What can Hierarchies do for Data Streams?

Xuepeng Yin and Torben Bach Pedersen

Department of Computer Science

Aalborg University

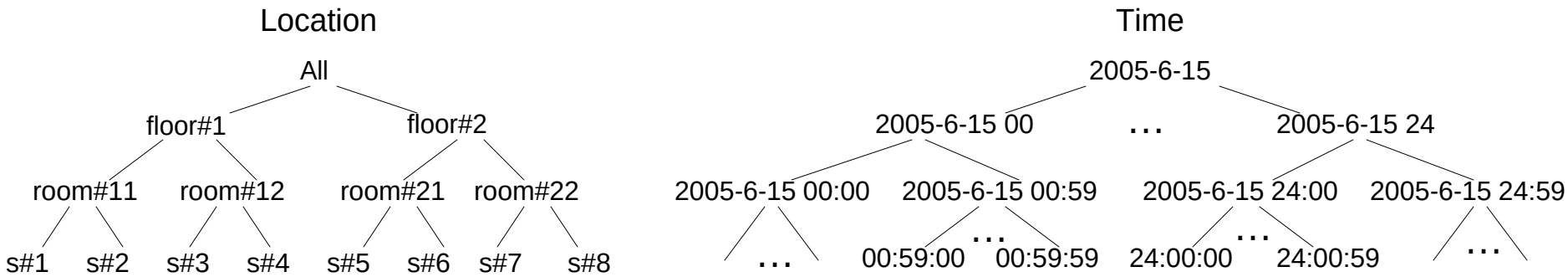
Outline

- Overview
- Stream query processing
- Comparison with Stanford CQL language
- Performance
- Conclusion and future work

Real-time OLAP on Data Streams

- Data streams
  - Sensor data, network flows, telephone calls and transaction logs
  - Special characteristics: rapid, unbounded, continuous, etc.
- Why?
  - To support the real-time enterprise better
  - It's still a very fresh topic
    - Traditional DBMS's are designed for static data
    - Recent data stream management systems are not OLAP-oriented
- Multi-dimensional and -granular data streams analysis
  - Characterize the stream data with multi-level dimensions
  - Allow information carried in data stream to be analyzed in different scales
  - Intuitively, reading data streams through human eyes

A Running Example



Given the input stream, SensorStream, show the rooms with average temperatures per minute higher than 25 degrees

r_1 (23.0,s#2,08:00:00)
r_2 (23.1,s#1,08:00:00)
r_3 (23.3,s#1,08:00:30)
p_1(* ,s#1,08:00)
r_4 (23.4,s#2,08:00:30)
p_2 (* ,s#2,08:00)
r_5 (25.0,s#1,08:01:00)
r_6 (25.0,s#2,08:01:00)
r_7 (26.3,s#1,08:01:30)
p_3(* ,s#1,08:01)
r_8 (26.1,s#2,08:01:30)
p_4(* ,s#2,08:01)

Group by room and minute →

aggregate →

r_1 (23.0,s#2,08:00:00)
r_2 (23.1,s#1,08:00:00)
r_3 (23.3,s#1,08:00:30)
t_1(23.4,room#21,08:00)
t_3(25.0,room#11,08:01)
r_6 (25.0,s#2,08:01:00)
r_7 (26.3,s#1,08:01:30)
r_8 (26.1,s#2,08:02:00)
t_2(25.6,room#11,08:01)

Temperature higher than 25

The SQL_{MS} language

- Show the rooms with average temperatures per minute higher than 25 degrees

```
SELECT CONTINUOUS AVG(Temperature), Room, Minute
FROM SensorStream
HAVING  AVG(Temperature)>25
```

- SQL_{MS} is a continuous language
  - Registered once, executed frequently until cancelled
- The output modes: CONTINUOUS and PERIODIC
  - CONTINUOUS: all input tuples participate in execution
  - PERIODIC: only tuples with timestamps ended by punctuations participate

