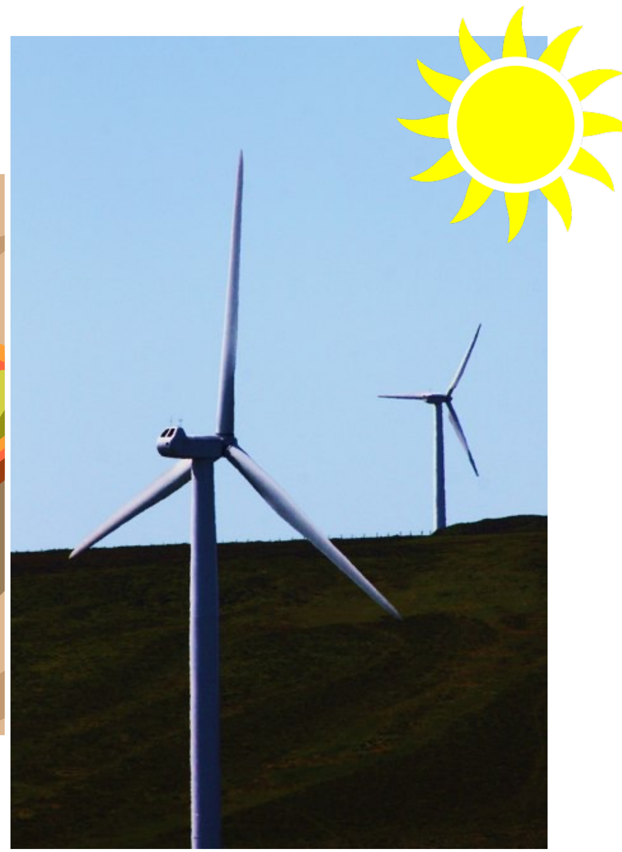
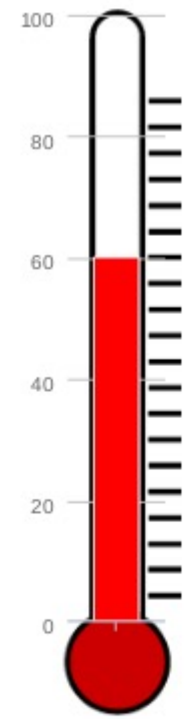
"Effektive metoder til at gemme og forespørge på store mængder tidsseriedata"

Christian Thomsen (chr@cs.aau.dk)

Joint work with Søren Kejser Jensen and Torben Bach Pedersen

Example – summer





Example – autumn

The challenge

- Wind turbines and solar panels have a lot of sensors that can deliver data values several times per second
- A modern wind turbine has up to 6,500 streams

