AEROSPIKE

SUMMIT '19

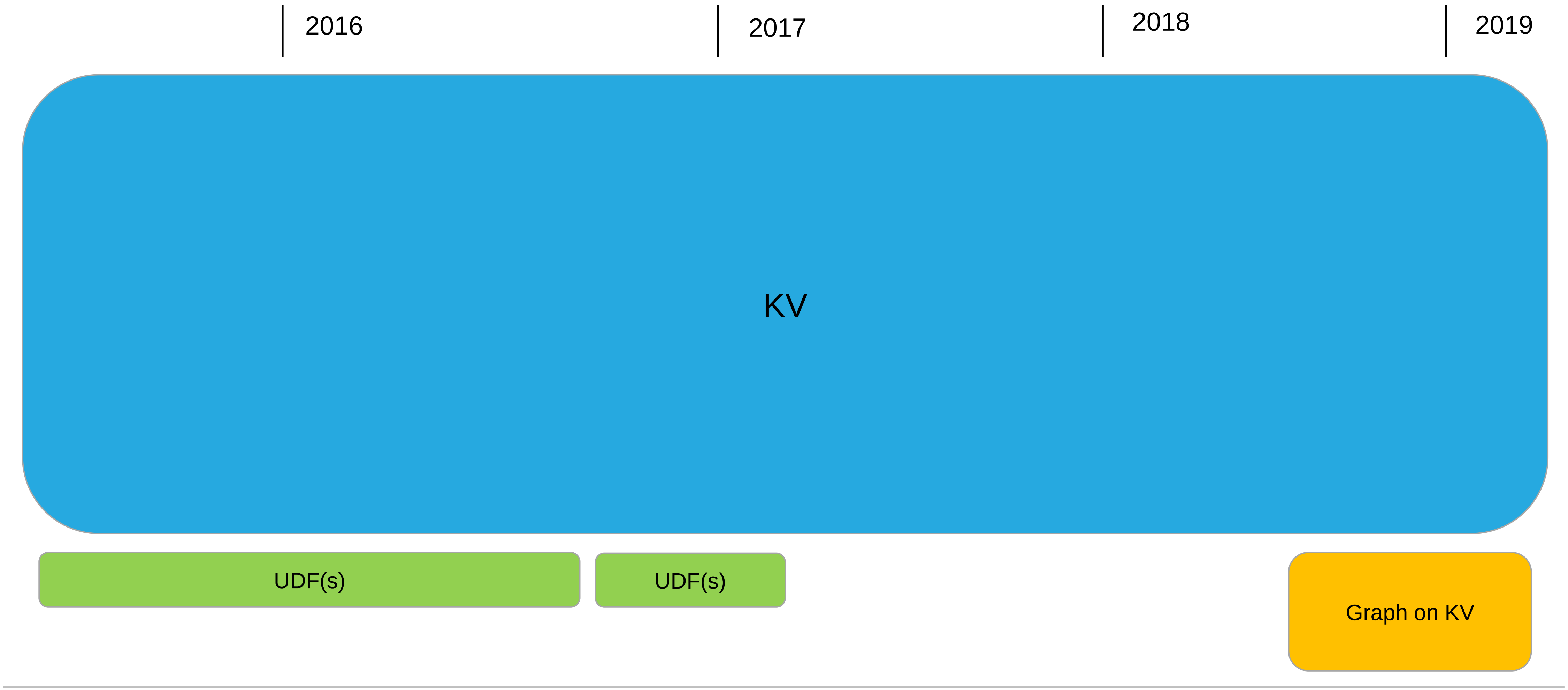
PayPal

Graph on Aerospike

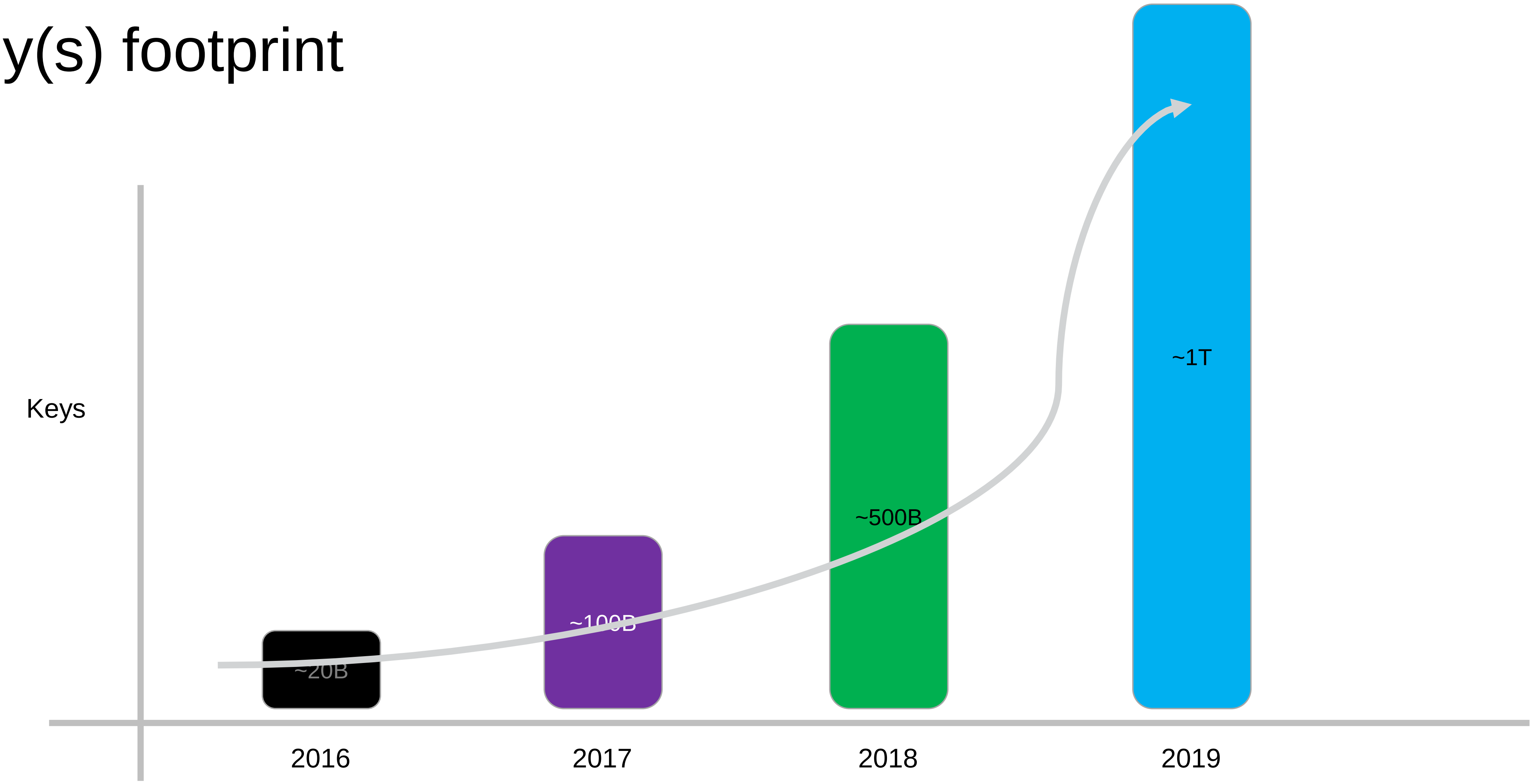
Athreya Gopalakrishna
Senior MTS
Database Engineering Architect
PayPal

