Lessons learned from designing a QA automation for analytics databases (Big Data)

Omid Vahdaty, Big Data ninja.

Agenda

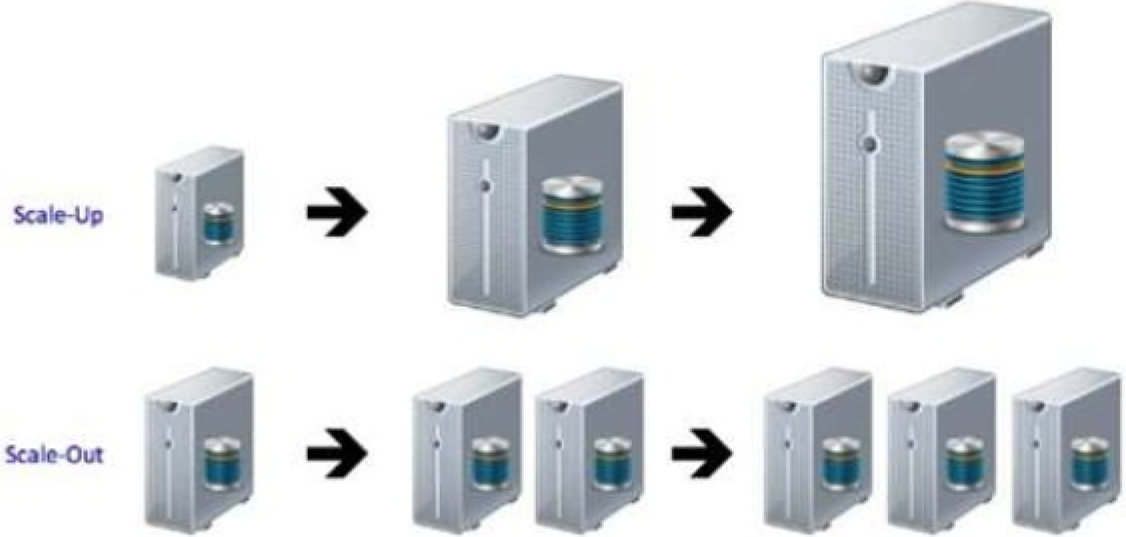
- Introduction to Big Data Testing
- Methodological approach for QA Automation road map
- True Story
- The Solution?
- Challenges of Big Data Testing.
- Performance & Maintenance.

Introduction to Big Data Testing

How Big is Big Data?

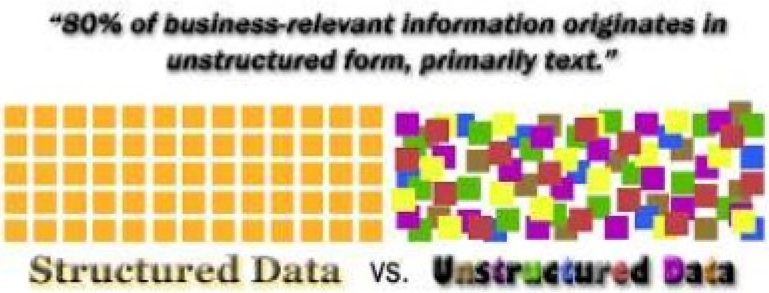
- 100M ?
- 1B ?
- 10B?
- 100B?
- 1Tr?