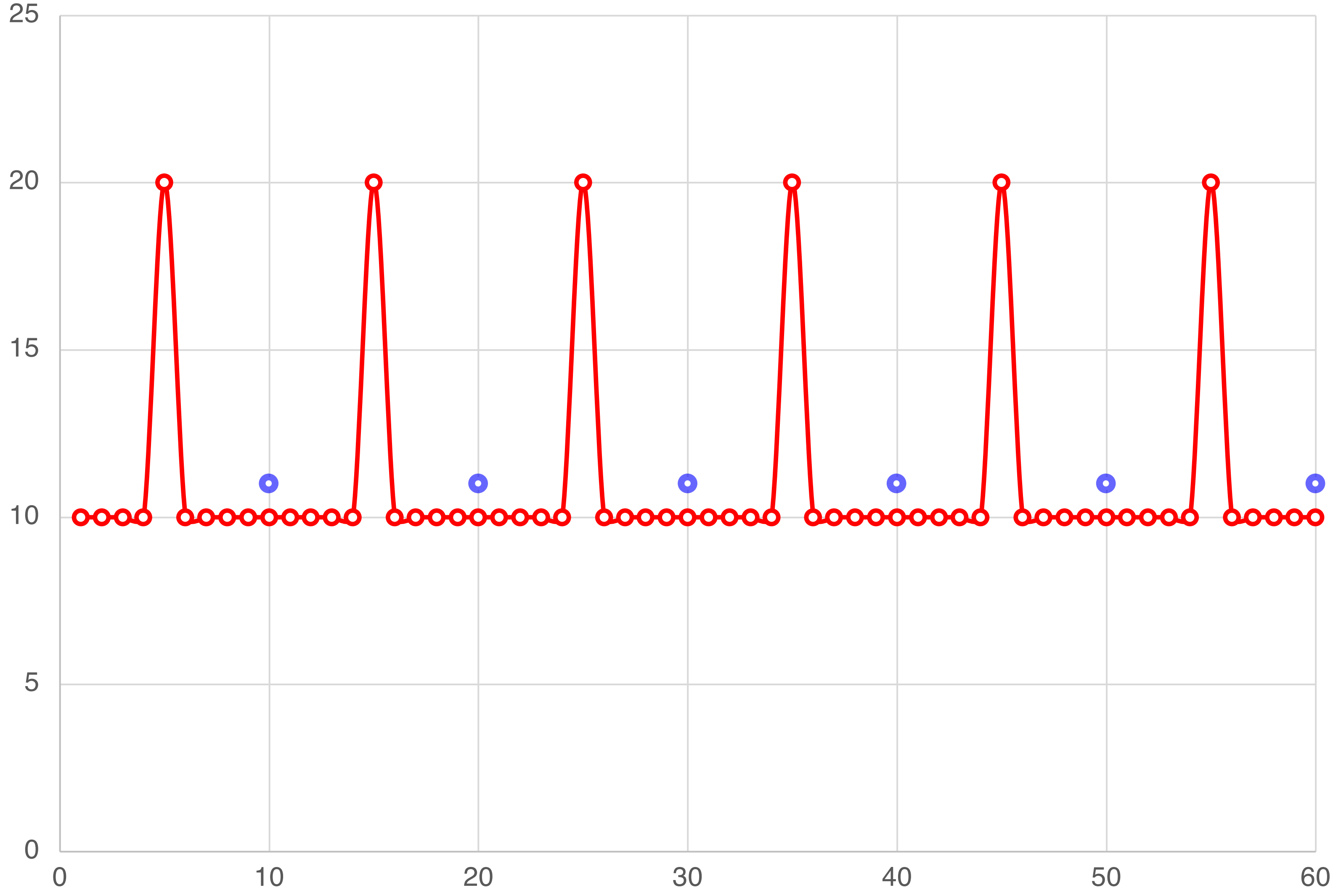
- This generates a lot of data
  - 10 reads/second, 4 bytes, 6,500 streams ➔ ~20 GiB per day from one wind turbine

  - In practice, some sensors are sampled less often today, but a wind turbine still produces around 1GiB data per day
  - The sampling frequencies and amounts of data to store are increasing

The current situation

- The available information is currently not exploited or stored
- Many monitoring solutions consider few (~100) sensor streams and store only a single value for every *x* minutes (e.g., the average)
  - *x* is typically ½, 1, 2, 5, or 10
- Important things might not be seen since outliers and fluctuations can be lost

Example of ”missing the point” :-)



What we want to do…

- Store and use all available sensor data
- Support efficient aggregate queries on historical data
- Support analysis of data while it is being ingested
- Detect underperformance and other problems immediately
- Enable predictive maintenance

Why is that good? $$$!