The SQL_{MS} language and Punctuations

- The DRILLDOWN clause
  - For some specific dimension values, enables analysis on their descendants at finer levels
  - Example: show the hourly average temperatures on all the floors and those of the rooms on floor#1 in particular

```
SELECT CONTINUOUS AVG(temperature), Floor, Hour
FROM SensorStream
WHERE  Floor='floor#1'
DRILLDOWN DESCENDANTS('floor#1', Room)
```

- A punctuation marks the end of a subset of stream tuples
  - e.g.,

t_1 (23.0,s#1,08:00:00)

t_2 (23.1,s#1,08:00:30)

p (* ,s#1,08:00)

* for the wildcard matching any value

A constant matching itself only

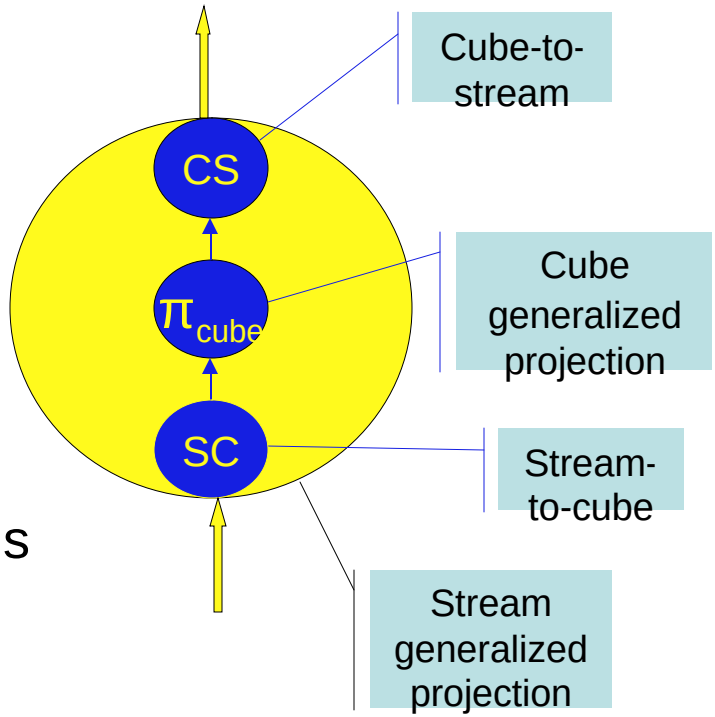
A dimension value with a partial order relationship to other values in the same column

The Cube and Stream Data Models

- A partial order between two levels, $L_{i1} \;_{i}\, L_{i2}$ and two dimension values, $e_1 \;_{D_i}\, e_2$
  - e.g., Room $\;_{\text{Location}}$ Floor and s#1 $\;_{\text{Location}}$ room#1
- A fact $(v_1, \ldots, v_m, \ldots, e_1, \ldots, e_n)$ is a tuple with the schema $(M_1, \ldots, M_m, \ldots, D_1, \ldots, D_n)$
  - M_j is a measure, D_i is a dimension
  - e.g., (23.1,s#1,08:00) with the schema (Temperature, Location, Time)
- Fact table R is a set of facts with the schema $(M_1, \ldots, M_m, \ldots, D_1, \ldots, D_n)$
  - e.g., $R_{SensorCube}$={(23.1,s#1,08:00) , (23.1,room#12,08:01) }
- A fact stream S is an ordered multiset of stream facts having the schema $(M_1, \ldots, M_m, \ldots, D_1, \ldots, D_n)$
- Example cube and stream: SensorCube and SensorStream
  - SensorCube: stored sensor data in one day
  - SensorStream: a live fact stream
  - Both have the schema (Temperature, Location, Time)

Stream Processing

- Inspired by Stanford STREAM
  - The stream-to-relation operator
  - The relation-to-relation operator
  - The relation-to-stream operator
- OLAP-like stream processing
  - The stream-to-cube operator
    - Breaks the input stream into cubes
    - Can be seen as a windowing operator
    - Time or tuple-based frequency
  - Cube operators for OLAP-like operations
    - Using existing cube operators (Yin & Pedersen, 2004)
    - Extended with multi-granularity support
  - Cube-to-stream
    - Historical data can also participate