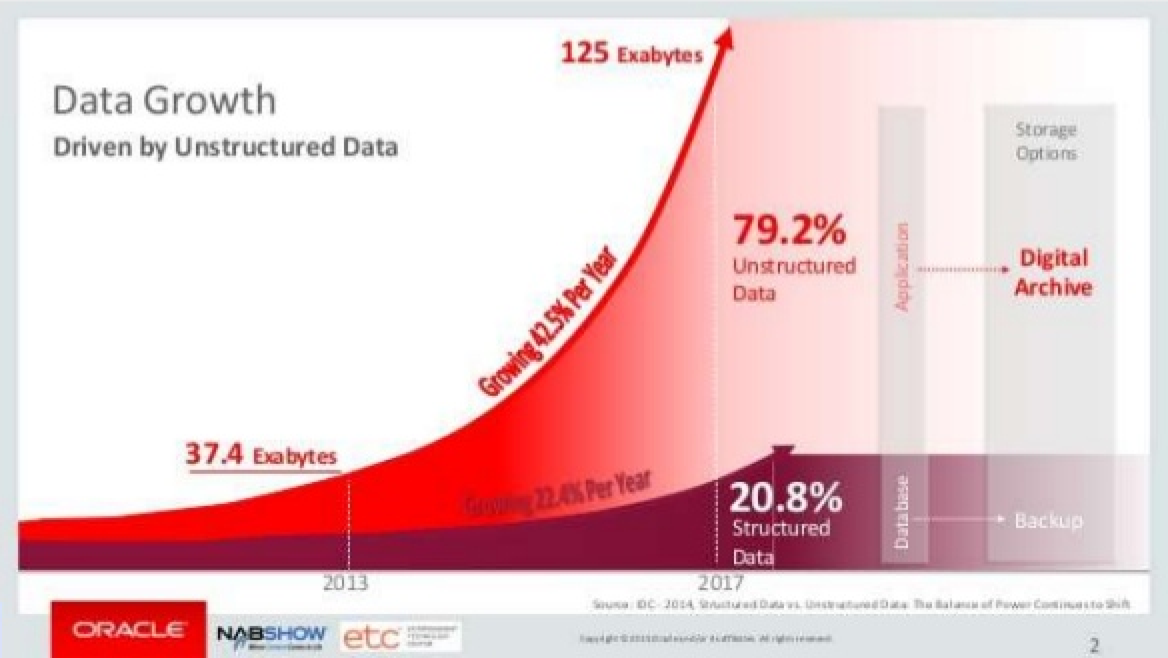
Scale OUT VS. Scale UP systems



Big Data ?

- Structured (SQL, tabular, strings, numbers)
- Unstructured (logs, pictures, binary, json,blob, video etc)


"80% of business-relevant information originates in unstructured form, primarily text."
Structured Data
vs.
Unstructured Data


Data Growth

Driven by Unstructured Data

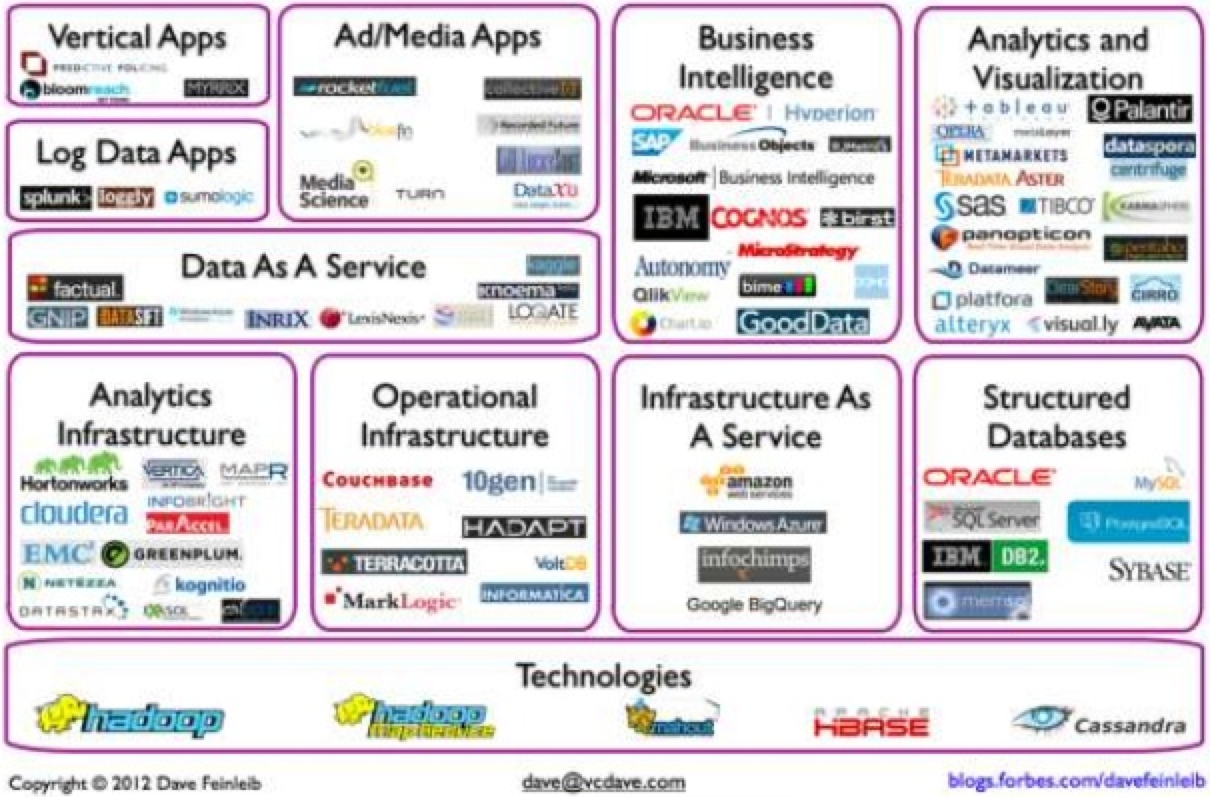
Source : IDC - 2014, Structured Data vs. Unstructured Data: The Balance of Power Continues to Shift

