Information Technology
& Data Analytics

Real-time Data Fusion @ Scale - *Boeing use cases for temporal, time series and geospatial analytics*

Ian Willson, PhD, Senior Technical Fellow, Data Engineering

EE392B IoT lecture, Stanford University, April 30, 2019

ian.a.willson@boeing.com

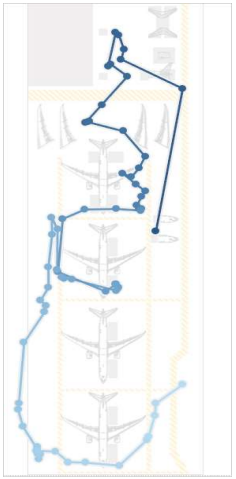
Data Engineering & IoT – Agenda

Lecture Perspective:

- My role - solve Boeing's data engineering problems @ scale (when it doesn't work out of the box) on our evolving platform(s)
- Not an overview of IoT, Boeing IoT or Boeing's AnalytX platform
- Not covering the rapid evolution of factory automation and AI
- Not intended to be suggest a particular implementation

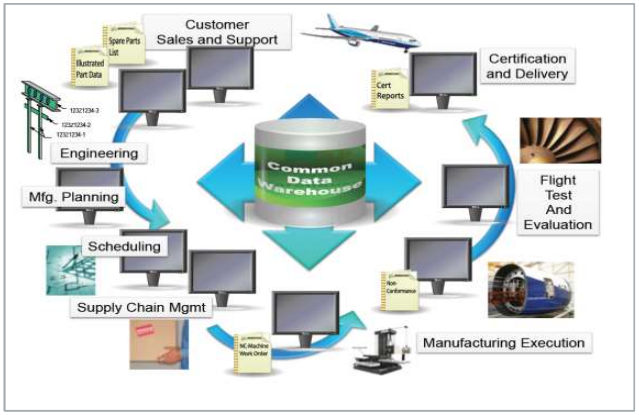
Agenda:

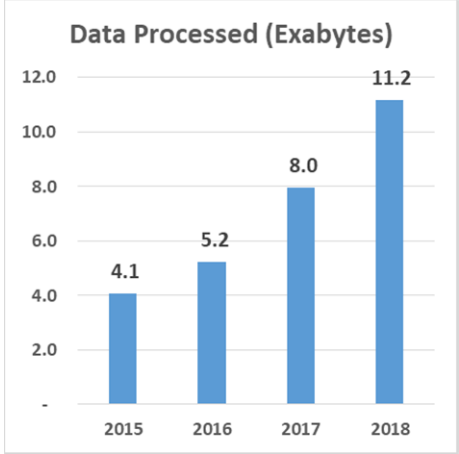
- Long history of advanced MPP RDBMS use
- Extending temporal to IoT data
- Run-time temporal concurrency analysis of log data
- RFID data – geo-spatial temporal analytics
- Real-time geo-spatial temporal IoT analytics
- Native time series data storage & real-time query
- Other considerations



Parallel Data Analytics @ Boeing

- Many innovations on MPP EDW (36 rack system in 1999)
- **Boeing AnalytX** platform widely used
  - Internal data platform: Hadoop (HDP) + MPP RDBMS
  - 64B SQL queries, 11 Exabytes of data processed (2018)
  - Most source system feeds are real-time (including IoT)
  - Direct SQL users (5K, most not IT or data scientists)
  - Data re-use (154x/day) due effective data caching





Innovations:

- Temporal Eng./Manuf. EDW (2 patents, widely cited)
- 10ms ragged hierarchy BOM queries (up or down N levels)
- RFID real-time home-built middleware, 3M tags
- Run-time temporal geo-spatial analytics on non-temporal data from event logs and location data
- Integration of time series with temporal & geo-spatial

Temporal SQL For IoT Data

For data that can represent a period of time, leverage SQL temporal data types & operators, normalize at rest:

PERIOD **[**Start, End**)** where End can be null or year 9999

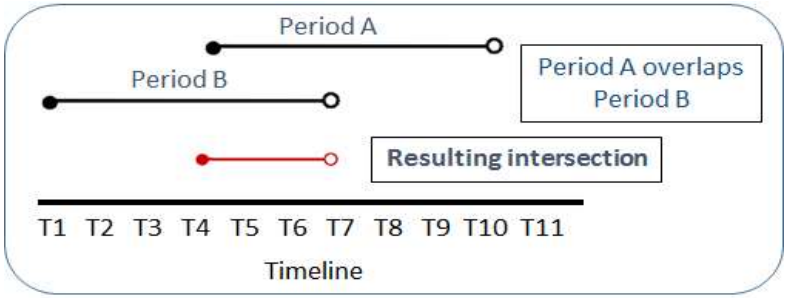
To efficiently and accurately find concurrency:

SELECT NORMALIZE ID, A.Period P_INTERSECT B.Period
FROM A INNER JOIN B ON A.Period OVERLAPS B.Period …

NORMALIZE cuts result set to 3 rows (vs. 1,226 rows @ 1Hz)
OVERLAPS is accurate regardless of duration or frequency
P_INTERACT returns the exact intersection of two conditions

Query Step Result:

- Single join step cut CPU 57%, elapsed time 65%
- 99.7% reduction in result rows, fully accurate vs. non-temporal reducing frequency to equality match points-in-time
- Full predictive analytics query (8 join example):
  - 4.7x faster, 87% less CPU vs. non-temporal on MPP
  - **4,354x faster** than mySQL (311x adjusted for HW)



Temporal normalized step result – 3 rows

ID	Validity Period	Duration (sec)
X	('2017-06-12 05:45:02.24', '2017-06-12 05:47:04.24')	122
X	('2017-06-12 05:47:04.24', '2017-06-12 05:47:08.24')	4
X	('2017-06-12 05:47:08.24', '2017-06-12 06:05:28.24')	1100

Query step results compared

Join 2 sensors on time	Matching Rows	Result Rows	Time (sec)	CPU (sec)	IO (GB)	Accuracy
Temporal normalized SQL	6,950,696	19,825	190	4,881	786	Entirely
Conventional SQL	12,370,239	5,804,641	548	11,419	1,954	Mostly
Temporal Savings		99.7%	65.4%	57.3%	59.8%	

Temporal Analytics For Log Data

- Data has beginning and end times (any data source)
- At run-time, build time periods via temporal SQL
- At run-time, perform time expansion at desired granularity to create 1 row/time interval

```
SELECT BEGIN(log_period)  FROM LOG_DATA  WHERE desired filter…
EXPAND ON PERIOD(starttime, endtime) AS log_period BY INTERVAL '0.01' SECOND
```

- Add SQL to find the busiest time periods, qualify on values, compute statistics at any level of granularity

```
SELECT TOP 20 period_centi_sec, count(*) FROM ( … ) DET
(period_centi_sec)   GROUP BY 1 ORDER BY 2 DESC
```

Centi-Second Period	Concurrency
2017-04-03 09:23:20.910	244
2017-04-03 09:30:35.240	237
2017-04-03 09:30:39.260	236

```
SELECT TOP 20 cast(period_centi_sec as char(16)), AVG(Cnt),
MAX(Cnt) FROM (SELECT period_centi_sec, count(*) FROM
( … ) DET (period_centi_sec)   GROUP BY 1)
HS (period_centi_sec, Cnt) GROUP BY 1 ORDER BY 2 DESC
```

Minute (start)	Avg. Concur.	Max Concur.
2017-04-03 09:31	202.2	231
2017-04-03 09:30	201.8	237
2017-04-03 09:23	186.2	244
2017-04-03 09:27	179.4	204
2017-04-03 09:32	171.4	212

Factory RFID Use Case

Boeing designed middleware processes active and passive RFID tag data across our sites, then replicates to DBMS

- 300+ buildings in 12 states, 2M+ tags, billions of rows

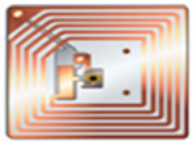
Temporal normalization is very useful for RFID tag data:

- Consecutive readings in ‘same’ location -> 1 row
- 86,400 rows/tag -> 5 exact location rows or 2 zone rows

SQL analytics combines temporal & geo-spatial SQL:

- Track parts, moving assets relative to time & location

However, many use cases can’t wait for mini-batch loads (build & normalize time period, convert to geo-spatial, etc.)



TagID	Location (e.g. lat/long)	Timestamp
002413F6	47.9061N, 122.2814W	2015-03-31 13:41:23
002413F6	47.9061N, 122.2814W	2015-03-31 13:41:24
002413F6	47.9061N, 122.2814W	2015-03-31 13:41:25
002413F6	47.9061N, 122.2814W	2015-03-31 13:41:26
002413F6	47.9061N, 122.2814W	2015-03-31 13:41:27
002413F6	47.9062N, 122.2815W	2015-03-31 13:41:28
002413F6	47.9062N, 122.2815W	2015-03-31 13:41:29
002413F6	47.9062N, 122.2815W	2015-03-31 13:41:30
002413F6	47.9062N, 122.2815W	2015-03-31 13:41:31
002413F6	47.9062N, 122.2815W	2015-03-31 13:41:32
002413F6	47.9064N, 122.2816W	2015-03-31 13:41:33
002413F6	47.9065N, 122.2816W	2015-03-31 13:41:34
002413F6	47.9066N, 122.2816W	2015-03-31 13:41:35
002413F6	47.9066N, 122.2816W	(… fixed rest of day)
86,400 rows/tag/day @1Hz (5MB/tag/day)		