Operators*

- Cube operators
  - The cube generalized projection, Π_{cube}
  - The cube selection operator, σ_{cube}
- Conversion operators
  - The stream-to-cube operator
  - The cube-to-stream operator
- Stream operators
  - The stream generalized projection, Π_{cube}
  - The stream selection, σ_{stream}

*Formal definitions can be found in the paper

Cube Generalized Projection (CGP)

- Turns the tuples into higher level facts and aggregates the measures correspondingly
- Allows multiple granularities in result tuples to enable roll-up and drill-down effects on certain levels
- Always aggregates lowest-level facts to be deterministic

Temperature	Location	Time
29	room#11	08
32	room#12	08
29	s#1	08
...		
32	s#8	08

Used for aggregation along the Location dimension

Example of the CGP Operator

- The example CGP operator:
  - computes the average temperatures per minute on the floors from 08:00:00 to 08:00:59
  - particularly, shows the temperatures of the rooms on floor#1

Algebra form of the operator:

$$\Pi_{Cube\{floor\#01, room\#0102\}, AVG(Temp), minute\{08:00:00\}}(Sensor) \bowtie Cube$$

A set containing a timestamp at the Minute level.

A set of values from the Floor and the Room levels of the Location dimension.

The operator produces a tuple each for the combinations:

$\{floor\#01, 08:00$ $\{floor\#02, 08:00$ $\{room\#0101, 08:00$ and $\{room\#0102, 08:00$

Note that sets of dimension values are used as parameters instead of levels, due to the multi-granular feature of the result tuples, like MDX.

Example of the CGP Operator (cont.)

$$\Pi_{Cube\{floor\#, room\#, AVG(Light), Time\}}(Sensor) \rightarrow SensorCube'$$

The input cube and the output cube are different in the fact table:

The input fact table: R

Light	Location	Time
60.0	s#1	08:00:00
66.8	s#2	08:00:00
62.1	s#3	08:00:00
70.3	s#4	08:00:00
60.2	s#1	08:00:30
66.9	s#2	08:00:30
62.1	s#3	08:00:30
70.5	s#4	08:00:30

$\{room\#11, 08:00$

$\{room\#12, 08:00$

$\{floor\#1, 08:00$

$\{floor\#2, 08:00$

The output fact table: R'

Light	Location	Time
63.475	room#11	08:00
66.25	room#12	08:00
64.8625	floor#1	08:00

The Cube Selection Operator

- Similar to the relational selection operator
- Levels at any granularity can be involved in predicates
- A tuple evaluates to true if containing the satisfying values or their descendants
- The example operator selects the tuples sent by sensors in room#11

$$\sigma_{Cube[Room\#=11]}(Sensor) = Cube\ Sensor\ C'$$

The input fact table: R

Temperature	Location	Time
60.0	s#1	08:00:00
66.8	s#2	08:00:00
62.1	s#3	08:00:00
70.3	s#4	08:00:00

The output fact table: R'

Temperature	Location	Time
60.0	s#1	08:00:00
66.8	s#2	08:00:00

The Stream-to-cube (SC) Operator

- Produces a cube using the received facts
- Triggered by time or tuple-based events
- Cubes can be seen as shifting windows
- Two output modes:
  - CONTINUOUS, converting all the input facts into the result cube
  - PERIODIC, only converting the subsets that are ended by punctuations.
  - Incomplete subsets are kept in input stream until complete
  - The subsets are defined by timestamp parameters
- Timestamp parameters
  - A timestamp is a range of time
  - Specify what tuples shall be eventually output in one cube
  - Timestamps can be multi-granular