OLAP or OLTP?

- Simply put:
    - **One query** running on a **lot of data** for reporting/analytics
    - **Many queries** running quickly in parralel applications (e.g user login to a website, credentials are stored on a DB)

Type of Big Data products

- Databases
  - OLAP, OLTP
  - Sql, NO SQL
  - In Memory, Disk based.
  - Hadoop Ecosystem, Spark
- Ecosystem tools
  - ETL tools
  - Visualizations tools
- General Application
- Analytics Based products
  - Froud
  - Cyber
  - Finances and etc.



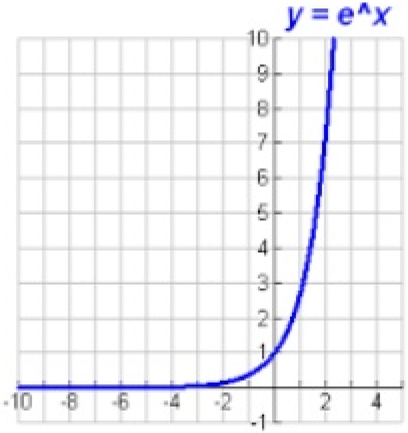
Copyright © 2012 Dave Feinleib dave@vcdave.com blogs.forbes.com/davefeinleib

Expectation Matching for today's lecture...

- OLAP Database
- Structured data only, Synthetic data was used in testing.
- (not about QA of analytics based products/solutions).
- (not about hadoop , event streaming cluster).
- (not try to sell anything)
- In a nutshell:
    - How to test an SQL based database?
    - What sort of challenges were met?

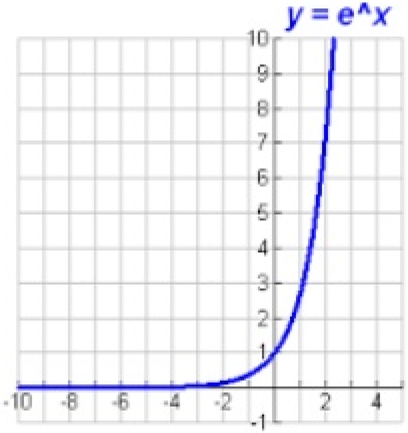
How Hard Can it be (aggregation)?

- Select Count(*) from t;
  - Assume
    - 1 Trillion records
    - ad-hoc query (no indexes)
    - Full Scan (no cache)
  - Challenges
    - Time it takes to compute?
    - IO bottleneck? What is IO pattern?
    - CPU bottleneck?
    - Scale up limitations?