Temporal normalization

TagID	Location (e.g. lat/long)	Time Period	Duration
002413F6	47.9061N, 122.2814W	[2015-03-31 13:41:23, 2015-03-31 13:41:28)	5
002413F6	47.9062N, 122.2815W	[2015-03-31 13:41:28, 2015-03-31 13:41:33)	5
002413F6	47.9064N, 122.2816W	[2015-03-31 13:41:33, 2015-03-31 13:41:27)	1
002413F6	47.9065N, 122.2816W	[2015-03-31 13:41:34, 2015-03-31 13:41:27)	1
002413F6	47.9066N, 122.2816W	[2015-03-31 13:41:35, 2015-04-01 13:41:23)	86,388
Temporal normalization - 1 row per consecutive geo location - 5 rows/tag/day			

Temporal normalization on geo zone

TagID	Geo Zone	Time Period	Duration
002413F6	Building 12 Room A	[2015-03-31 13:41:23, 2015-03-31 13:41:33)	10
002413F6	Building X Room Z	[2015-03-31 13:41:33, 2015-04-01 13:41:23)	86,390
Temporal geo-coded normalization - 1 row per consecutive geo zone location - 2 rows/tag/day			

Real-Time RFID Temporal Geo-Spatial SQL

Run analytics directly on raw replicated data, no separate ETL

Real-time replication from broker, data every 5-7 sec (active RFID)

- Adjust to expected ping-rates for each tag, watch for gaps
- Set geo-location accuracy to normalize on (6 digit XY for ~0.1m)

High normalization rates (333x on equality)

- Varies by tag type (moving vehicle to tool), accuracy of reader, geo conversion, 3,042x compression (fact table, BLC) vs. text

RFID data integration with EDW & data lake for BI, dashboards & data science, run-time SQL builds time period, returns dwell time:

Asset Type	Tags	Avg. Dwell (sec)	Norm. Rate (X)
Contract Tools	242	3,200	640
Hand Tools	228	14,026	2,805
Ground Support Equipment	578	6,738	1,348
Safety-Emergency	69	1,298	260
Vehicle	84	193	39

Tag_ID	Period Start	Location Dwell (sec)	Latitude	Longitude
XXXX	2017-04-02 16:56:29	0:00:24.000000	47.925719	-122.267436
XXXX	2017-04-02 16:55:47	0:00:42.000000	47.925707	-122.267411
XXXX	2017-04-02 16:54:57	0:00:50.000000	47.925719	-122.267436
XXXX	2017-04-02 16:49:59	0:04:58.000000	47.925707	-122.267412
XXXX	2017-04-02 16:48:45	0:01:14.000000	47.925719	-122.267436

```
SELECT Tag_ID, BEGIN(Validity_TP), INTERVAL(Validity_TP) HOUR(4) TO SECOND, ConvY, ConvX, Bldg, PERIOD(starttime, endtime) as
Validity_TP FROM

(SELECT entityId, CAST(convertedLocationX as Decimal(15,6)) as LocX , CAST(convertedLocationY as Decimal(15,6)) as LocY ,
reportedtime, MAX(reportedtime) over (Partition by entityId order by reportedtime ROWS BETWEEN 1 FOLLOWING AND 1 FOLLOWING)
as endtime, NEW ST_Geometry('ST_Point', LocX , LocY), Bldg FROM AIT_TAGHISTORY_V WHERE ENTITYID = 'XXX') ME (Tag_ID,
ConvX, ConvY, starttime, endtime, ConvertedXY_GEO, Bldg)

WHERE starttime <> endtime and (endtime > starttime or endtime is null) ORDER BY 2 DESC;
```

Real-Time RFID Analytics Results

Use run-time temporal SQL OVERLAP join condition on dynamically created time periods and geo-spatial conditions
Queries run in a few seconds (total latency ~10 sec)

Resulting data is fed into R to apply algorithms:

- Density based clustering of readings
- Combine with EDW data (shop floor data, bill of material)
- Join location to route to analyze autonomous vehicles (on or off planned route, duration parked)

Infer assembly status by movement of tagged items (line move)

- Visualizations tools show asset utilization, availability
- Location-based alert from algorithms & real-time EDW data
- Develop & train models for edge deployment

