Model-Based Time Series Management at Scale

PhD Thesis Defense

November 4th, 2019

Søren Kejser Jensen
skj@cs.aau.dk

Center for Data Intensive Systems, Daisy
Department of Computer Science
Aalborg University
Denmark

Supervisor: Torben Bach Pedersen
Co-Supervisor: Christian Thomsen

AALBORG UNIVERSITY
DENMARK

daisy
Center for Data-Intensive Systems

Agenda

Background
Motivation and Publications

- Denmark targets being self-reliance on renewable energy by 2050.[1]
- 61.6% of Denmark's total energy production was renewable in 2016.[2]
- 71.8% of renewable energy produced in 2016 comes from Wind.[2]
- The energy produced by a wind turbine can be reduced due to, e.g.:
  - The wings of the wind turbine being covered in ice.
  - The coating on the wings deteriorating over time.
  - Dust from fields being harvested clogs the exhaust.
- Damaged parts can cause months of downtime as new are delivered.
- Through extensive monitoring, problems with wind turbines can be preemptively detected and corrected during scheduled maintenance.

[1] The Danish Energy Model - Innovative, efficient and sustainable
[2] Environmental report for Danish electricity and CHP for 2016 Status Year

Background
Motivation and Publications

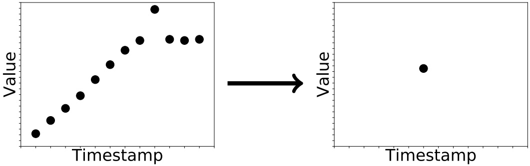
Meetings with manufacturers, owners, and energy traders:

- Modern turbines are monitored by up to 7,000 high-quality sensors.
- The sensors are installed with wired power and connectivity.
- Each sensor produces a data stream sampled to, e.g., a 10hz series.
- Collected measures include: Air Pressure, Humidity, Watt, Total Watt, Rotation Speed, Temperature, Wind Direction, Wind Speed.
- The time series are regular, cleaned, but missing values can occur.
- Aggregates with the following structure are the primary query type:
    - `SELECT {Aggregates} FROM {Table Name} WHERE time >= {Start} AND time < {End} {Optional Extra} GROUP BY time({Resolution}) {Optional Extra}`

Challenges
Motivation and Publications

Meetings with manufacturers, owners, and energy traders:

- The storage needed makes storing the high-frequency data infeasible.
- Manufactures of wind turbines gather petabytes of data each month.
- Simple aggregates (e.g. 10-minute averages) are stored instead of the high-frequent series, thereby removing fluctuations and outliers.



- Users believe problems can be found earlier with high-frequency data.
- Compression need only be lossless for some types of time series:
    - E.g., the amount of energy produced must be exact for billing.

Publications in the Thesis

This thesis proposes methods and algorithms for model-based time series management at scale, all implemented in the open-source *ModelarDB*.

Time Series Management Systems

[A] Time Series Management Systems: A Survey, TKDE'17

Model-Based Time Series Management

[B] ModelarDB: Modular Model-Based Time Series Management with Spark and Cassandra, PVLDB'18

[C] Scalable Model-Based Management of Correlated Dimensional Time Series in ModelarDB, Unpublished[3]

[D] Demonstration of ModelarDB: Model-Based Management of Dimensional Time Series, SIGMOD'19

[3] Currently under submission

Agenda

Paramount Properties

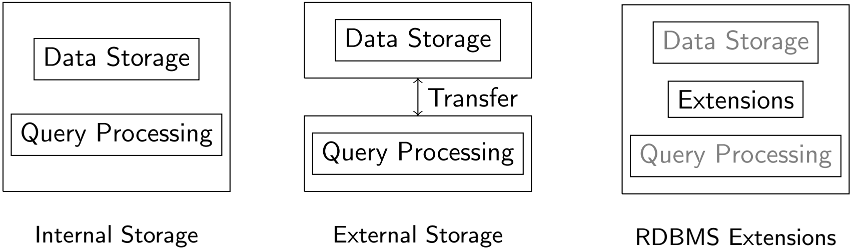
Time Series Management Systems

Paramount properties for systems managing wind turbine data:

- **Distribution:** The system must be able to scale to many nodes.
- **Stream Processing:** Data points are arriving continuously as a regular time series and must be queryable with a short latency.
- **Compression:** High compression is needed for high-frequency data.
- **Efficient Retrieval:** Indexes or ordered storage for fast retrieval.
- **Approximate Query Processing:** Approximate answers can be accepted for some time series and enables use of lossy compression.
- **Extensibility:** Allows users with domain knowledge to implement new storage methods optimized specifically for their data sets.

Architectures
Time Series Management Systems

- Existing systems were surveyed with a focus on the six properties.
- The survey showed that three different architectures were common.



Internal Storage

Time Series Management Systems

tsdb, FAQ, WearDrive, RINSE, Perera et al., Plato, Chronos, Pytsms, PhilDB

Advantages

- Each component can be optimized specifically for the system.
- Transfer of data between multiple sub-systems is not required.
- Simple to deploy as the system does not need external sub-systems.

Disadvantages

- Cannot reuse external sub-systems already deployed.
- Long development times if embeddable sub-systems are not used.

Missing Paramount Properties

- None of the systems can natively run distributed on a cluster.
- Most systems are not extensible so users cannot add new formats.

External Storage

Time Series Management Systems

TSDS, SciDB, Respawn, SensorGrid, Guo et al., Tristan, Druid, Huang et al., Williams et al., Bolt, Storacle, Gorilla, Unnamed, servIoTicy, BTrDB

Advantages

- External sub-systems can be reused to reduce development time, and simplify deployment if the sub-systems are already deployed.

Disadvantages

- Transfer of data between multiple sub-systems is required.
- The query processor is restricted to the data store's interface.
- Complex to deploy due to the multiple separate sub-systems.

Missing Paramount Properties

- Only very limited prototypes focus on general-purpose AQP.
- Most systems are not extensible so users cannot add new formats.

RDBMS Extensions

Time Series Management Systems

TimeTravel, F^{2}DB, Bakkalian et al.

Advantages

- Existing functionality can be reused to reduce development time.
- Transfer of data between multiple sub-systems is not required.
- Simple to deploy as the system does not need external sub-systems.

Disadvantages

- Unused functionality like transactions add overhead and complexity.
- The extensions are restricted to the API provided by each RDBMS.

Missing Paramount Properties

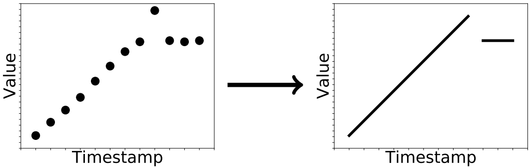
- None of the systems can natively run distributed on a cluster.
- TimeTravel and F^{2}DB focus on forecasting, not compression.

Agenda

Individual Time Series

Model-Based Compression

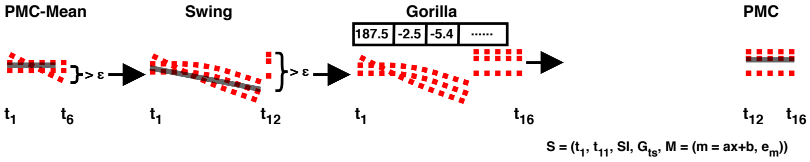
- A time series can be stored efficiently as a sequence of models.
- A model is any representation that can reconstruct the sub-sequence of values it represents within a known error bound (possibly zero):
  - E.g., $v = a \times t + b$ can represent a sub-sequence using only a and b.
- Fitted models are stored with the necessary metadata in a segment.
- A model type finds a model's parameters by fitting it to data points.
- Different model types can represent different structures efficiently.



- A model type dictates the representation and error function used.