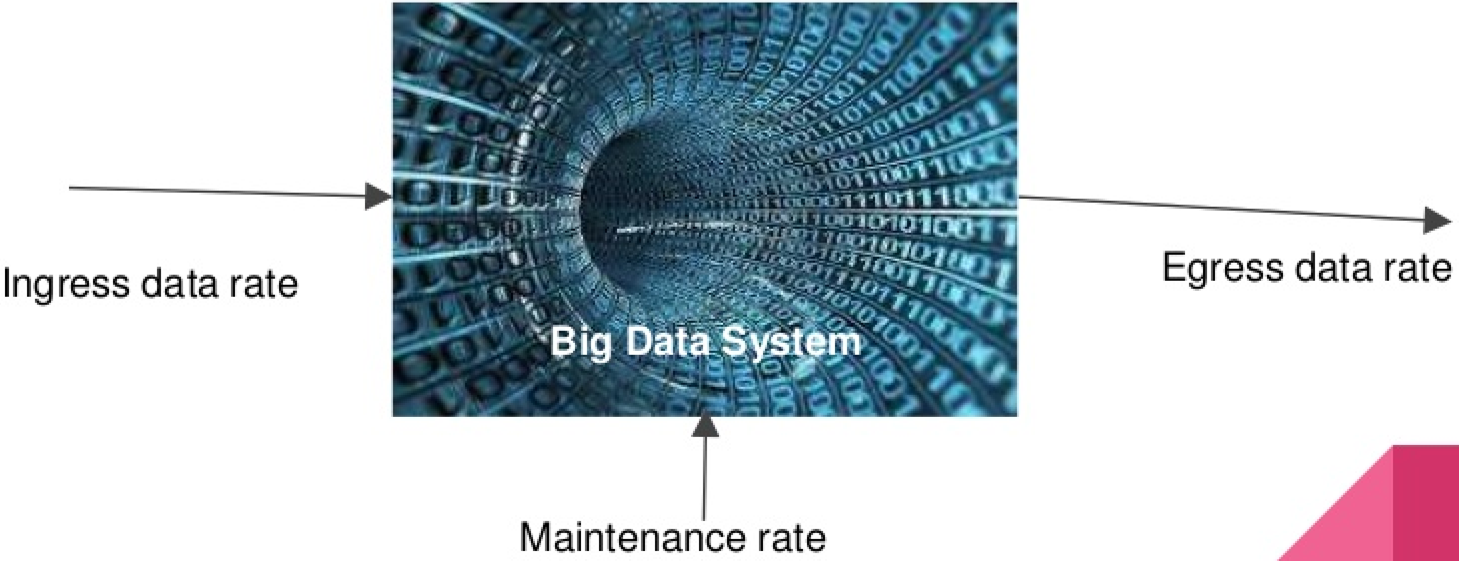


How Hard Can it be (join)?

- Select * from t1 join t2 where t1.id =t2.id
    - Assume
        - 1 Trillion records both tables.
        - ad-hoc query (no indexes)
        - Full Scan (no cache)
    - Challenges
        - Time it takes to compute?
        - IO bottleneck? What is IO pattern?
        - CPU bottleneck?
        - Scale out limitations?



3 Pain Points in Big Data DBMS



Ingress data rate

Big Data System

Egress data rate

Maintenance rate

Big Data ingress challenges

- Parsing of Data type
  - Strings
  - Dates
  - Floats
- Rate of data coming into the system
- ACID: Atomicity, Consistency, Isolation, Durability
- Compression on the fly (non unique values)
- amount of data
- On the fly analytics
- Time constraints (target: x rows per sec/hour)

Big Data egress challenges

- Sort
- Group by
- Reduce
- Join (sortMergeJoin)
- Data distribution
- Compression
- Theoretical Bottlenecks of hardware.

Methodological approach for QA Automation road map

Business constraints?

- Budget?
- Time to market?
- Resources available?
- Skill set available ?

Product requirements?

- What are the product’s supported use cases?
- Supported Scale?
- Supported rate of ingress?
- Supported rate on egress?
- Complexity of Cluster?
- High availability?

The Automation: Must have requirements?

- Scale out/up?
- Fault tolerance!
- Reproducibility from scratch!
- Reporting!
- “Views” to avoid duplication for testing?
- Orchestration of same environment per Developer!
- Debugging options
- versioning!!!

The method

- Phase 0: **get ready**
  - Product requirements & business constraints
  - Automation requirements
  - Creating a testing matrix based on insight and product features.
- Phase 1: **Get insights (some baselines)**
  - Test **Ingress** separately, find your **baseline**.
  - Test **Egress** separately, find your **baseline**.
  - Test **Ingress while Egress** in progress., find your **baseline**.
  - Stability and Performance
- Phase 2: Agile: **Design an automation system** that satisfies the requirements
  - **Prioritize** automation features based on your insights from phase 1.
  - **Implement** an automation infrastructure as soon as possible.
  - Update your **testing matrix** as you go (insight will keep coming)
  - **Analyze** the test results daily.
- Phase 3: **Cost reduction**
  - How to reduce compute time/ IO pattern/ network pattern/storage **footprint**
  - **Maintenance** time (build/package/deploy/monitor)
  - **Hardware** costs

True Story

Testing GPU based DBMS

(Peta Scale)

So why build a DBMS from scratch?