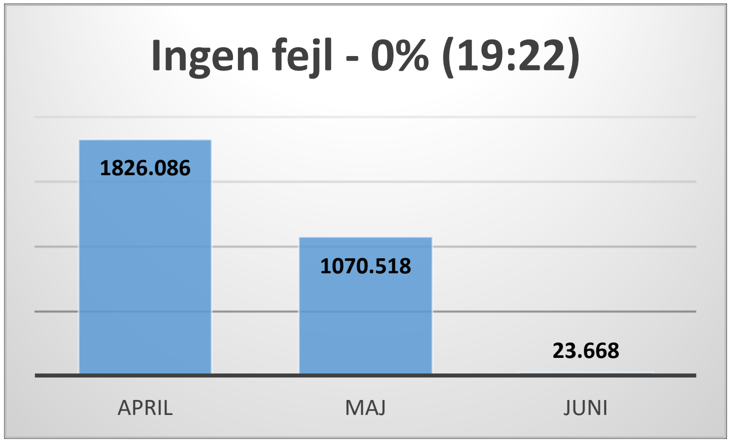
- For example, detect and fix a problem before the wind turbine breaks
- Reduced costs for service and spare parts
  - No over-time hours, crane booked in advance
- Service when there is little wind anyway
- Less downtime → more production
  - Delivery of a gearbox or wing can take months
- The service cost represents 11-30% of the onshore wind energy cost
- Global wind service revenue: 8 billion USD

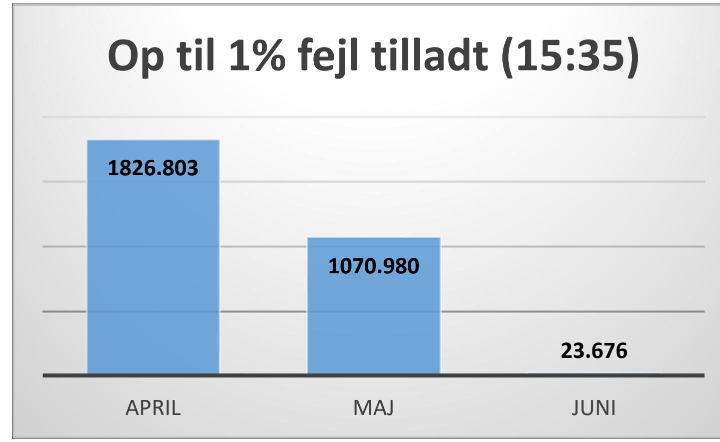
How we do it

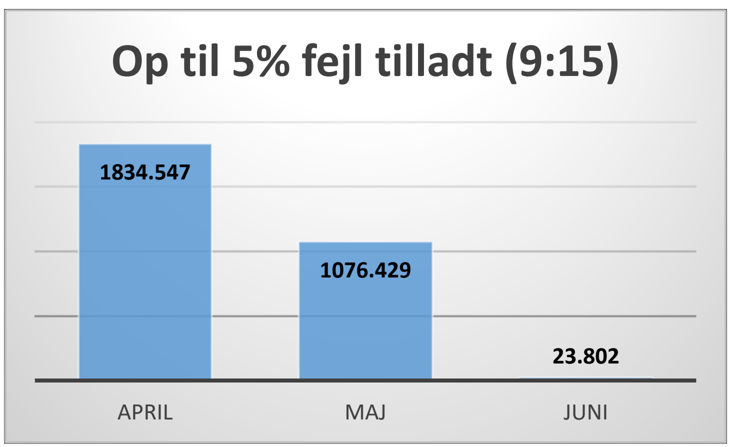
- Time-series can contain millions of points
- An efficient way to store and process them is to represent them by ***models***
- We use a ***model-based*** approach for the time-series data
- A (user-defined) error-bound can be set
  - For example 5%, 1%, or even 0%

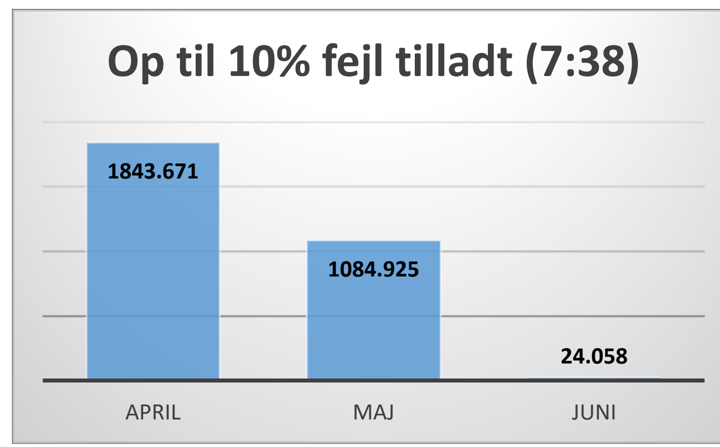
- Allowing an error in the representation can lead to better compression and performance