Correlated Time Series
Model-Based Compression

- A data set of time series can contain redundant information:
  - Co-located temperature sensors will produce similar values.
- In ModelarDB we consider time series correlated for a set of model types if compressing them as a group reduces the storage required.
- A list of model types fit models to data points, e.g., a constant (PMC-Mean), linear (Swing), and lossless (Gorilla) model type:



- The model providing the best compression ratio is written to disk.

How to Group Time Series?
Model-Based Compression

Problems

- The model types used change the definition of correlation, so users should be able to create groups that match their model types.
- Resources required to compute correlation might not be available.
- A group of correlated series should be ingested together on a node.
- To prevent migration, groups should be made before ingestion starts.

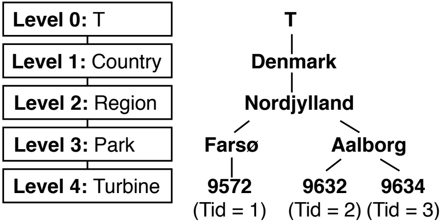
Solution

- Time series are grouped based on user hints given using primitives.
- The primitives can be combined and allow users to state that series are correlated based on their source or their dimensional hierarchy.
- Users can use their domain knowledge or analyze historical data.

Static Grouping
Model-Based Compression

Grouping 9632 and 9634:

- From specific sources:
  `9632 9634`
- Sharing a specific member:
  `Location 3 Aalborg`
- Share members until a level:
  `Location 3`
- The dimensions distance:
  `0.25`

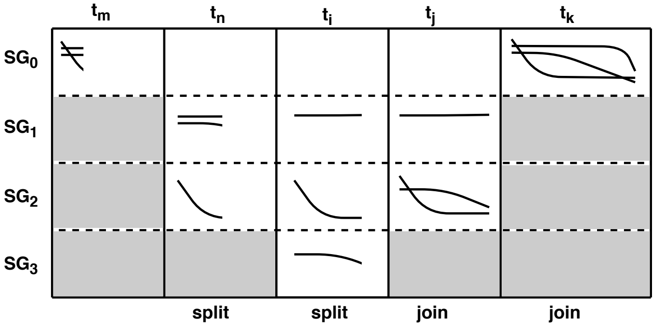


Location dimension for wind turbines

The most recent version of ModelarDB implements `auto` grouping based on the rule of thumb that using the shortest distance provides a benefit.

Dynamic Grouping
Model-Based Compression

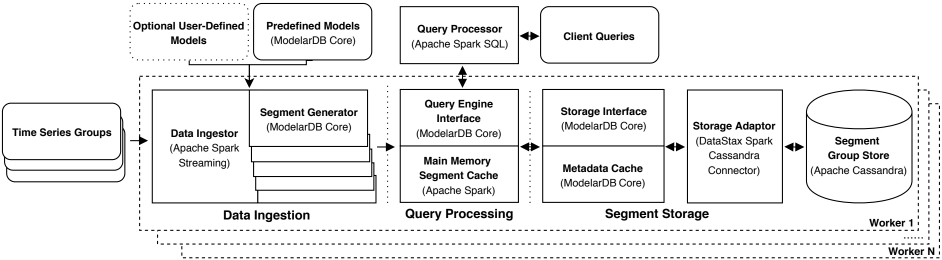
- Correlated series might temporarily not produce similar values:
    - E.g., A temperature sensor covered by a shadow and one that is not.
- This can be efficiently detected as the compression ratio is reduced.
- Dynamically splitting and joining groups remedies this problem:



Agenda

ModelarDB Architecture

Implementation of ModelarDB



- The general architecture consists of three sets of components: Data Ingestion, Query Processing, and Segment Storage.
- ModelarDB implements it using Apache Spark and Cassandra.
- External storage is used for extensibility and to simplify development.
- Extensible with regards to both data storage and query processing.

Storage

Implementation of ModelarDB

Segment

Gid (PK)	StartTime (PK)	Gaps (PK)	EndTime	Mid	Parameters
1	1460442200000	[]	1460442620000	1	0x3f50cfc0
3	1460642900000	[2]	1460645060000	2	0x3f1e ...
...	...	...	...	...	...