Bruce Li
Director, EServ Infra Architect
PayPal

Evolution of Solutions

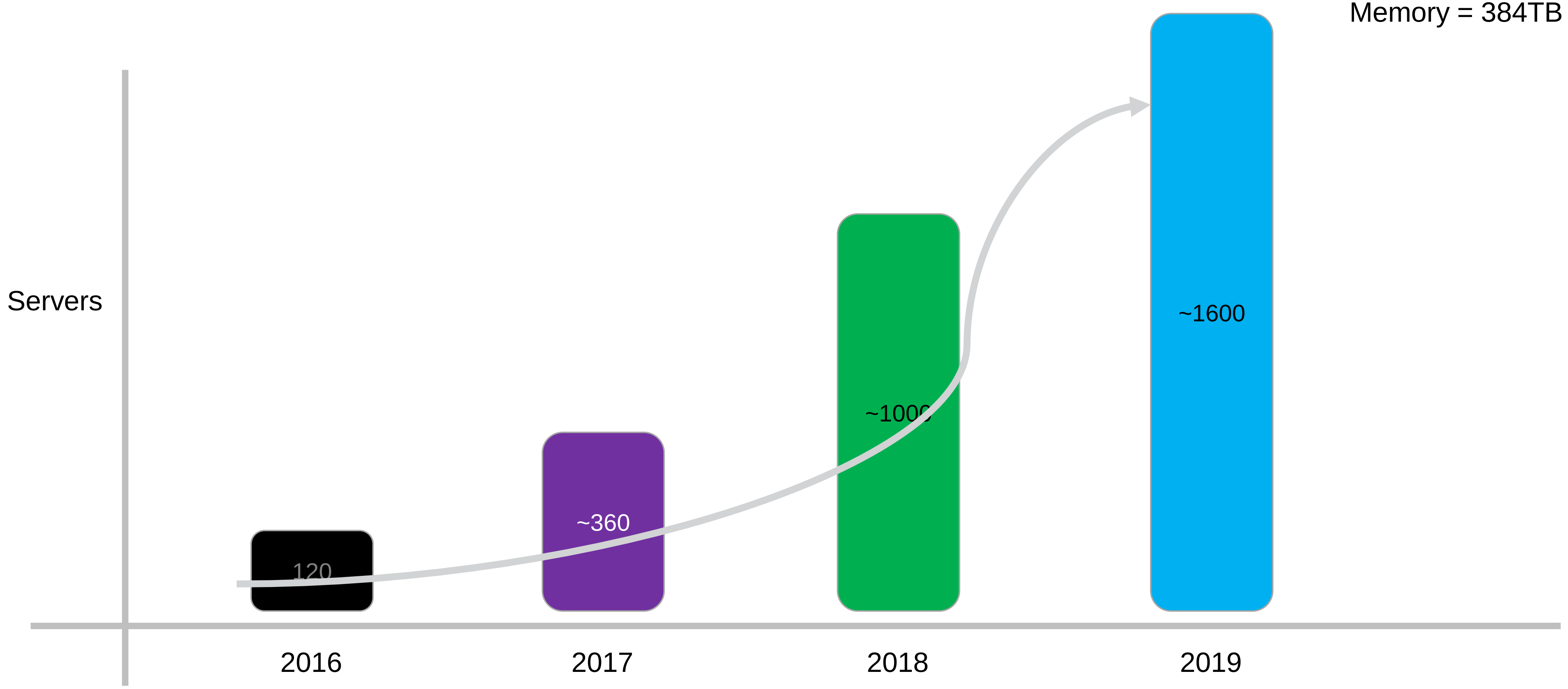


Key(s) footprint



Server footprint

Storage = ~9PB in 2019
Memory = 384TB



Buildout Strategy

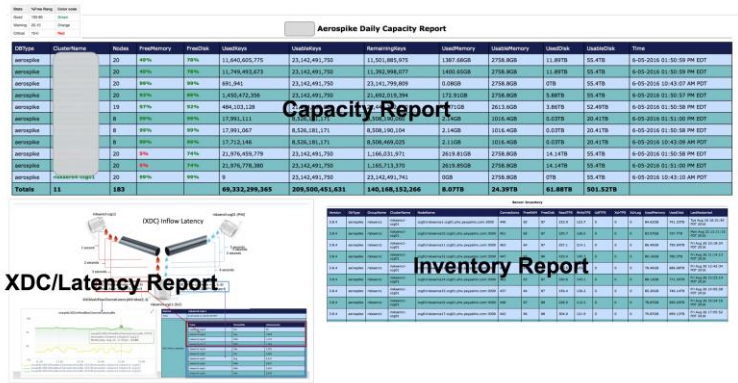
Battery

Goal

Battery Pack

Solutions

Database Reporting



Aerospike Daily Capacity Report

State	%Free Rang	Color code
Good	100-60	Green
Warning	20-11	Orange
Critical	10-0	Red