Ryd

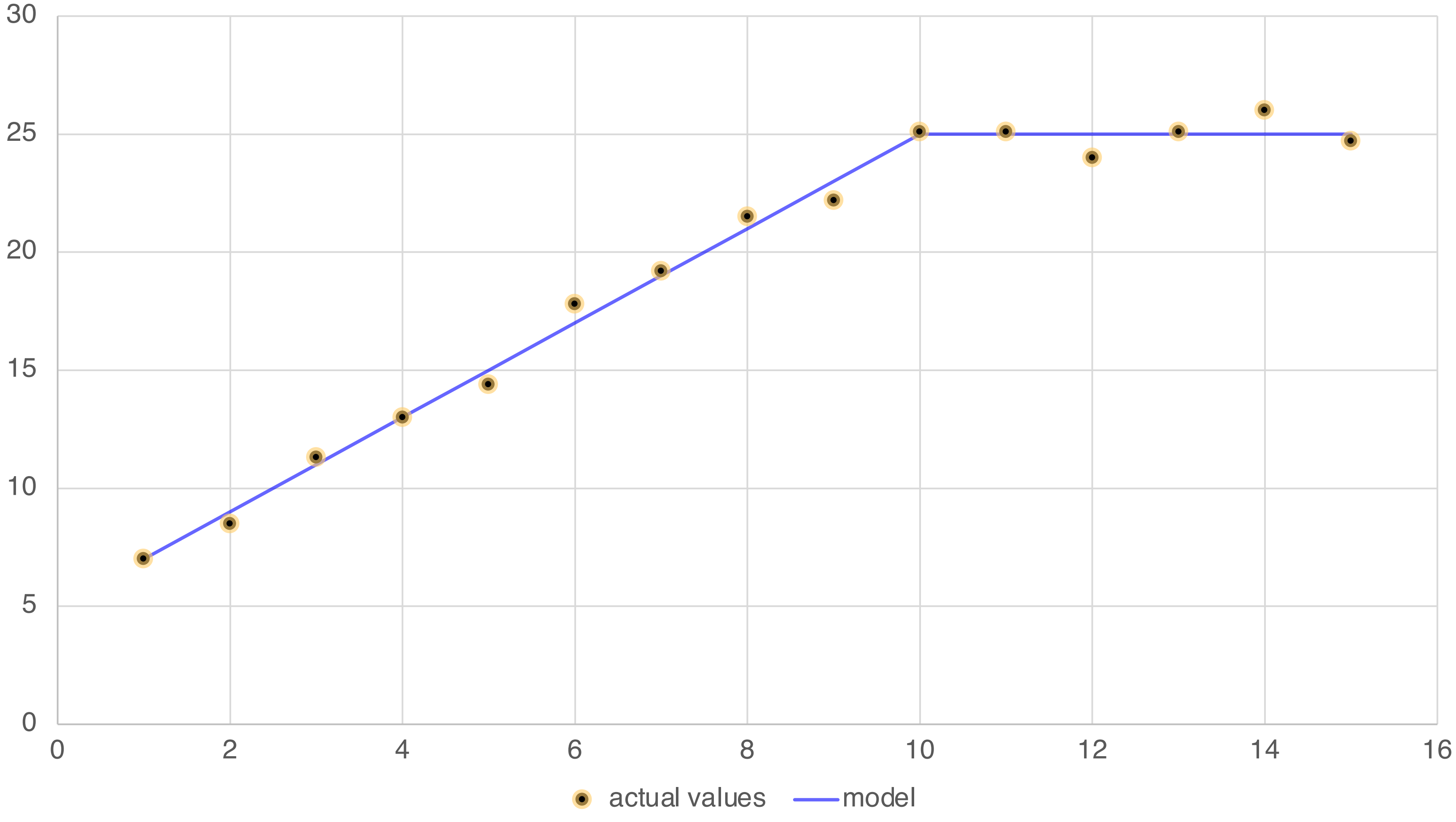

Ingen fejl - 0% (19:22)
1826.086
1070.518
23.668
APRIL
MAJ
JUNI


Op til 1% fejl tilladt (15:35)
1826.803
1070.980
23.676
APRIL
MAJ
JUNI


Op til 5% fejl tilladt (9:15)
1834.547
1076.429
23.802
APRIL
MAJ
JUNI


Op til 10% fejl tilladt (7:38)
1843.671
1084.925
24.058
APRIL
MAJ
JUNI

Simple example of models



ModelarDB

- We have developed the time series management system ***ModelarDB*** which uses models to store time series data
- Time series-specific functionality implemented in a system-agnostic library

- We have implemented some model types and the user can *optionally* add more