Model

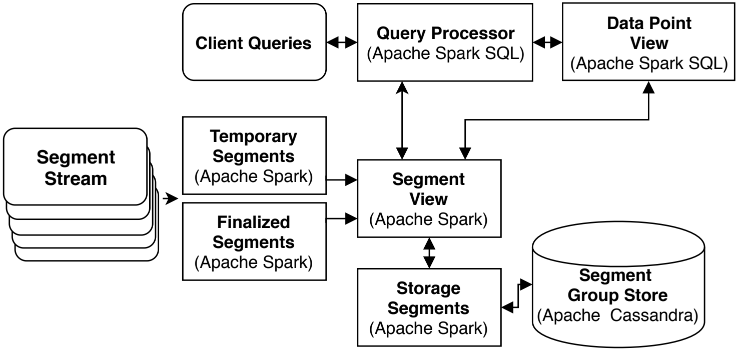
Mid (PK)	Classpath
1	PMC-Mean
2	Swing
3	Gorilla

Time Series

				Location Dimension					Nth Dimension			
Tid (PK)	Gid	Scaling	SI	Country	Region	Park	Entity	...	Level 1	Level 2	...	Level N
1	1	1.0	60000	Denmark	Nordjylland	Farsø	9572					
2	3	1.0	30000	Denmark	Nordjylland	Aalborg	9632					
3	3	4.75	30000	Denmark	Nordjylland	Aalborg	9634					

- Time Series and Model store metadata for series and model types.
- Segment stores sub-sequences of series as segments with a model.

Query Processing
Implementation of ModelarDB



- Queries are given in SQL and executed on segments or data points.
- Predicate push-down and code-generation reduce execution time.
- For low latency, temporary segments are regularly cached in memory.

Model and Segment API

Implementation of ModelarDB

Model	
`new(Mid, Error, Limit)`	●
`append([Data_Point])`	●
`initialize([[Data_Point]])`	●
`parameters(Start_Time, End_Time, SI, [[Data_Point]])`	●
`get(Tid, Start_Time, End_Time, SI, Parameters, Offsets)`	●
`length()`	●
`size(Start_Time, End_Time, Resolution, [[Data_Points]])`	●

Segment	
`get(Timestamp, Index)`	●
`min()`	○
`max()`	○
`sum()`	○

- Methods marked ● are required while ○ methods are optional.
- Model and Segment are split to reduce the size of segment.
- The interfaces are used both internally and for user-defined types.

Evaluation - Environment

Implementation of ModelarDB

- The evaluation uses real-life data sets from the energy domain.
- Most experiments are performed on a modest local cluster.
- Scalability experiments are performed using Microsoft Azure.
- For comparison the evaluation includes the state-of-the-art big data systems and file formats most commonly used in industry.[4]
  - InfluxDB, Apache Cassandra, Apache Parquet, and Apache ORC
- No model-based time series management system is publicly available.
- Small, large, and multi-dimensional aggregates are evaluated using:
  - A command-line interface (CLI), a Spark DataFrame (S), the Segment View (SV), or the Data Point View (DPV).

[4]From meeting with companies, our survey, and https://db-engines.com/en/

Evaluation - Data Sets

Implementation of ModelarDB

EP 45,353 time series collected from energy producers with a sampling interval of 60s and occupying 339 GiB as gzipped CSV.

- It contains two dimensions with each containing two levels:
  - **Production:** *Entity* $\rightarrow$ *Type* $\rightarrow \top$
  - **Measure:** *Concrete* $\rightarrow$ *Category* $\rightarrow \top$
- Three time series with energy production are grouped per entity.

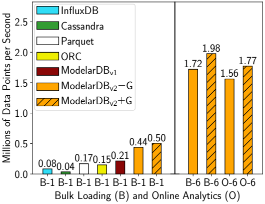
EH 286 time series collected from wind turbines with a sampling interval of 100ms and occupying 582.68 GiB as gzipped CSV.

