Information Technology
& Data Analytics

IoT @ Boeing: From Flight Data Recorders to Factory RFID Analytics

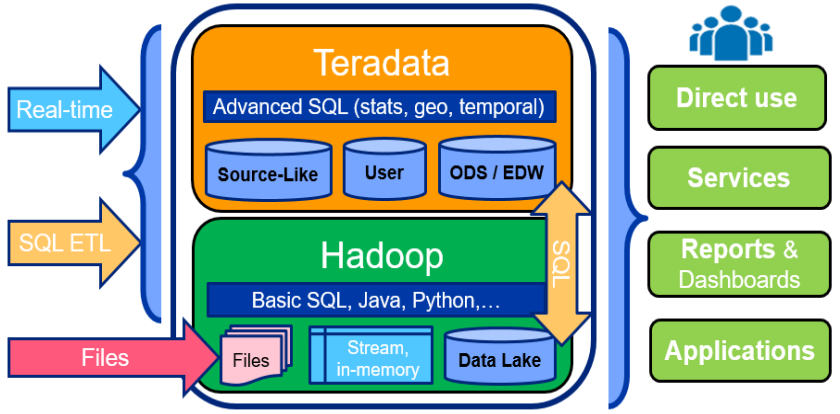
Ian Willson, PhD, Senior Technical Fellow, Data Engineering
Boeing AnalytX

Presented at: XLDB 2018, Stanford University, May 1, 2018

ian.a.willson@boeing.com

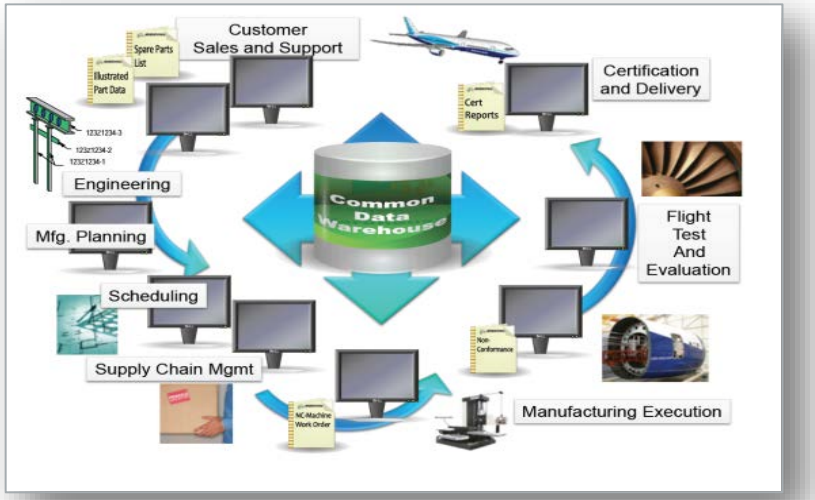
Parallel Data Analytics @ Boeing

- Teradata DBMS for 28 years, UDA for 5 years (HDP, QG)
- Wide range of data (engineering to service), 8M attributes
- Many source EDW feeds are real-time dbms replication
- Basis for Boeing's AnalytX platform (2017 launch)
- Self service SQL users (700 new vs. yr. ago, 100 managers)
- Growth (47% yr. ago, 47PB SQL IO peak day, 80x - 2006)



Atypical Teradata Use Cases:

- Tactical & low-latency (91% of EDW SQL <= 0.01 sec)
- Real-time web-based BOM navigation (50K users)
- Data re-use (150x/day) -> 97% in-memory (automatic)



Selected Boeing Innovations:

- Temporal normalized EDW (2 patents, normalize all but)
- 10ms ragged hierarchy queries (up or down N levels)
- RFID real-time feed via home-built middleware, 2M+ tags

IoT Example 1 – Airplane Sensor Data (FDR)

SQL analytics (diagnose/predict) on **Flight Data Recorder:**

- 15GB / 787 flight (59M rows) text, 0.2-20Hz, 1,675 sensors
- **Trillions of rows** at a fleet level (10M+ flights)
- *100 flight hour analytics qry took 217 hours (mySQL, 3 servers)*

Time-alignment (conditions A + B + C concurrently) explodes complexity + non-temporal SQL solutions can be inaccurate:

- Time is in *ms* offsets, recording rates vary -> how to align?
- Round time to join, may discard higher freq. data (A.sec=B.sec)