Example of the SC Operator

$SC_{[\{08:00,08:01\},]}(S)$ produces cubes containing the tuples emitted during the periods 08:00 and 08:01

r_1 (23.0,s#2,08:00:00)
r_2 (23.1,s#1,08:00:00)
r_3 (23.3,s#1,08:00:30)
p_1(* ,s#1,08:00)
r_4 (23.4,s#2,08:00:30)
p_2 (* ,s#2,08:00)
r_5 (25.0,s#1,08:01:00)
r_6 (25.0,s#2,08:01:00)
Execution at 08:01:20

r_5 (25.0,s#1,08:01:00)
r_6(25.0,s#2,08:01:00)
r_7(26.3,s#1,08:01:30)
p_3(* ,s#1,08:01)
r_8(26.1,s#2,08:01:30)
p_4(* ,s#2,08:01)
r_9(26.6,s#1,08:02:00)
r_{10}(26.6,s#1,08:02:30)
Execution at 08:02:32

PERIODIC CONTINUOUS PERIODIC CONTINUOUS

The Cube-to-stream (CS) Operator

- Presents a cube in a streamed fashion
- Historical data can also participate
- Explicitly generates punctuations
- The example operator appends facts from the input cube to *O*

$$CS(Sensor\ Cube)$$

The fact table of SensorCube: R ⇨ The output stream: $O' = \{r_1, p_1, r_2, p_2, r_3, p_3\}$

Temperature	Location	Time
23.7	room#11	08:00
23.5	room#12	08:00
23.6	floor#1	08:00

r_1(23.7,room#11,08:00)
p_1(* ,room#12,08:00)
r_2(23.5,room#12,08:00)
p_2(* ,room#12,08:00)
r_3(23.6, floor#1 ,08:00)
p_3(* , floor#1 ,08:00)

Stream Generalized Projection (SGP)

- The SGP operator performs roll-up and aggregate operations on the current facts in the input stream, and outputs the results in a stream fashion
- Can be seen as a composition of conversion and cube operators
- The CONTINUOUS mode updates aggregation results continuously
- The PERIODIC mode is suitable for holistic aggregates

Example of the SGP Operator

The SGP operator

$$\Pi_{s\ t\ r\ e[a(m\{\ r\#o1o1m\}M,\ Q\ 0D\ 8E\ :A0V0G,\ T0\ e8m:\ p0\ e1r\}a)\ t>\mu\ ]r\ (e\ )}\ (SensorStrean$$

- computes the average temperatures per minute for room#11 from 08:00 to 08:01
- outputs the same result as

$$C\ (\Pi_{c\ u[\ \%\ \#e\ 1r1\ o\ ,o\{\ m\ 8A\ 0V0<G\ 0B\ :e\ 0m1>\}p\ ]\ e\ )r[\ \ 0t\ 8u:\ 0\ 0e,\ 0M8\ O\ 0D1\ E,\ ]}\ S\ e\ (n\ s\ o\ )\ r\ S\ t\ )\ r\ e\ a)\ m$$

- MODE=CONTINUOUS, temperatures are updated in every execution and gradually reach the final values
- MODE=PERIODIC, an execution produces unique and final temperature values for the complete subsets

Example of the SGP Operator (cont.)

$$\Pi_{s\,t\,r\,e[\,a(\,m[\,\#\,o1\,o1m\}M,\ (O\ 0D8E:A0V0G,\ T<a8\,m\ p\ d\ 1\ r\}\alpha)\,t,u]\,r(e)}(SensorStrea$$

r_1 (23.0,s#2,08:00:00)
r_2 (23.1,s#1,08:00:00)
r_3 (23.3,s#1,08:00:30)
p_1(* ,s#1,08:00)
r_4 (23.4,s#2,08:00:30)
p_2 (* ,s#2,08:00)
r_5 (25.0,s#1,08:01:00)
r_6 (25.0,s#2,08:01:00)

Execution at 08:01:20

t_1(23.2,room#11,08:00)

t_1(23.2,room#11,08:00)

t_2(25.0,room#11,08:01)

r_5 (25.0,s#1,08:01:00)
r_6(25.0,s#2,08:01:00)
r_7(26.3,s#1,08:01:30)
p_3(* ,s#1,08:01)
r_8(26.1,s#2,08:01:30)
p_4(* ,s#2,08:01)
r_9(26.6,s#1,08:02:00)
r_{10}(26.6,s#1,08:02:30)