- It contains two dimensions with three and two levels respectively:
  - **Location**: *Entity* $\rightarrow$ *Park* $\rightarrow$ *Country* $\rightarrow \top$
  - **Measure**: *Concrete* $\rightarrow$ *Category* $\rightarrow \top$
- Time series with the same concrete measures are grouped per park.

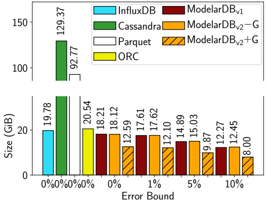
Evaluation - Ingestion and Storage

Implementation of ModelarDB

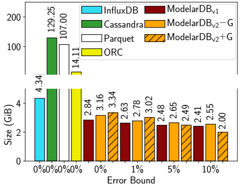
- Results with groups of correlated time series are shown with stripes.



Ingestion Rate, EP



Storage Used, EP

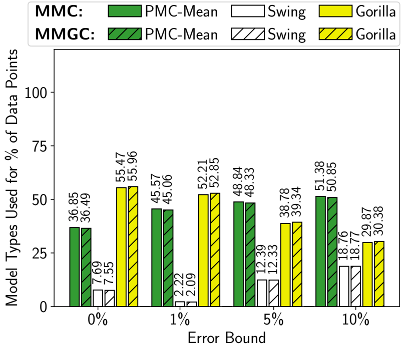


Storage Used, EH

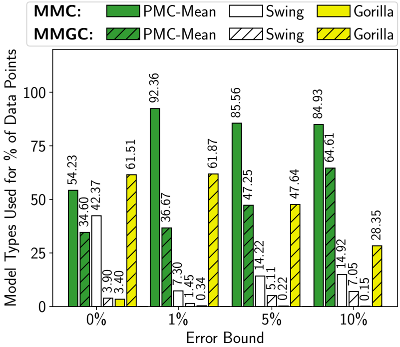
- Ingestion rate is 2.59–12.5 times faster than the industry formats.
- The storage required is reduced by 1.09–16.17 times for EP and by 1.30–64.63 times for EH compared to the industrial formats.
- Grouping series increases ingestion rate and can lower storage usage.

Evaluation - Model Types

Implementation of ModelarDB



Model Types, EP

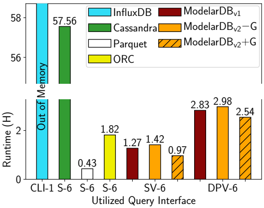


Model Types, EH

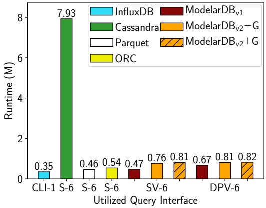
- ModelarDB automatically adapts to each data set and error bound.
- Grouping increases ModelarDB's use of the lossless model type.

Evaluation - Aggregate Queries

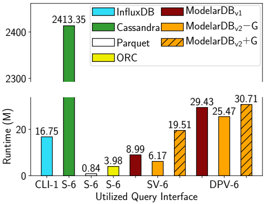
Implementation of ModelarDB



Large Scale, EP



Small Scale, EP

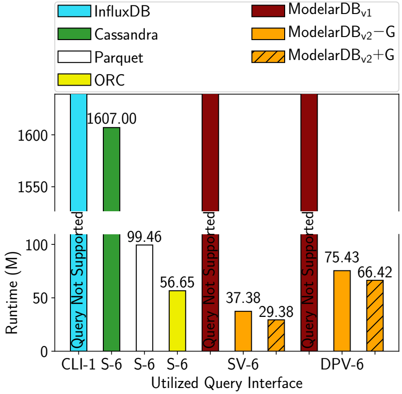


Small Scale, EH

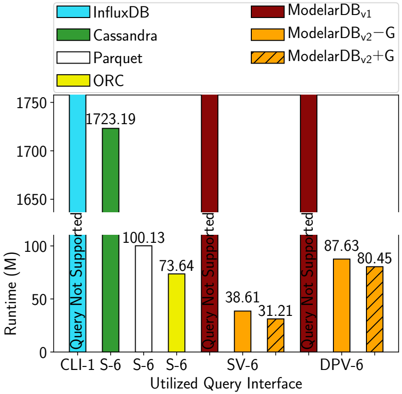
- Grouping time series decreases query time for large scale aggregate queries, but increases query time for small scale aggregate queries.
- Experiments run on Azure show that ModelarDB scales linearly.