- Most **compiler** are designed for **CPU** → **execution tree** for CPU runtime engine → **new compiler** with **GPU resource** in mind.

- Most **Algorithm** were written for **CPU** → same algorithm logic, **redesign** for **parallelism** philosophy of GPU → **new runtime** with **GPU** resource in mind → **performance increase by order of magnitude.**

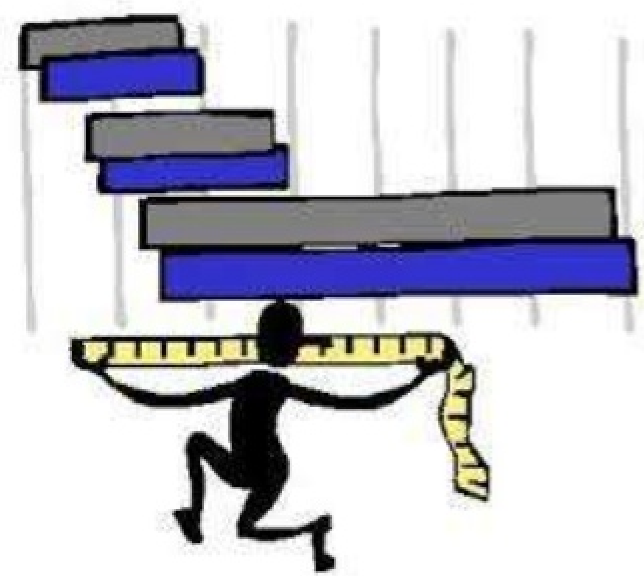
- **VISION: fastest DBMS, cost effective, true scalability.**

True story

- Product: Big Data gpu based DBMS system for analytics. [peta scale]
    - Structed data only, designed for OLAP use cases only
- Technical Constraints
    - How to test SQL syntax?(infinite input) **SQL Coverage** from 0% to 20% to 90% to 100%?.
    - What is the expected **impact of big data**?
    - How to **debug performance** issues?
    - Can you **virtualize**? Should you virtualize?
    - **Running time** - how much it takes to run all the tests?
- Company Challenges
    - **Cost** of Hardware? High end 2U Server
    - **Expertise** ? skillset?
    - **Human resources**: Size of QA Automation team?
    - How do you **manage automation**?
    - What is your **MVP**? (Chicken and Egg)
    - **TIME TO MARKET!!!** The product needs to be released.

The baseline concept

- Requirements
  - CSV , DDL , query
  - Competing DB
  - Your DB
  - Synthetic data used mostly.
  - (real data in peta scale is hard to comeby)
- Steps
  - Insert same data to same table on both DB’s
  - Run same query on both DB’s
  - Compare results on both.
  - If equal → test pass.

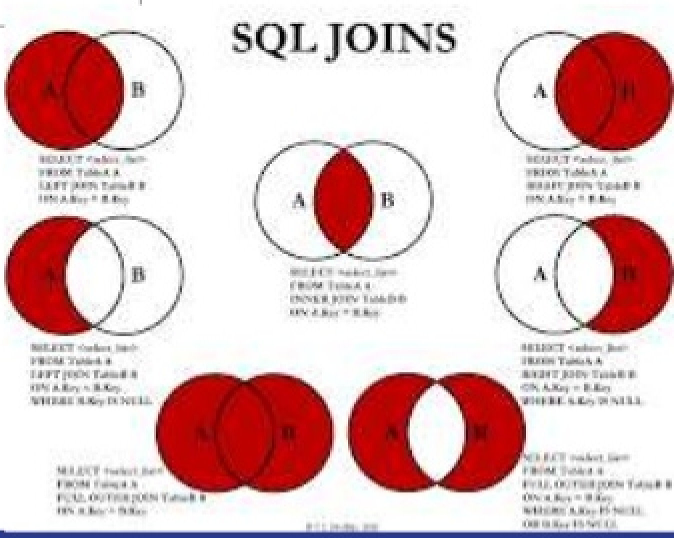


The Solution: SQL testing framework.