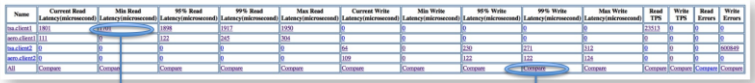
DBType	ClusterName	Nodes	FreeMemory	FreeDisk	UsedKeys	UsableKeys	RemainingKeys	UsedMemory	UsableMemory	UsedDisk	UsableDisk	Time
aerospike	[redacted]	20	49%	78%	11,640,605,775	23,142,491,750	11,501,885,975	1387.68GB	2758.8GB	11.89TB	55.4TB	6-05-2016 01:50:59 PM EDT
aerospike	[redacted]	20	49%	78%	11,749,493,673	23,142,491,750	11,392,998,077	1400.65GB	2758.8GB	11.89TB	55.4TB	6-05-2016 01:50:59 PM EDT
aerospike	[redacted]	20	99%	99%	691,941	23,142,491,750	23,141,799,809	0.08GB	2758.8GB	0TB	55.4TB	6-05-2016 10:43:07 AM PDT
aerospike	[redacted]	20	93%	89%	1,450,472,356	23,142,491,750	21,692,019,394	172.91GB	2758.8GB	5.88TB	55.4TB	6-05-2016 01:50:57 PM EDT
aerospike	[redacted]	19	97%	92%	484,103,128	[illegible]	[illegible]	[illegible]	2613.6GB	3.86TB	52.49TB	6-05-2016 01:50:58 PM EDT
aerospike	[redacted]	8	99%	99%	17,991,111	8,526,181,171	8,508,190,060	2.14GB	1016.4GB	0.03TB	20.41TB	6-05-2016 01:51:00 PM EDT
aerospike	[redacted]	8	99%	99%	17,991,067	8,526,181,171	8,508,190,104	2.14GB	1016.4GB	0.03TB	20.41TB	6-05-2016 01:50:58 PM EDT
aerospike	[redacted]	8	99%	99%	17,712,146	8,526,181,171	8,508,469,025	2.11GB	1016.4GB	0.03TB	20.41TB	6-05-2016 10:43:09 AM PDT
aerospike	[redacted]	20	5%	74%	21,976,459,779	23,142,491,750	1,166,031,971	2619.81GB	2758.8GB	14.14TB	55.4TB	6-05-2016 01:50:58 PM EDT
aerospike	[redacted]	20	5%	74%	21,976,778,380	23,142,491,750	1,165,713,370	2619.85GB	2758.8GB	14.14TB	55.4TB	6-05-2016 01:51:00 PM EDT
aerospike	[redacted]	20	99%	99%	9	23,142,491,750	23,142,491,741	0GB	2758.8GB	0TB	55.4TB	6-05-2016 10:43:10 AM PDT
Totals	11	183			69,332,299,365	209,500,451,631	140,168,152,266	8.07TB	24.39TB	61.88TB	501.52TB	

Capacity Report

XDC/Latency Report

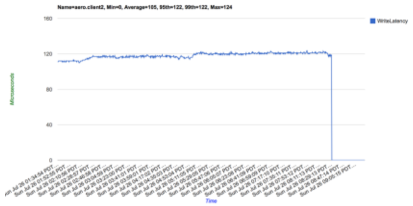
Inventory Report

The Dashboard

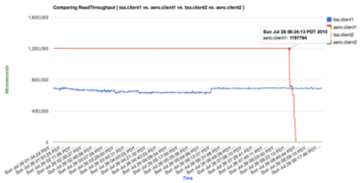


Name	Current Read Latency(microsecond)	Min Read Latency(microsecond)	95% Read Latency(microsecond)	99% Read Latency(microsecond)	Max Read Latency(microsecond)	Current Write Latency(microsecond)	Min Write Latency(microsecond)	95% Write Latency(microsecond)	99% Write Latency(microsecond)	Max Write Latency(microsecond)	Read TPS	Write TPS	Read Errors	Write Errors
tsa.client1	1801	1701	1898	1917	1950	0	0	0	0	0	23513	0	0	0
aero.client1	111	0	122	245	304	0	0	0	0	0	0	0	0	0
tsa.client2	0	0	0	0	0	64	0	230	271	312	0	0	0	600849
aero.client2	0	0	0	0	0	109	0	122	122	124	0	0	0	0
All	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare	Compare

Metrics Visualization



Comparison Visualization

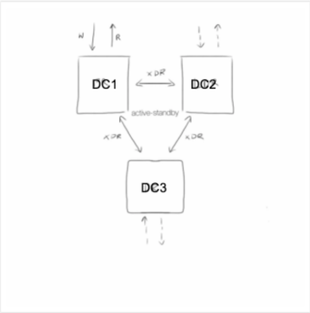


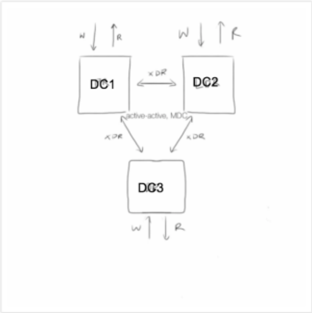
Manticore

Sniper

Ansible

Auto-failovers



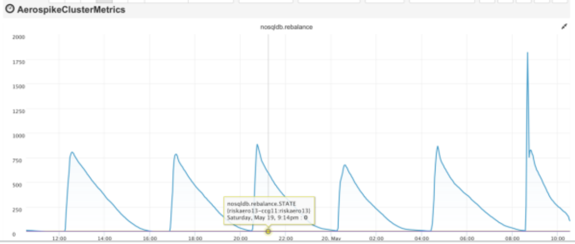


Database Lifecycle Automation

Ansible Automation API(s) – Programmatic OR Human interface

- prepare_new_node
- wipe_out_server
- create_cluster
- reconfigure_database
- add_node
- change_password
- reset_cluster_name
- apply_os_patch_rolling

- backup
- restore
- prepare_tools_node
- remove_node
- validate_cluster
- switch_paxos_protocol
- turn_off_clear_port
- apply_os_patch_single_node