- ModelarDB adapts to the dataset and automatically picks the best model type to use for a given part of a time series
- Query processing and storage from existing systems
  - Apache Spark and Cassandra, respectively
  - Can be replaced by others

Storage requirements for a real-world data

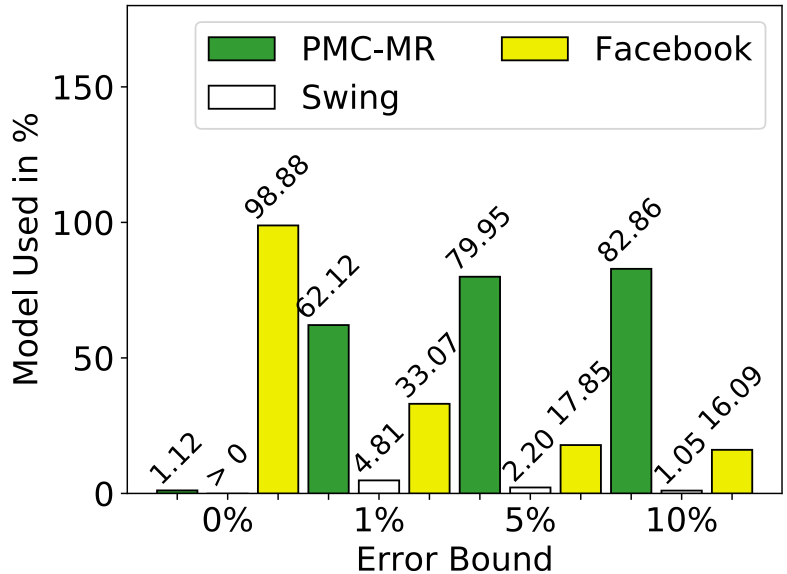
Storage Method	Size in GiB
CSV files	582.68
PostgreSQL 10.1	782.87
RDBMS-X (row)	367.89
RDBMS-X (column)	166.83
InfluxDB 1.4.2	4.33 – 4.44
Apache Parquet files	106.94
Apache ORC files	13.50
Apache Cassandra 3.9	111.89
ModelarDB	2.41 – 2.84

When the error bound is 10%, the actual average error is only 0.005% here!