- **Lightweight** python test suite per feature
  - **DDL**
  - **Set of Queries**
  - **Json** for data generation requirements
  - Data generation utilities (**genUtil**)
  - **test results** report - in CSV format.
  - **Expected error** mechanism.
  - **Results compare** mechanism
  - **Command line** arguments for advanced testing/config/tuning/views.
- Wrapper test suite
  - group set of test suites by logic - e.g **Daily**
  - Aggregate **reporting** mechanism
- Scheduling ,reporting,monitoring,deployment, alerting mechanism

Testing concept of SQL syntax

	Simple	Aggregation	Joins
Simple	x	x	x
Aggregation	x	x	x
Joins	x	x	x



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Challenges with SQL syntax testing

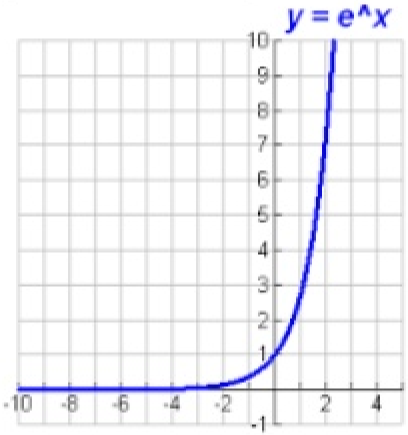
- **(Binary data not supported in the product)**
- **Repetition** of queries.
- Different **data type** names.
- Different **data ranges** per data types.
- Different **accuracy** per data types.
- The competition DB that supports big data is expensive...
- **Accuracy** of results. (different DB's return different accuracy)
- **SQL has some extreme cases** (differ per vendor).
- **Datetime** format.
- **Unsupported** features.
- **Duplication** of testing.
- **Very hard to predict which queries are useful** (negative and positive testing).

Challenges with Small Data testing

- Generating Random data
    - **Reproducible** every time -->**Strict data ranges per test** (length, numeric range, format, accuracy)
    - What is the amount of unique values? Different histogram, different bottlenecks:
        - **Unique values challenges:**
            - Per chunk?
            - Per Column
        - **Non unique values challenges:**
            - **String**?
                - Lengths
                - Compressible?
            - **Numeric**?
                - Overflow?
                - Floating point?

Testing matrix: Data Integrity on Big data.

	Data (no null)	Compression	Null Data	Lookup.	Partitions	JDBC/ODBC
1 row						
1M Rows						
10M rows						
100M Rows						
1B rows						
10B rows						
100B rows						
1 Trillion rows						
10 Trillion rows						



Challenges with Big Data testing.

- **Generating** Time of data → genUtil with json for user input
- **Reproducibility** → genUtil again.
- **Disk space** for testing data → peta scale product → peta scale testing data
- **Results compared** mechanism on big data require a big data solution→ SMJ
- **Network** bottlenecks on a scale out system...
- **ORDER** IS NOT GUARANTEED!

Insights on the fly: User Defined Test Flow

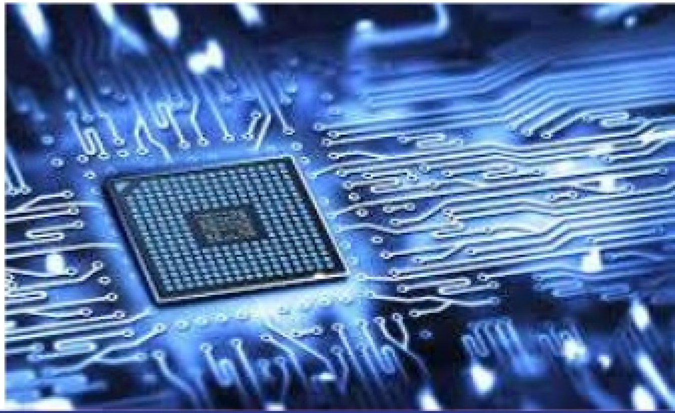
- A **series of queries in a specific order** - **crushed** the system
- **Solution**: Automation infrastructure on a group of queries: **User defined Test Flow.**
- Good for: pre and post condition of , custom testing, system perspective testing. (partitions)

The solution: some Metrics

1. Extra large **Sanity** Testing per commit (!)
2. **Hourly** testing on latest commit (24 X 7)
    a. 800 queries
    b. Zero data
3. **Daily** test:
    a. 400,000 queries
    b. Zero data for 90% , 10% testing upto 1B records.
4. **Weekly** testing:
    a. Big data testing 10B and above.
5. **Monthly** testing:
    a. Full regression per version.