Evaluation - Multi-Dimensional Queries EP

Implementation of ModelarDB



Month and Category, EP

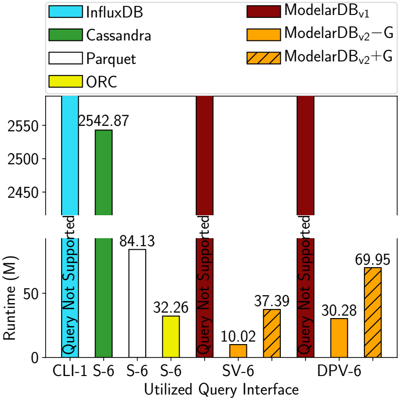


Month and Concrete, EP

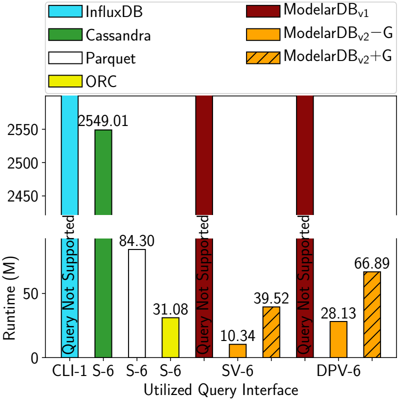
- Grouping reduces query time as each query only reads whole groups.

Evaluation - Multi-Dimensional Queries EH

Implementation of ModelarDB



Month and Park, EH



Month and Entity, EH

- Grouping increases query time as the cluster is not fully utilized.

Agenda

Summary of Contributions

Conclusion and Future Work

- A survey of existing time series management systems.

- An architecture for a model-based time series management systems.
- A model-agnostic compression algorithm for individual time series.
- A general schema for storing multiple time series as models.
- Methods for executing aggregate queries on user-defined models.
- Optimizations using predicate push-down and code-generation.

- Compression of correlated time series using multiple model types.
- Extensions of our methods from individual to correlated time series.
- Methods for partitioning time series into groups of correlated series.
- Methods for executing multi-dimensional aggregates on models.

- The extensible and modular implementation of ModelarDB.

Conclusion

Conclusion and Future Work

- Proposed a model-based approach for management of time series motivated by need for more detailed monitoring of wind turbines.
- Our ModelarDB system fulfills the paramount requirements:
  - **Distribution:** The system scales linearly to at least 32 nodes.
  - **Stream Processing:** Data is ingested and can be queried online.
  - **Compression:** Models efficiently represent each sub-sequence.
  - **Efficient Retrieval:** Data is partitioned and ordered for fast retrieval.
  - **AQP:** Allowing approximate answers reduce query response time.
  - **Extensibility:** Model types, query processor, and storage can change.
- ModelarDB hits a sweet spot and offers very fast ingestion, good compression, and fast, scalable online aggregate query processing.
- The contributions of this thesis increases the scale at which large time series can be collected, stored, and analyzed.

Future Work

Support new domains:

- Support time series with irregular sampling intervals.
- Evaluate our methods with data series from other domains.

Ingestion and storage optimizations:

- Support ingestion directly at the sources to save bandwidth.
- Design a native storage format optimized for model-based storage.
- Inform the user if the current model types provide poor compression.

Query processing:

- Index model values to answer queries faster and enable high level analytics directly on models.

Automated parameter selection:

- Reduce the number of parameters to only an error bound.

Our Modest Cluster