	Sensor 6		Sensor 7		Non
Flight ID	Millisecond Offset	Whole second	Millisecond Offset	Whole second	Temporal Result
4900053	675	0	714	0	**0**
4900053	1,675	1	1,714	1	1
4900053	2,675	2	2,714	2	2
4900053	3,675	3	3,714	3	3
115 rows from second 4-118 for sensors & result					
4900053	119,675	119	119,714	119	119
4900053	120,675	120	120,714	120	120
4900053	121,675	121			
4900053	122,675	122			
4900053	123,675	123			
4900053	124,675	124			
4900053	125,675	125	125,714	125	**125**
4900053	126,675	126	126,714	126	126
4900053	127,675	127	127,714	127	127
4900053	128,675	128	128,714	128	128
4900053	129,675	129			
29 rows from second 130-158 for sensor 6, no result					
4900053	159,675	159	159,714	159	**159**
1,100 rows from second 160-1260 for sensors & result = **1,226 total rows**					

Solution – Temporal Normalization + MPP SQL:

- Build time periods, collapse consecutive identical readings with no gaps given sensor frequency into a single row
- ~7x row normalization reduction for commercial aircraft
- 105x net size reduction from plain text (DB compression x temporal normalization x 4 column fact table design)
- Align time quickly & precisely using temporal SQL operators

Flight ID	Sensor ID	Time ms Offset	Value
4,900,004	31	15,390	2.67
4,900,004	31	15,640	2.67
4,900,004	31	15,890	2.67
	100 more rows		
4,900,004	31	41,140	2.67

Normalizing process

Flight ID	Sensor ID	Validity	Value	Rows Collapsed
4,900,004	31	('2000-01-01 00:00:15.390', '2000-01-01 00:00:41.390')	2.67	104

Airplane Sensor Analytics via Temporal SQL

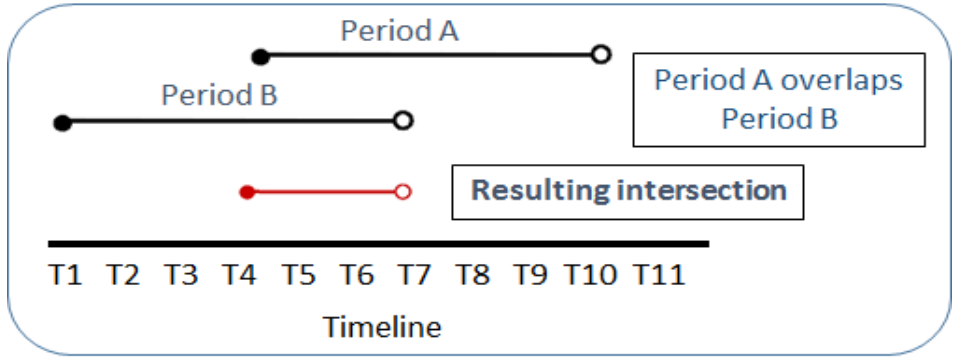
Faster, simpler & fully accurate temporal SQL:

```
SELECT NORMALIZE FlightID, A.Period P_INTERSECT
B.Period FROM A INNER JOIN B ON A.Period OVERLAPS
B.Period AND …
```

OVERLAPS is accurate regardless of duration or frequency

P_INTERACT returns the exact intersection of two conditions

NORMALIZE cuts result set to 3 rows (vs. 1,226 rows @ 1Hz)



Prognostic Analytics Query Result:

- Find concurrent time for conditions on 3 sensors
- Join 6 sensors time-correlated to prior result, calculate 3 min deviation for each second, apply further conditions
- Single join step cut CPU 57%, elapsed time 65%
- Full prognostic query (8 joins):
  - 4.7x faster, 87% less CPU vs. non-temporal on MPP
  - **4,354x faster** than mySQL (311x adjusted for HW)
- 3 accurate rows vs. 1,226 inaccurate rows (prior page)

Temporal normalized step result – 3 rows

Flight ID	Validity Period (temporal)	Duration seconds
4,900,053	('2011-06-30 00:00:00.714', '2011-06-30 00:02:01.714')	122
4,900,053	('2011-06-30 00:02:05.714', '2011-06-30 00:02:09.714')	4
4,900,053	('2011-06-30 00:02:39.714', '2011-06-30 00:20:59.714')	1,100

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%	

Result - Airplane Sensor Data Pipeline

Mini-batch load - **1 flight/sec** end-to-end (~90 flight batch)