The solution: Hardware perspective

- 50 x **Desktops**: Optiplex 7040, I7 4 cores, 32GB. 1X 4Tb.
- 10 x **High End Servers**: Dell R730/R720: 20 cores, 128GB. 16X 1TB disks. Nvidia K40
- DDN **storage** with 200TB+ GPFS
- **Mellanox** FDR switch for the storage
- **1GB switches** for the desktop compute nodes.



Performance and Maintenance

Performance Challenges: app perspective

- **Architecture bottlenecks [count(*), group by, compression, sort Merge Join]**
- What is **IO pattern**?
  - OLTP VS OLAP.
  - Columnar or Row based?
  - How big is your data?
  - READ % vs WRITE %.
  - Sequential? random?
  - Temporary VS permanent
  - Latency VS. throughput.
  - Multi threaded ? or single thread?
  - Power query? Or production?
  - Cold data? Host data?

Performance Challenges: OPS perspective

- Metrics
  - What is Expected Total **RAM required per Query**?
  - OS **RAM cache , CPU Cache** hits ?
  - **SWAP** ? yes/no how much?
  - OS metrics - **open files**, stack size, realtime priority?
- Theoretical limits?
  - **Disk type selection** -limitation, expected throughput
  - **Raid** controller, RAID type? Configuration? caching?
  - **File system** used? DFS? PFS? Ext4? XFS?
  - Recommended file system filesystem **Block size? File system metadata**?
  - Recommended **File size on disk**?
  - **CPU selection** - number of threads , cache level, hyper threading. Cilicon
  - **Ram Selection - and placement on chassi**
  - **Network - the difference b/w 40Gbit and 25Gbit. NIC Bonding is good for?**
  - PCI Express 3, 16 Langes. PCI Switch.

Maintenance challenges: OPS Perspective

How to Optimize your hardware selection ? (Analytics on your testing data)

a. **Compute intensive**
   i. GPU CORE intensive
   ii. CPUcores intensive
      1. Frequency
      2. Amount of thread for concurrency

b. **IO intensive?**
   i. SSD?
   ii. NearLine sata?
   iii. Partitions?

c. **Hardware uniformity** - use same hardware all over.

d. Get rid of **weak links**

Maintenance challenges: DevOps Perspective

- Lightweight - python code
  - **Advantage** → focus on **generic skill** set (python coding)
  - **Disadvantage** → **reinventing the wheel**
- Fault tolerance - Scale out topology
  - **Cheap desktops** → quick recovery from Image when hardware crushes
    - **Advantage** - >
      - Cheap, low risk, pay as you go.
      - Gets 90 % of the job DONE!
    - **Disadvantage**
      - footprint is hell.
      - Management of desktops - requires creativity
- Rely heavily on automation feature: **reproducibility from scratch**
- **Validation automation** on infrastructure & Deployment mechanism.

Maintenance challenges: DevOps Perspective

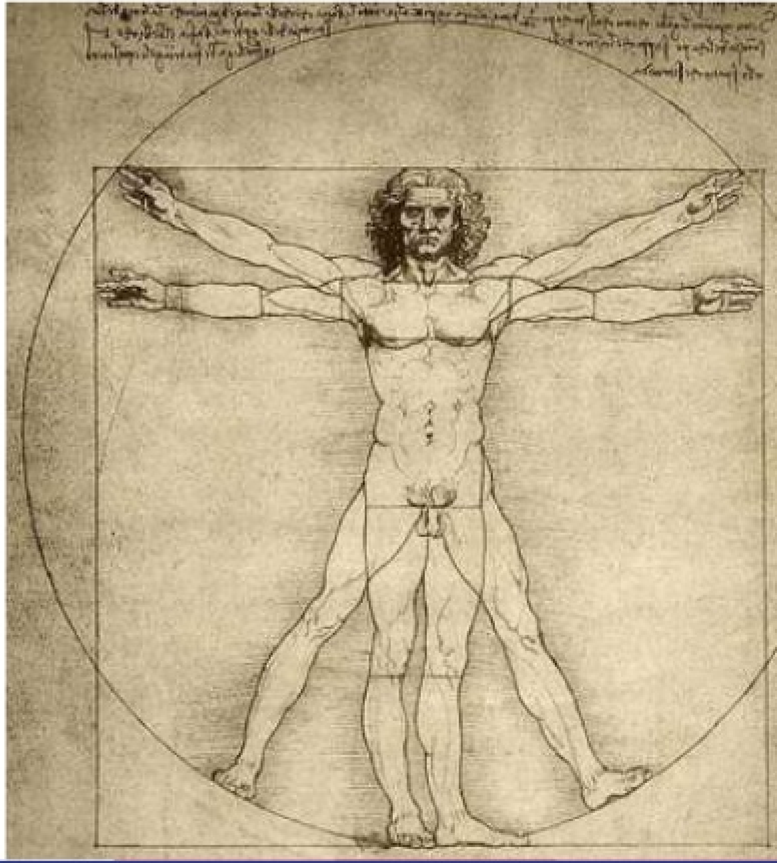
- Infrastructure
  - **Continuous integration**: continuous build, continuous packaging, installer.
  - **Continuous Deployment**: pre flight check, remote machine (on site, private cloud, cloud)
  - **Continuous testing**: Sanity, Hourly, Daily, weekly
- Reporting and Monitoring:
  - **Extensive report server** and analytics on current testing.
  - Monitoring: **Green/Red** and system real time metrics.