Execution at 08:02:32

t_1(25.6,room#11,08:01)

t_2(25.6,room#11,08:01)

PERIODIC CONTINUOUS PERIODIC CONTINUOUS

The Stream Selection Operator

- The operator evaluates the predicate on the facts in the input stream and outputs the satisfying facts to an output stream
- Can be seen as a composition of conversion and cube operators
- The example operator appends the facts sent by sensors in room#11 to the output stream *O*

$$\sigma_{Stream[mRoom\#=11]}(SensorStream)$$

$$SensorStream = \{r_1, r_2, r_3, r_4\}$$

$$O' = \{r_1, p_1, r_2, p_2\}$$

r_1(23.7,s#1,08:00:00)
r_2(23.5,s#2,08:00:00)
r_3(23.5,s#3,08:00:00)
r_4(23.6,s#4,08:00:00)

r_1(23.7,s#1,08:00:00)
p_1(* ,s#1,08:00:00)
r_2(23.5,s#2,08:00:00)
p_2(* ,s#2,08:00:00)

Comparison with CQL

- *Continuous Query Language* (CQL) of the Stanford STREAM project
  - Based on SQL
  - Uses the base columns of the streams
  - Uses joins to include other columns to in query results
  - Uses unions for multi-granular results
- To show a dedicated OLAP-oriented stream language, e.g., SQL_{MS}, is desirable
- The conclusion also applies to other SQL-based stream languages
- The task: show the average temperatures per hour and minute for the entire building, all the floors and the rooms on floor two

The SQL_{MS} query:

```
SELECT PERIODIC AVG(temperature), Location.ALL, Time.Hour
FROM SensorStream
DRILLDOWN    DESCENDANTS(Location.All, Location.Floor)
             DESCENDANTS(Location.'floor#2', Location.Room)
             DESCENDANTS(Time.Hour, Time.Minute)
```

Comparison with CQL

The CQL query:

```
SELECT Rstream(AVG(temperature) AS  temp, NULL AS room, NULL AS floor, 'Overall' AS building)
FROM SensorStream[Range 1H, Slide 1H]
                UNION
SELECT Rstream(AVG(temperature) AS  temp, NULL AS room, floor AS floor, NULL AS building)
FROM SensorStream[Range 1H, Slide 1H], SensorLocation
WHERE  SensorStream.location=SensorLocation.location
GROUP BY floor
                UNION
SELECT Rstream(AVG(temperature) AS  temp, room AS room, NULL AS floor, NULL AS building)
FROM SensorStream[Range 1H, Slide 1H], SensorLocation
WHERE  SensorStream.location=SensorLocation.location AND SensorLocation.floor='floor#2'
GROUP BY room
                UNION
SELECT Rstream(AVG(temperature) AS  temp, NULL AS room, NULL AS floor, 'Overall' AS building)
FROM SensorStream[Range 1M, Slide 1M]
                UNION
SELECT Rstream(AVG(temperature) AS  temp, NULL AS room, floor AS floor, NULL AS building)
FROM SensorStream[Range 1M, Slide 1M], SensorLocation
WHERE  SensorStream.location=SensorLocation.location
GROUP BY floor
                UNION
SELECT Rstream(AVG(temperature) AS  temp, room AS room, NULL AS floor, NULL AS building)
FROM SensorStream[Range 1M, Slide 1M], SensorLocation
WHERE  SensorStream.location=SensorLocation.location AND SensorLocation.floor='floor#2'
GROUP BY room
```

Comparison with CQL

CQL results

(temp, room, floor, building, timestamp)

(26.0, room#21, NULL, NULL, 2005-06-15 08:00:59)
(26.2, room#22, NULL, NULL, 2005-06-15 08:00:59)
(24.5, NULL, floor#1, NULL, 2005-06-15 08:00:59)
(26.1, NULL, floor#2, NULL, 2005-06-15 08:00:59)
(25.3, NULL, NULL, Overall, 2005-06-15 08:00:59)
. . . . . .