However – query cost grew too much with data size

Tag ID	Period Start	Dist. Point (m)	Dwell Time (sec)	Latitude	Longitude
XXXX	2017-06-21 13:15:46.000	372.1	40	47.923432	-122.267073
XXXX	2017-06-21 12:08:52.000	372.4	4,014	47.923427	-122.267060
XXXX	2017-06-21 12:08:36.000	373.5	16	47.923417	-122.267062
XXXX	2017-06-21 11:36:09.000	372.6	1,947	47.923426	-122.267062
XXXX	2017-06-21 11:32:15.000	371.3	234	47.923436	-122.267053
XXXX	2017-06-21 07:30:42.000	372.5	14,493	47.923426	-122.267061

Analytics Case	Rows	Time (sec)	CPU (sec)	CPU Skew	IO Skew	IO (GB)
1 tag 12 hr dist & dwell	24	1.71	148.3	1.8	2.5	11.3
51 tags 1 hr dist & dwell	130	3.47	664.3	2.2	2.9	78.2

Native Time Series RFID Storage & Simple Retrieval

- Only get 'new readings' (location changed from last received), normalized upstream
- Only difference is data layout & indexing:
  - **Non Time-Series**: hash on a surrogate ID for data distribution, add index on the RFID Tag ID
  - **Time-Series**: hash on RFID Tag + Time Bucket
    - Storage reduced 39% (both compressed)
- Run simple queries on small ranges of time and tags with 2X look back on time series due to normalization
  - Queries run in ~0.10 sec on any sized data table
- Build additional run-time temporal geo-spatial analytics on top at minimal additional cost

```
SELECT entityId, convertedLocationX, convertedLocationY, lastupdatetime
FROM BCDW_DEV_NONCORE_T.AIT_TAGHISTORY_TIMES_IW
WHERE ENTITYID in ('0024DFC9', '0024225F', '0024226A', '0024226D', '0024226E')
AND TD_TIMECODE >= cast('2018-05-10 12:25:00' as timestamp)
AND TD_TIMECODE <= cast('2018-05-10 12:25:00' as timestamp) + INTERVAL '15' MINUTE
ORDER BY 4 DESC;
```

```
)
PRIMARY INDEX ( ID )
INDEX AIT_TAGHISTORY_ENTITYID_NUSI ( ENTITYID );

)
PRIMARY TIME INDEX (TIMESTAMP(6), DATE '2010-01-01', MINUTES(60),
COLUMNS(ENTITYID), NONSEQUENCED);
```

	Name	Type	CurrentPerm
10	AIT_TAGHISTORY_ORIG_IW	Table	23,921,336,320
11	AIT_TAGHISTORY_TIMES2_IW	Table	14,470,975,488

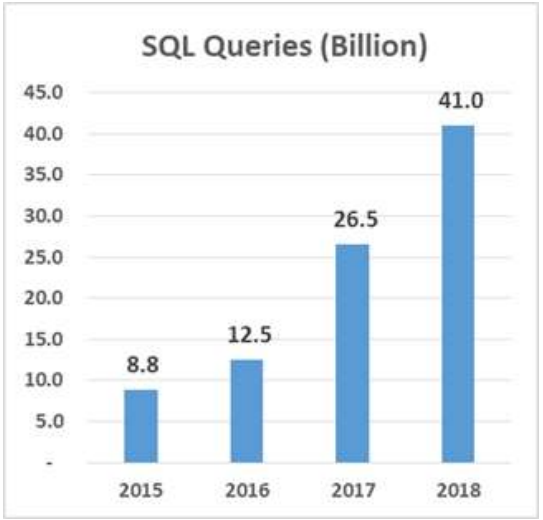
Row Count - BCDW_DEV_NONC

	Count(*)
1	162,027,816

	CPU	Runtime	IO (KB)
Original Avg.	10.91	0.49	17,718,356
Time Series Avg.	2.06	0.08	1,009,407
Avg. Reduction (%)	**81%**	**84%**	**94%**

Other Considerations

- Data ingest has changed, nearly all via real-time replication
  - Hundreds of concurrent fine-grained ingest processes, quick but less efficient than bulk methods, huge volume
  - Perm. storage extends to object stores (on-prem & cloud)
- Need to capture, publish and re-use 20+ years of work (API view)
  - Essential to manage & optimize heterogeneous systems
- What to run where – choices are much broader than just the methods presented (edge, central, hybrid, many components)
  - Automation with AI will play a key role in dynamic system management & more broadly run-time workload placement
- Native support vs. extending vs. building on top
  - Boeing initially built and patented a metadata-driven temporal normalization & load process
- Upstream normalization of IoT data needs to be partly undone for time series retrieval (need 1 time period/bucket for retrieval)
- When to normalize & how much should be carefully considered



Rows	10 hour sample @ 1Hz
36,000	1 parameter, raw data
2,262	Normalized on full equality
67	Normalized on 1 less decimal digit