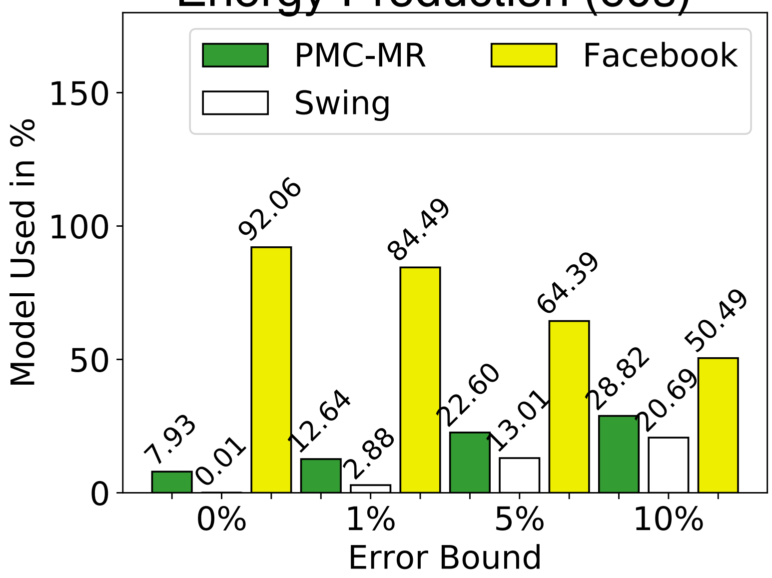
Models used for different data sets

High-frequent production data (.1s)

Extended REDD data (1s)



Energy Production (60s)