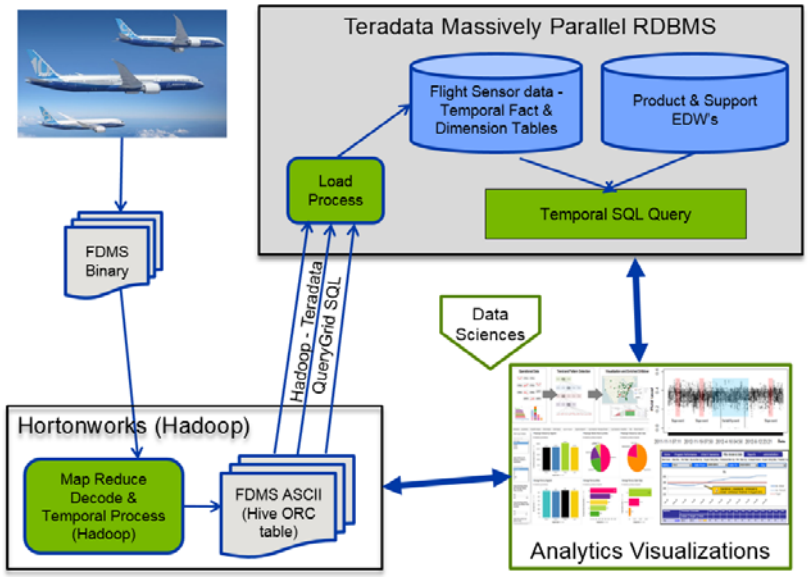
- Hadoop binary decode, normalize & store
- Hadoop used for ML, airline services, etc.
- QueryGrid Hadoop to TD final processing & load

Larger queries scale linearly (1K - 100K flight hours)

Normalizing on equality is the starting point, consider:

- Limit precision to measurement accuracy, further reduce as appropriate for task (2,262 rows -> 67)

During SQL analytics – normalize each step to collapse rows, particularly when only time period is needed



Rows	10 hour flight @ 1Hz
35,552	1 parameter, raw data
2,262	Normalized on equality
67	Normalized, 1 less decimal digit

Validity Period	Value
('2013-12-10 13:49:41', '2013-12-10 13:49:49')	11.94
('2013-12-10 13:49:49', '2013-12-10 13:49:50')	11.92
('2013-12-10 13:49:50', '2013-12-10 13:49:52')	11.94
('2013-12-10 13:49:52', '2013-12-10 13:50:00')	11.92
('2013-12-10 13:50:00', '2013-12-10 13:50:04')	11.90
… 1027 more rows until 2013-12_10 20:14.15	

Validity Period	Value
('2013-12-10 13:48:53', '2013-12-10 13:49:41')	12.0
('2013-12-10 13:49:41', '2013-12-10 20:14:15')	11.9
('2013-12-10 20:14:15', '2013-12-10 22:30:56')	11.8

IoT Example 2 – Factory RFID Data

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

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

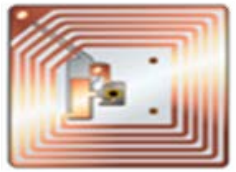
Temporal normalization is very useful for RFID tag data:

- 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

Key challenge was **latency** – many use cases could not 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 tag data integration with EDW and UDA data lake for BI, dashboards & data science productivity

- Run-time SQL builds time period via OLAP, returns dwell time:

Method	Rows (Billion)	Space (GB)
Conventional SQL	1,261	264,902
Temporal normalized on equality Teradata	4	87
	100K tags, 0.2Hz, 2 years	

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) as Period_Start_TS, INTERVAL(Validity_TP) HOUR(4) TO SECOND As Location_Dwell, ConvY as Latitude, ConvX as
Longitude, Buildingname, 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), buildingname FROM AIT_TAGHISTORY_V WHERE ENTITYID = 'XXX') ME (Tag_ID, ConvX, ConvY, starttime, endtime,
ConvertedXY_GEO, buildingname)

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)

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

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)

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

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

Challenges & Next Steps

Mythology - MPP Edition:

- New hardware options (NVMe + many CPU's) eliminates MPP/clever design
- Cloud – scaling up/down MPP is effortless, cloud is always cheaper
- App dev mind set (go get own data, do most processing in code in app)
- Open source is an easy swap in (but how to contribute, maturity, fixing bugs)
- Teradata is expensive (2XXX appliance nodes vs. Hadoop are ~3X)

Boeing AnalytX Platform Tasks:

- How to avoid silos of data & analytics in cloud native CI/CD modern teams
- How many tools and methods to use, disparity of capabilities, rapid change
- "4D" – add Teradata 16.20 Time Series tables & operators (in-work)
- Optimizing heterogeneous data access across HDFS, Hive, Teradata, …
- Optimizing AnalytX platform with rapidly evolving new authoring silos which partly do their own analytics (SAP, Dassault, SaaS providers)