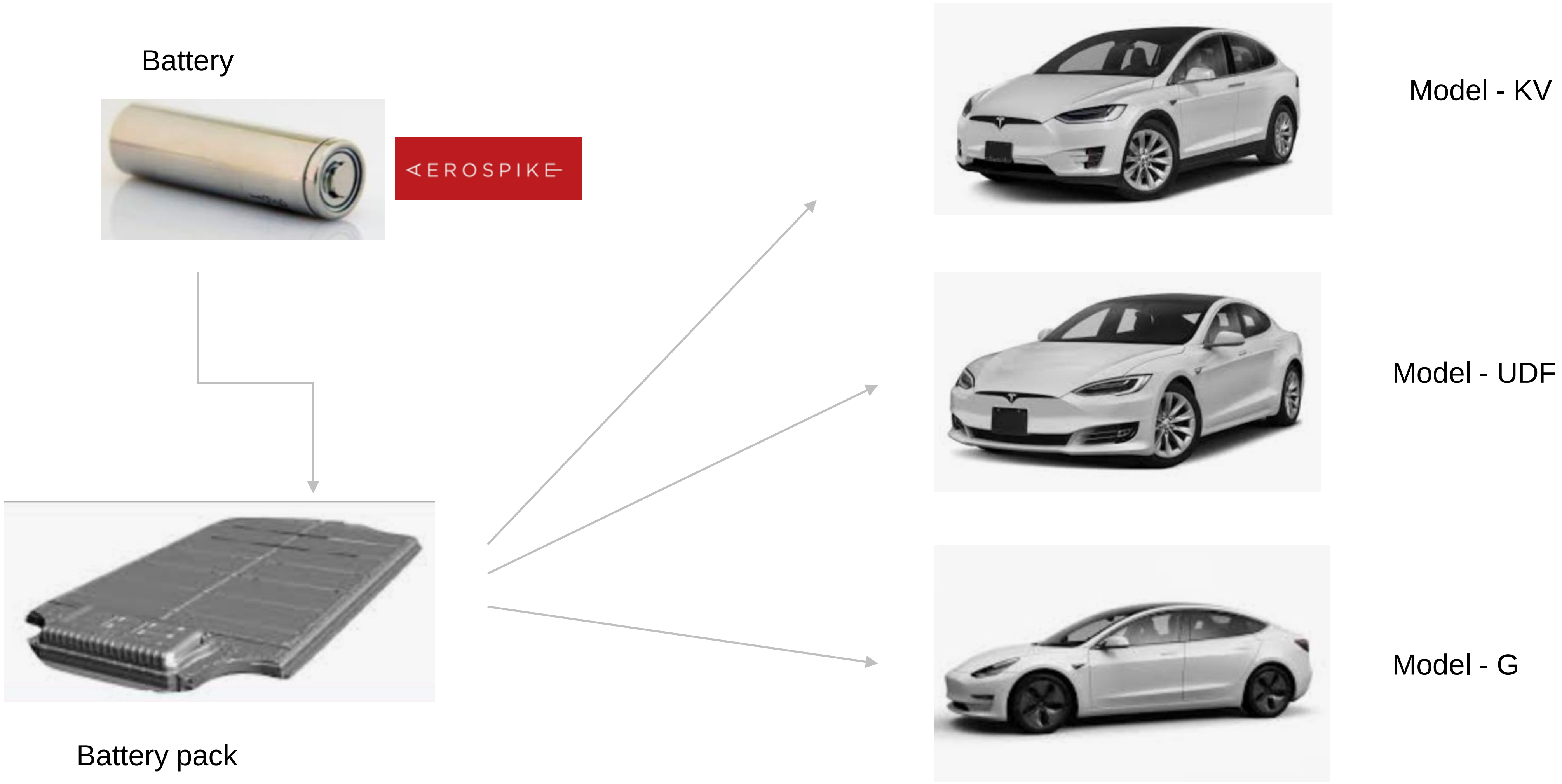


1200+ serve

62 clusters

4 Datacen

Deployment Strategy



Battery

Battery pack

Model - KV

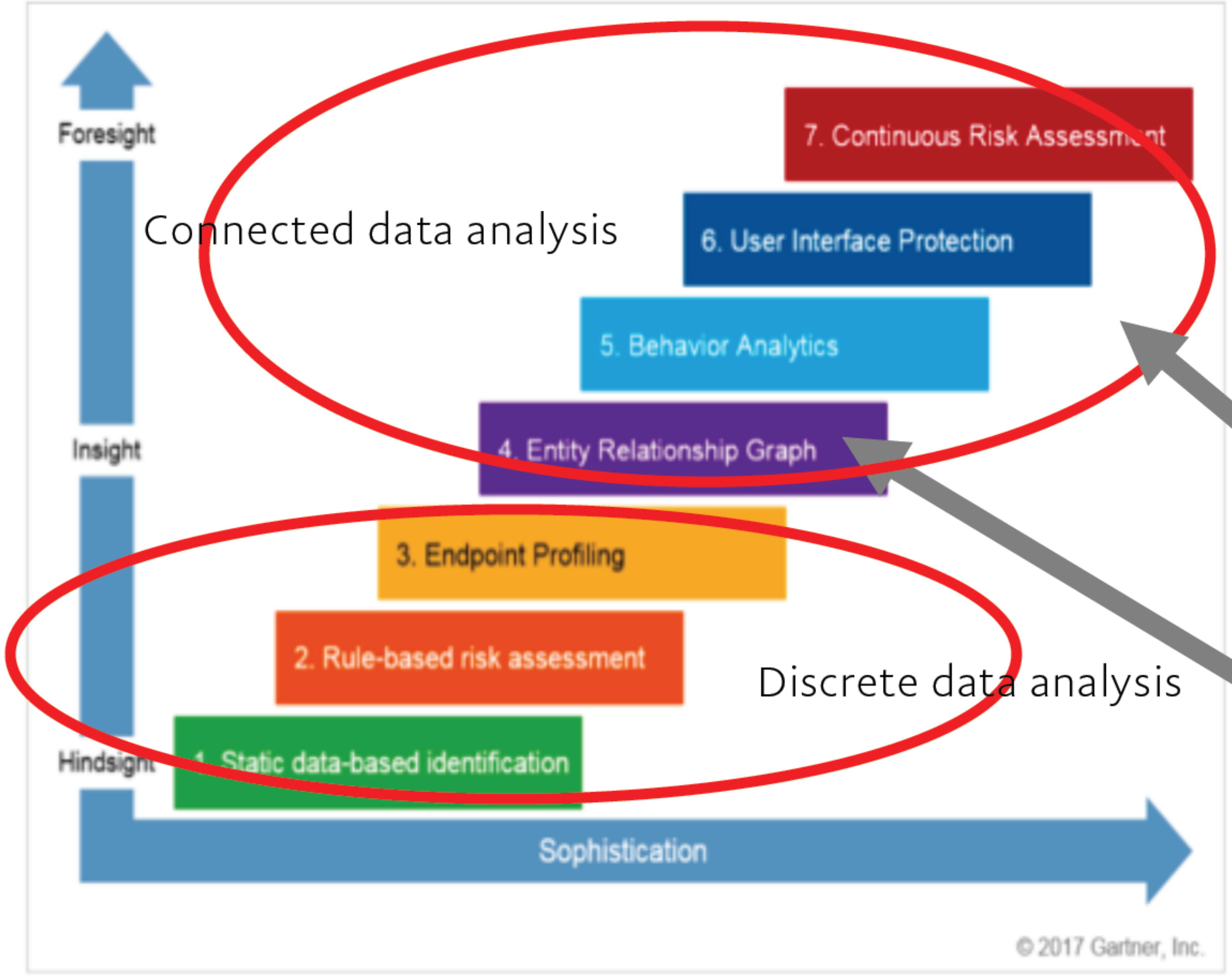
Model - UDF

Model - G

Model – G = Graph

Why Real-time Large Scale Graph Database

Fraud Prevention Capability Levels


Foresight
Insight
Hindsight
7. Continuous Risk Assessment
6. User Interface Protection
5. Behavior Analytics
4. Entity Relationship Graph
3. Endpoint Profiling
2. Rule-based risk assessment
1. Static data-based identification
Sophistication
Connected data analysis
Discrete data analysis
© 2017 Gartner, Inc.

Source: Gartner (July 2017)

By 2022, 60% of organizations will be at capability **Level 5** or above in the Gartner Fraud Detection Capability Model, up from less than 30% today

Graph analytics

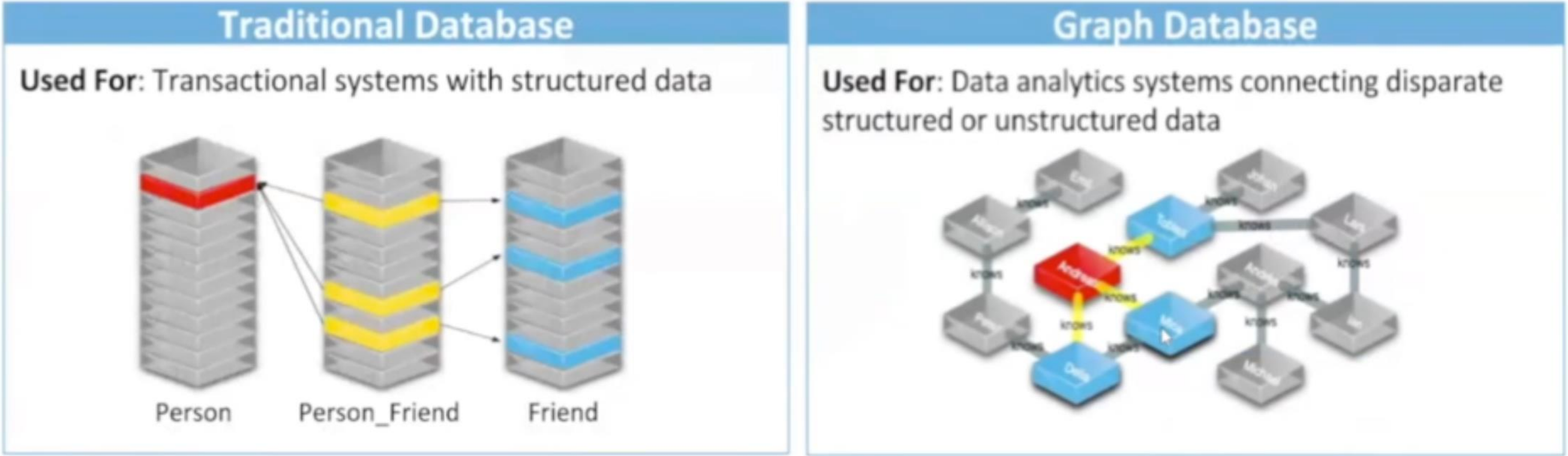
- Leverage models to determine the "connectedness" across data points to create data nodes/linkages/communities and their demarcation points. Nodes are connected explicitly or implicitly, indicating the levels of influence, strength, frequency, and quality of interaction, and probability.
- Effective in uncovering any hidden ultimate beneficiaries or dishonest nodes in the graph.

Graph analysis

- To analyze the metadata of an account, entity or transaction, etc., as well as the relationships, linkages between data points, to create a risk assessment of said data points.
- Identify risk through associations or links to negative lists, high-velocity activity or morphing.

Graph Database vs Traditional Database

Graphs are suited for use cases where connections between data points are just as important as data points themselves



Traditional Database

Used For: Transactional systems with structured data

Graph Database

Used For: Data analytics systems connecting disparate structured or unstructured data

- SQL Join Hell
- Difficult to express native graph query logic
- Difficult to support flexible and fast-changed schema

- Flatten the view and simplifies the queries
- Optimized for graph queries/computations
- Flexible data types

Needs for Real-time Graph Database

- The old way before graph database
  - ❑ Use offline data processing to generate KV data sets then push them to online KV storages
  - ❑ Online applications have to write code based on KVs to implement different join/linking logic

  Problems:

  - Long delay of data refreshes
  - Difficult for offline data processing without graph database support
  - Not a generic/flexible solution, long TTM
  - Difficult for graph data governance
- Solution of normal graph database
  - ❑ Leverage graph database for offline/NRT data processing and convert data to KVs for online integration
  - ❑ Online applications have to write code based on KVs to implement different join/linking logic because performance of normal graph databases can't meet tight online SLA requirement