Evaluation

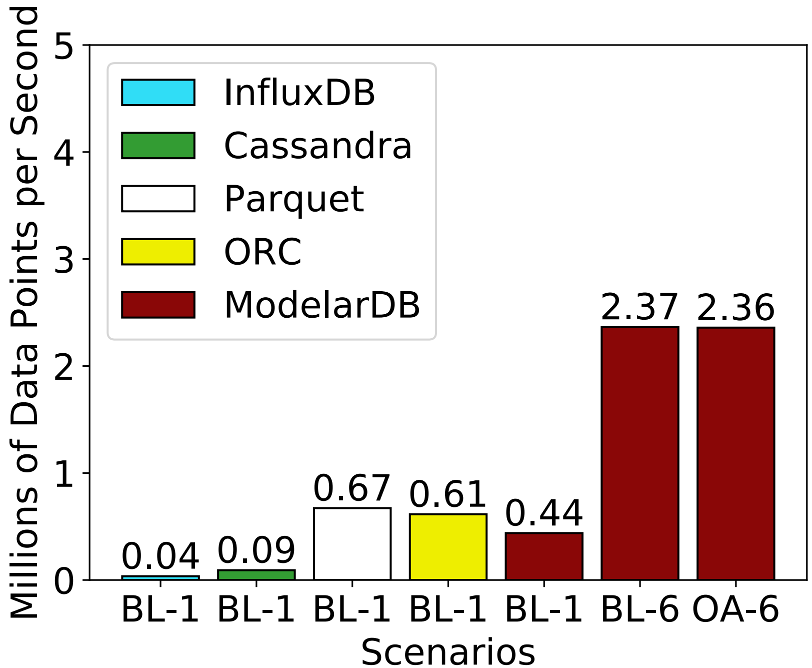


Figure: Ingestion, Ext. REDD

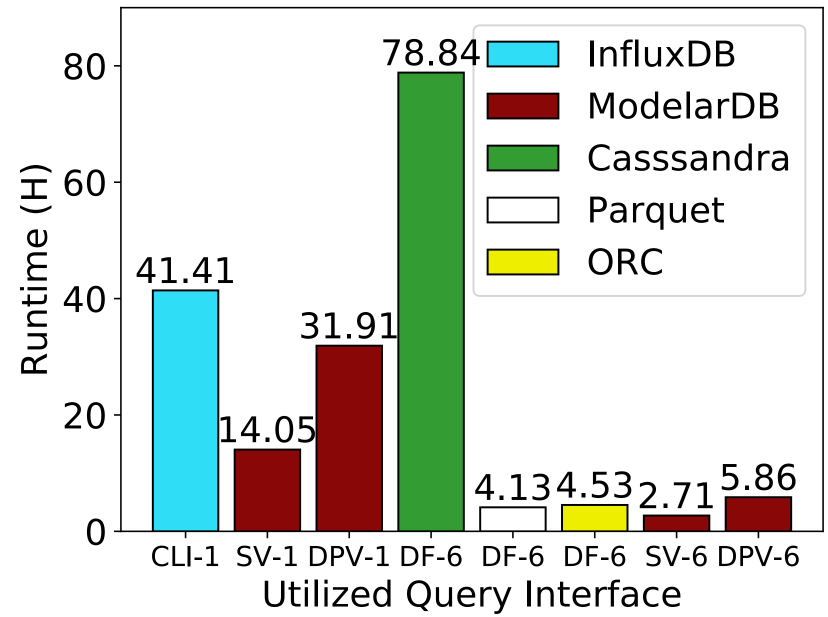


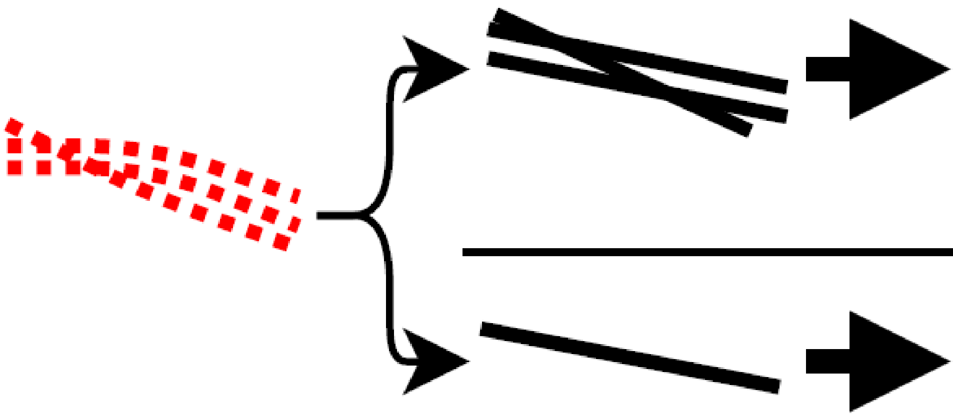
Figure: Aggregate Queries, Ext. REDD

- Only InfluxDB, Cassandra, and ModelarDB can answer queries while ingesting data points

Performance summary

- ModelarDB provides support for fast ingestion, good compression, and fast large aggregate queries
- ModelarDB remains competitive for small aggregate and point/range queries
- Other systems are good for *one* of these, but not both
- ModelarDB also supports queries while ingesting data

Next step: Exploiting correlation



Start Time	End Time	Model
100	2300	v = -0.047t+192.2
100	2300	v = -0.046t+180.1
100	2300	v = -0.057t+205.4