(26.4, room#21, NULL, NULL, 2005-06-15 12:59:59)
(26.8, room#22, NULL, NULL, 2005-06-15 12:59:59)
(25.0, NULL, floor#1, NULL, 2005-06-15 12:59:59)
(25.8, NULL, floor#2, NULL, 2005-06-15 12:59:59)
(25.4, NULL, NULL, Overall, 2005-06-15 12:59:59)

SQL_{MS} results

(Temperature, Location, timestamp)

(26.0, room#21, 2005-06-15 08:00)
(26.2, room#22, 2005-06-15 08:00)
(24.5, floor#1, 2005-06-15 08:00)
(26.1, floor#2, 2005-06-15 08:00)
(25.3, Overall, 2005-06-15 08:00)
. . . . . .

(26.4, room#21, 2005-06-15 12)
(26.8, room#22, 2005-06-15 12)
(25.0, floor#1, 2005-06-15 12)
(25.8, floor#2, 2005-06-15 12)
(25.4, Overall, 2005-06-15 12)

Comparison of queries:

	# of lines	# of unions	# of characters
SQL_{MS}	5	0	199
CQL	31	6	990

Comparison of results:

	# of rows	# of columns	# of cells	# of NULLs
SQL_{MS}	1525	3	4575	0
CQL	1525	5	7625	3050

Comparison with CQL

- Task 2: show the average temperature per hour per floor
- Task 3: if the average temperature of the whole building exceeds 30 degrees, then show the floors with average temperatures higher than 32 degrees, and also the temperatures of each room on these floors

		# of lines	# of unions	# of characters	# of cells
Task 2	SQL_{MS}	2	0	77	12
	CQL	4	0	142	12
Task 3	SQL_{MS}	6	0	255	24
	CQL	27	3	834	40

Conclusion: for OLAP-like queries, SQL_{MS} is more intuitive and compact, as well as easier to construct. This also gives a potential for more efficient query evaluation.

Performance Study

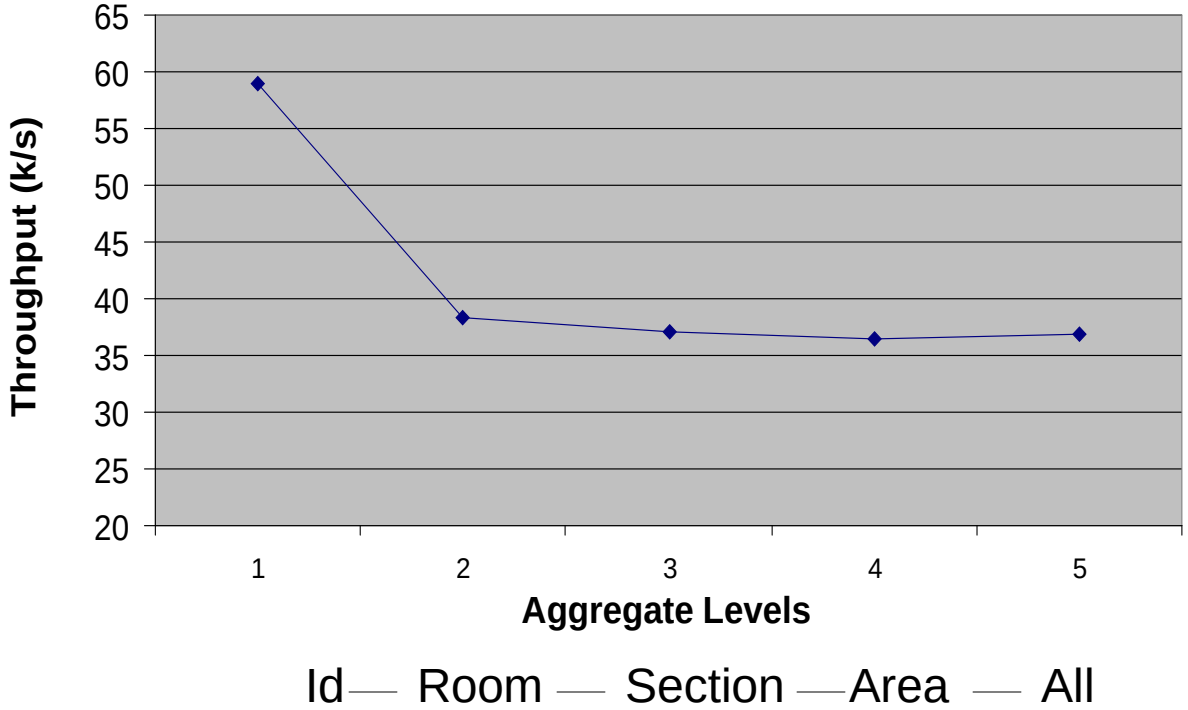
- Test platform
  - 1.86GHz Laptop, 1.5G ram
  - Windows XP SP 2
  - MS .NET 1.1