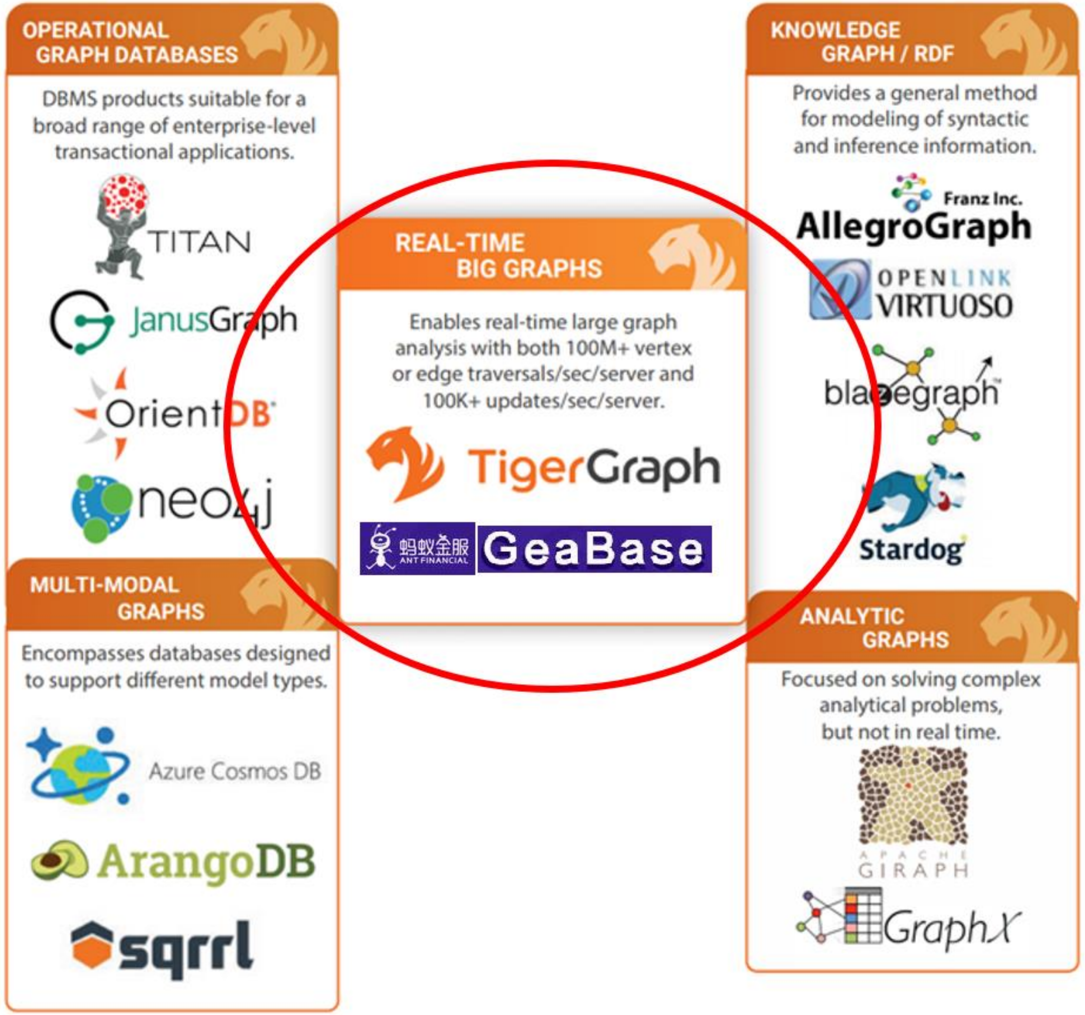
  Problems:

  - High total cost of ownership
  - Not a generic/flexible solution for online application, long TTM
  - Difficult for graph data governance
- Solution of real-time graph database
  - ❑ Leverage real-time graph database for offline/NRT/RT data processing and directly persist data to the graph database
  - ❑ Online applications directly integrate with real-time graph database

  Benefits:

  - Lower total cost of ownership
  - Low TTM for online integration
  - Enable very fast data refreshes
  - Enable centralized graph data governance

Graph Database Landscape & Selection



OPERATIONAL GRAPH DATABASES

DBMS products suitable for a broad range of enterprise-level transactional applications.

MULTI-MODAL GRAPHS

Encompasses databases designed to support different model types.

REAL-TIME BIG GRAPHS

Enables real-time large graph analysis with both 100M+ vertex or edge traversals/sec/server and 100K+ updates/sec/server.

KNOWLEDGE GRAPH / RDF

Provides a general method for modeling of syntactic and inference information.

ANALYTIC GRAPHS

Focused on solving complex analytical problems, but not in real time.

https://cdn2.hubspot.net/hubfs/4114546/Collateral/TigerGraph-Graph-Database-Landscape-Infographic.pdf

Why Aerospike as Real-time Graph Storage

- Super fast with persistence support
- Share-nothing architecture, highly scalable
- Highly available
- Native XDR
- Flexible data types
- Tabular support
- Async non-blocking IO support for Java client (Netty)

Already proven in PayPal

Comparisons of TigerGraph, JanusGraph & Milkyway

	TigerGraph	JanusGraph (+ Syclla)	Milkyway (+Aerospike)
Category	Real-time (**milliseconds - seconds**)	Near-real-time (**seconds - minutes**)	Real-time (**milliseconds**)
License	Commercial only	Apache 2 for JanusGraph, additional for Syclla	Apache 2 for TinkerPop/Gremlin and Aerospike
Open Source	No	Yes	Not now
Implemented Language	C++	Java	Java
Update throughput	100K+ updates/sec/node	30K-100K+ updates/sec/node	100K+ updates/sec/node
Query throughput (2-hop query) (*adjusted based on the known benchmarks)	~1K QPS/node (see ①)	n * 100 QPS/node	10K+ QPS/node
Query latency (2-hop query)	Milliseconds level (see ①)	Seconds level	Avg latency ~30ms
Known usage in production	"TigerGraph and GSQL are used in production to support multi-hop queries spanning 3-10+ hops, all in a graph with 100+ billion nodes and nearly a trillion relationships" (see ②)	Netflix: here Huawei: here …	PayPal in online production: 5+ billions of vertices in Q1, 2019 10+ billions of vertices in Q2, 2019 …
Gremlin query support	No (reasons see ②)	Yes	Yes
ACID transaction	Supported	Not supported on Cassandra or HBase or Scylla (see ③ & ④)	Not supported for now
Native MDC support	?	No?	Yes