Maintenance challenges: Innovation Perspective

1. Innovation on Data generation (using **GPU to generate data**)
2. Innovation on Competitor DB running time (**hashing**)
3. Innovation on Workload (one **dispatcher**, many compute nodes)
4. Innovation on saving testing data /compute time
    a. **Hashing**
    b. **Cacheing**
    c. **Smart Data Creating - no need to create 100TB CSV to generate 100TB test.**
    d. **Logs: Test results were managed on The DBMS itself :)**
5. File system **ZFS Cluster** (utilizing unused disk space, redundancy, deduplication)
6. Nvidia TK1 /**GRID GPU → Footprint!!!**
7. "**Virtualizing GPU**" → Footprint!!!
    a. **Dockers**
    b. **rCUDA**
8. **Amazon p2 gpu instances** did not exist at that time (even so still very expensive)

Building the team Challenge

- Engineering **skills** required
  - Hardware: Servers
  - IO (storage, disks, RAID, IO patten)
  - FileSystems
  - Linux , CLI, CMD, installing, compiling, GIT
  - Basic network understanding
  - High Availability, Redundancy
  - HPC
- **Coding**: python.
- **QA Big** Data mindset
- **DevOps** mindset
- **Automation** Mindset
- Systematic **Innovation** methodologies



Lesson learned (insights)

- Very hard to guarantee **100% QA coverage**
- Quality in a reasonable **cost is a huge challenge**
- Very **easy to miss milestones with QA of big data,**
- each mistake create a **huge ripple effect in timelines**
- Main **costs**:
  - **Employees** : training them.
  - **Running Time**: Coding of innovative algorithms inside testing framework,
  - **Maintenance**: Automation, Monitoring, Analyzing test results, Environments setup
  - **Hardware:** innovative DevOps , innovative OPS, research
- Most of our innovation was spent on:
  - **Doing more tests with less resources**
  - agile **improvement** of backbone
  - **Avoiding big data complexity problems in our testing!**
- **Industry tools** ? great, but not always enough. Know their philosophy...
- Sometimes you need to **get your hands dirty**
- Keep a fine **balance between business and technology**

Take away message:

- Testing simplifications:
  - **Ingress** only
  - **Egress** only
  - Ingress while egress
  - Testing matrix - useful on complex big data systems
- The QA automation team
  - coding/**scripting** skills is a MUST
  - **Engineering** skill is a MUST
  - **DevOps** understanding is a MUST
  - Allocating time for **innovation** is a MUST
  - Allocating time for **cost reduction** is a MUST
- The Automation infrastructure
  - **KISS**
  - **M*V*P**
  - **IS A PRODUCT** by itself. With its own Product manager and Architect

Special Thanks...

- Citi innovation Center
- Asaf - Birenzvieg Big Data & Data Science - Israel meetup
- PayPal Risk and Data solution Group

CITI
INNOVATION
LAB
TLV

Stay in touch...

- Omid Vahdaty in
- +972-54-2384178

HALO ANALYTICS SQREAM TECHNOLOGIES jajah Telefonica