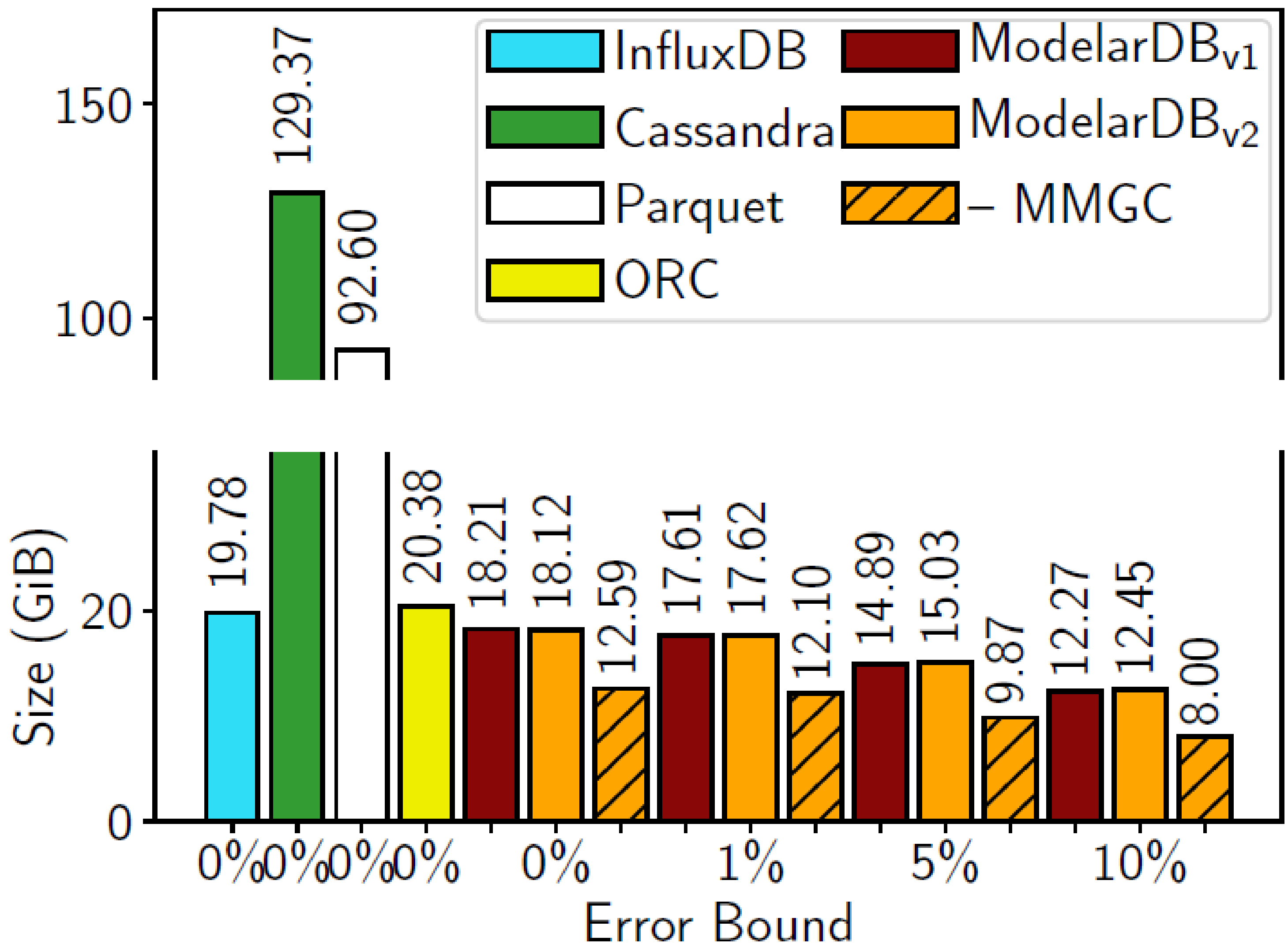
Start Time	End Time	Model
100	2300	v = -0.0465t+186.1

Specifying correlation

- Detecting correlation in data is an orthogonal problem
- We let the user hint correlation

- If the time series in a group cannot be represented by a single model, ModelarDB *splits* the group
  - Respects the error bound
- The time series can be *joined* again later

Evaluation



InfluxDB
Cassandra
Parquet
ORC
ModelarDB$_{v1}$
ModelarDB$_{v2}$
– MMGC
Size (GiB)
150
100
20
0
129.37
92.60
19.78
20.38
18.21
18.12
12.59
17.61
17.62
12.10
14.89
15.03
9.87
12.27
12.45
8.00
0% 0% 0% 0%
0%
1%
5%
10%
Error Bound

Conclusion and future work

- ModelarDB provides model-based compression within an error bound
- ModelarDB adapts to the dataset and compresses well by dynamically choosing among multiple models
- Good performance
- Integrated with Spark and Cassandra
- Future directions
  - Indexing to increase query performance further
  - Dynamic sampling
  - Advanced edge processing

More information

- S.K. Jensen, T.B. Pedersen, and C. Thomsen: ”ModelarDB: Modular Model-Based Time Series Management with Spark and Cassandra”, PVLDB 11(11), is available from http://www.vldb.org/pvldb/vol11/p1688-jensen.pdf

- S.K. Jensen, T.B. Pedersen, and C. Thomsen: ”Scalable Model-Based Management of Correlated Dimensional Time Series in ModelarDB” is available from https://arxiv.org/abs/1903.10269

Acknowledgments

This research was supported by

- the DiCyPS center funded by Innovation Fund Denmark
- the GOFLEX project EU grant agreement No 731232
- and Microsoft Azure for Research