- Data
  - Intel Lab sensor data
  - 2.3 million readings, 150MB totally
  - 500,000 tuples, 35.8MB for experiments

- Transmission rate, approx. 64k tuples/second

- Implementation environment: Visual Studio .NET 20003 and C#

Throughput VS. Aggregate levels

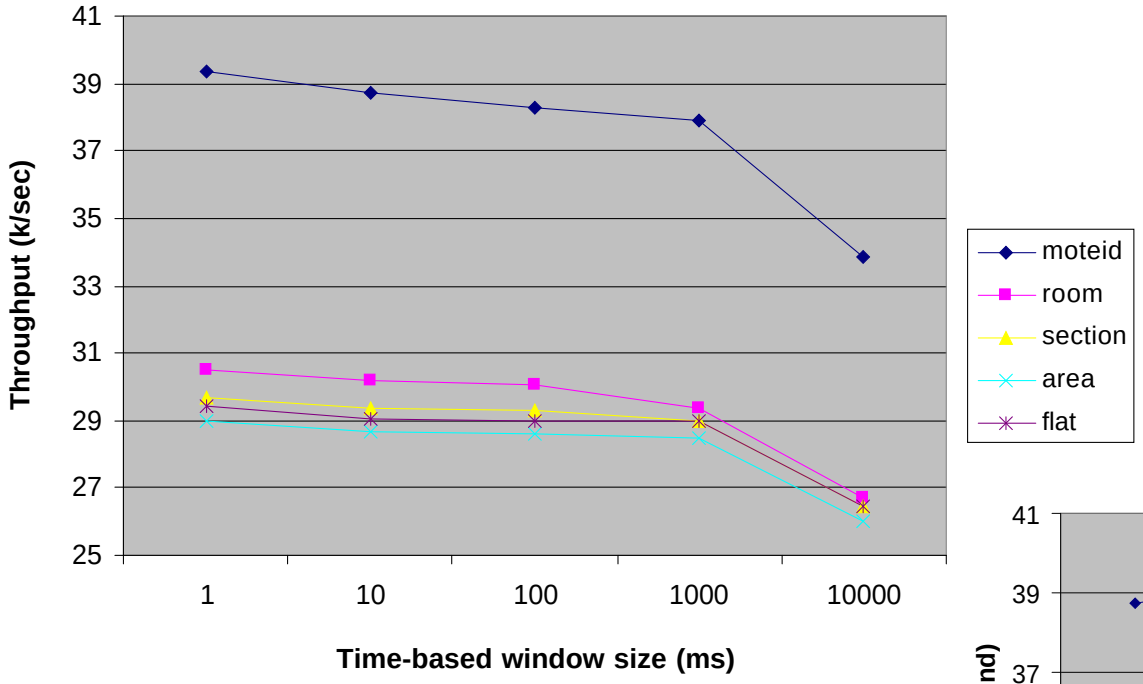


Average temperature per *level*:

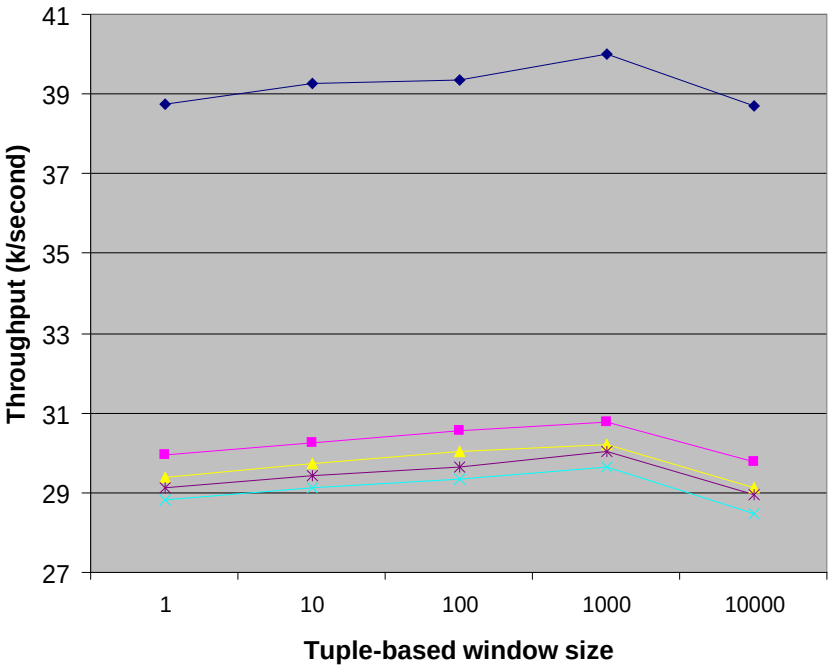
SELECT CONTINUOUS
AVG(temperature),
level
FROM SensorStream

Conclusion: throughput is close to transmission rate when aggregating on the bottom level. Aggregating on higher levels do not differ throughputs very much due to use of hashtables for roll-up operations.

Throughput VS. Window Size



Conclusion: too many tuples aggregated in each execution slows down the performance



Conclusion: aggregating the buffered tuples too frequently also slows down the performance

Conclusion and Future Work

- We have discussed:
  - Real-time enterprise makes OLAP-like operations on data streams desirable
  - The stream operators make multi-dimensional analysis on data streams possible
  - The experiments show promising performance
- Future work
  - More optimizations on the stream engine
    - Reuse of finer pre-aggregations or data-reduction by multi-level aggregations
    - Approximations, e.g., sampling
    - Compile a SQL_{MS} query into machine code
  - Experiments

Related Work

- OLAP data modeling and querying
  - Built on static data (Chatziantoniou & Ross, 1996;Jagadish, Lakshmanan, & Srivastava, 1999)
  - OLAP and XML federation (Yin & Pedersen, 2004)
- Data Stream Management Systems
  - STREAM (The Stream group, 2003), Gigascope (Cranor et al., 2002), Aurora (Carney et al., 2002), etc.
  - Query languages are SQL-based
  - Operators are analogous to relational algebra
  - Multi-level aggregation, e.g., Gigascope (Zhang et al., 2005), an optimization for computing multiple aggregation queries, does not enable multiple aggregations in one query nor roll-up to higher levels

References

- Chatziantoniou, D., Ross, K.A.: Querying Multiple Features of Groups in RelationalDatabases. In *Proc. of VLDB*. (1996) 295–306
- Carney D. et al.: Monitoring Streams - A New Class of Data Management Applications. In *Proc. of VLDB*. (2002) 215–226
- Cranor C. D. et al.: Gigascope: High Performance Network Monitoring with an SQL Interface. In *Proc. of SIGMOD*. (2002) 623
- Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D.: What can Hierarchies do for DataWarehouses? In *Proc. of VLDB*. (1999) 530–541
- The STREAM group: STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin* **26**(1) (2003) 19–26
- Yin, X., Pedersen, T.B.: Evaluating XML-Extended OLAP Queries Based on a Physical Algebra. In *Proc. of DOLAP*. (2004) 73–82
- Zhang R. et al.: Multiple Aggregations Over Data Streams. In *Proc. of SIGMOD*. (2005) 299–310

?