① https://globenewswire.com/news-release/2018/06/12/1520391/0/en/TigerGraph-Announces-Free-Developer-Edition-of-the-World-s-Fastest-Graph-Database.html
② https://www.tigergraph.com/2018/05/22/its-time-for-a-modern-graph-query-language/
③ https://docs.janusgraph.org/latest/tx.html
④ https://github.com/JanusGraph/janusgraph/issues/926

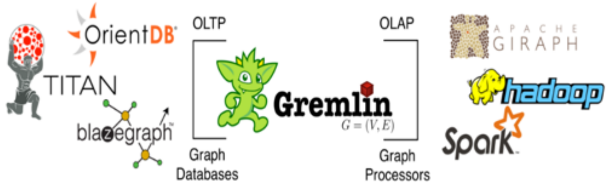
Milkyway – Large Scale Graph Database Solution

High-Level Design Principles

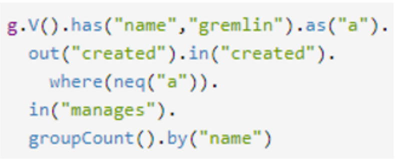
1. Centralized configuration & metadata-driven solution
   - Schema enforced
2. Linear scalability
   - Can support trillion+ of vertices & edges
3. Eventually consistency
   - Idempotent update for each type of write operation
   - Storage issues can always be fixed by bin-log replay
   - No transaction & ACID guaranteed

Graph Query Languages

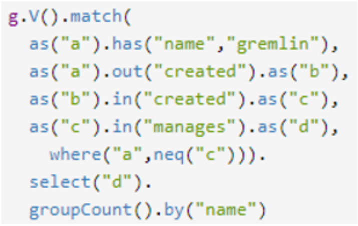
★ Gremlin – from Apache TinkerPop (Apache **top-level** project)



- Support OLTP and OLAP traversals
- Support imperative and declarative transversals
- Widely supported by different graph databases
- **Winning graph query language**



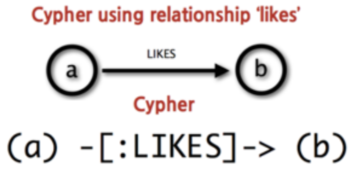
```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
    where(neq("a")).
  in("manages").
  groupCount().by("name")
```



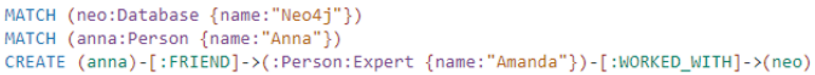
```
g.V().match(
  as("a").has("name","gremlin"),
  as("a").out("created").as("b"),
  as("b").in("created").as("c"),
  as("c").in("manages").as("d"),
    where("a",neq("c"))).
  select("d").
  groupCount().by("name")
```

http://tinkerpop.apache.org/

OpenCypher (Neo4j's query language)



- Declarative query language
- Powerful query capabilities, learning curve is not easy
- Open source, but relatively small community



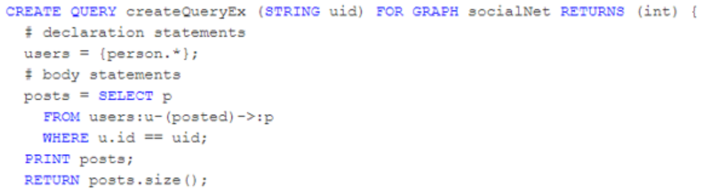
```
MATCH (neo:Database {name:"Neo4j"})
MATCH (anna:Person {name:"Anna"})
CREATE (anna)-[:FRIEND]->(:Person:Expert {name:"Amanda"})-[:WORKED_WITH]->(neo)
```

https://neo4j.com/developer/cypher-query-language/

GSQL (TigerGraph's query language)

TigerGraph GSQL

- Declarative query (or programing) language
- Very powerful, but not user friendly
- Highly optimized for MPP & distributed compute
- Commercial version only

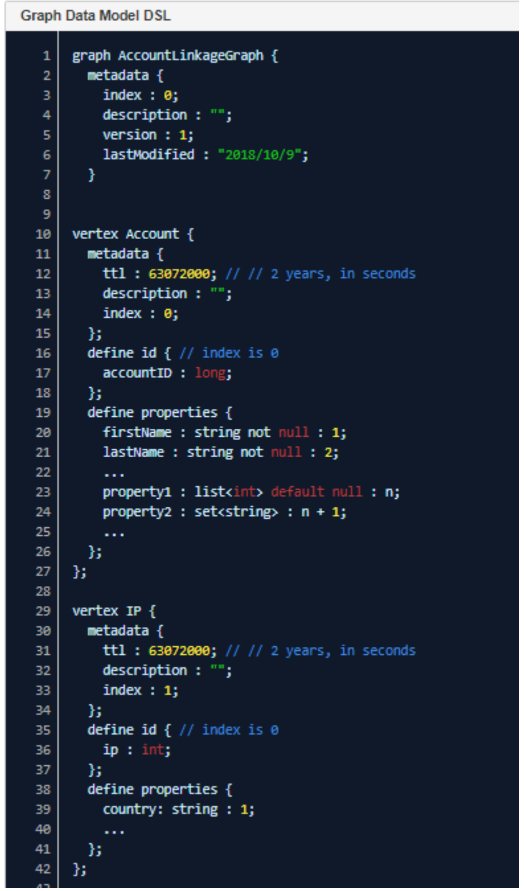


```
CREATE QUERY createQueryEx (STRING uid) FOR GRAPH socialNet RETURNS (int) {
  # declaration statements
  users = {person.*};
  # body statements
  posts = SELECT p
    FROM users:u-(posted)->:p
    WHERE u.id == uid;
  PRINT posts;
  RETURN posts.size();
}
```

https://doc.tigergraph.com/GSQL-Language-Reference-Part-2---Querying.html

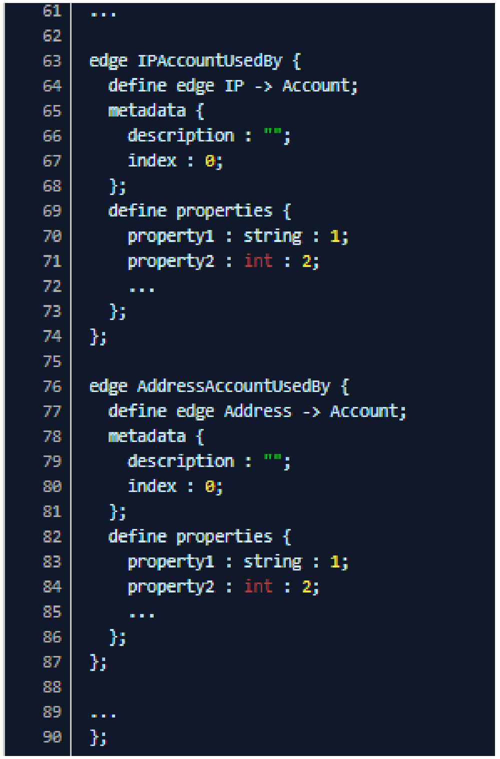
Milkyway Graph Schema

Schema example:



Graph Data Model DSL

```
graph AccountLinkageGraph {
  metadata {
    index : 0;
    description : "";
    version : 1;
    lastModified : "2018/10/9";
  }


vertex Account {
  metadata {
    ttl : 63072000; // // 2 years, in seconds
    description : "";
    index : 0;
  };
  define id { // index is 0
    accountID : long;
  };
  define properties {
    firstName : string not null : 1;
    lastName : string not null : 2;
    ...
    property1 : list<int> default null : n;
    property2 : set<string> : n + 1;
    ...
  };
};

vertex IP {
  metadata {
    ttl : 63072000; // // 2 years, in seconds
    description : "";
    index : 1;
  };
  define id { // index is 0
    ip : int;
  };
  define properties {
    country: string : 1;
    ...
  };
};
```



```
...

edge IPAccountUsedBy {
  define edge IP -> Account;
  metadata {
    description : "";
    index : 0;
  };
  define properties {
    property1 : string : 1;
    property2 : int : 2;
    ...
  };
};

edge AddressAccountUsedBy {
  define edge Address -> Account;
  metadata {
    description : "";
    index : 0;
  };
  define properties {
    property1 : string : 1;
    property2 : int : 2;
    ...
  };
};

...
};
```

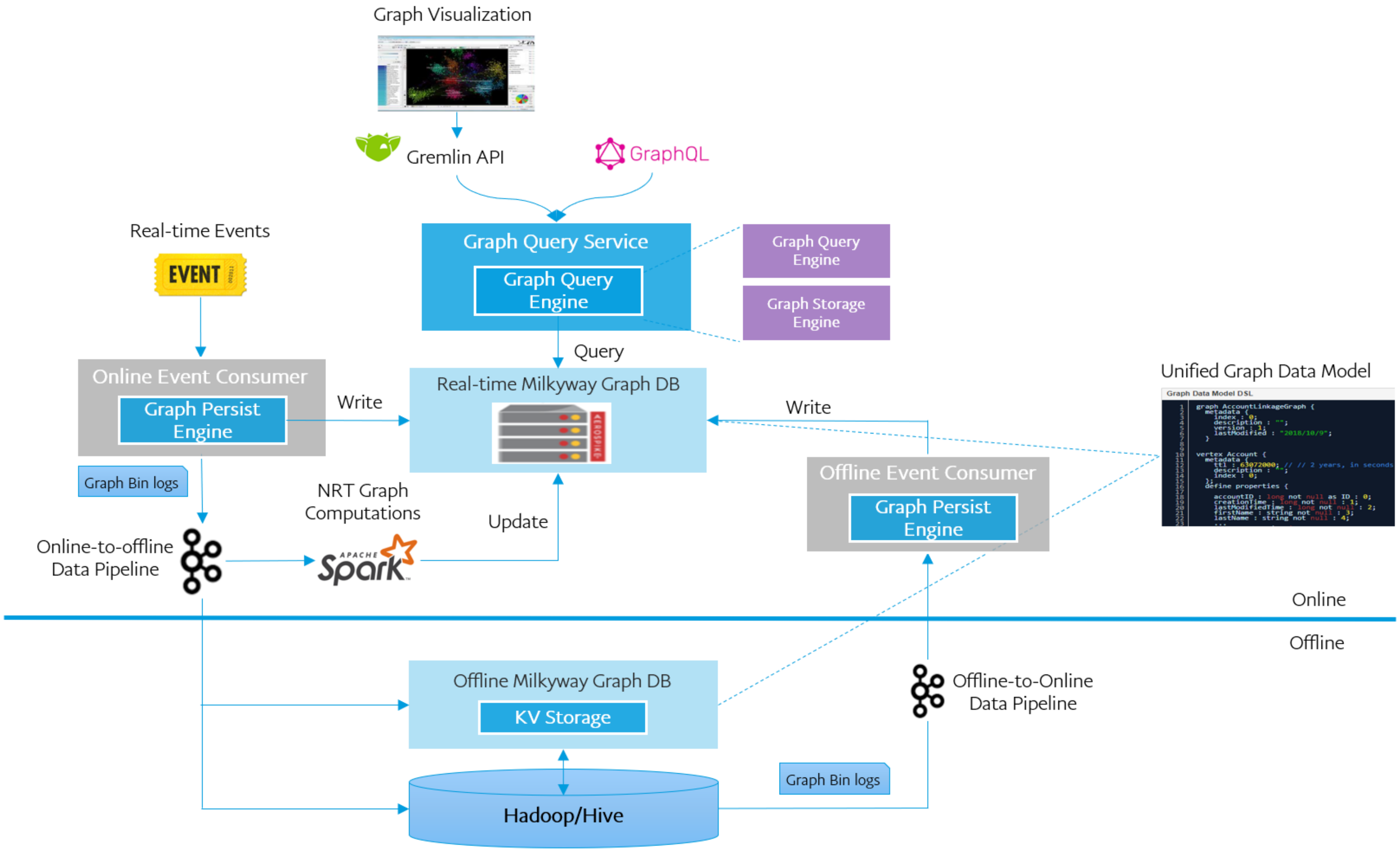
Property data types:

Name	Description
string	character sequence
char	individual character
boolean	true or false
byte	byte value
short	short value
integer	integer value
long	long value
float	4 byte floating point number
double	8 byte floating point number
list	list type
map	map type

Default property metadata:

- creation-time : long
- last-modified-time : long

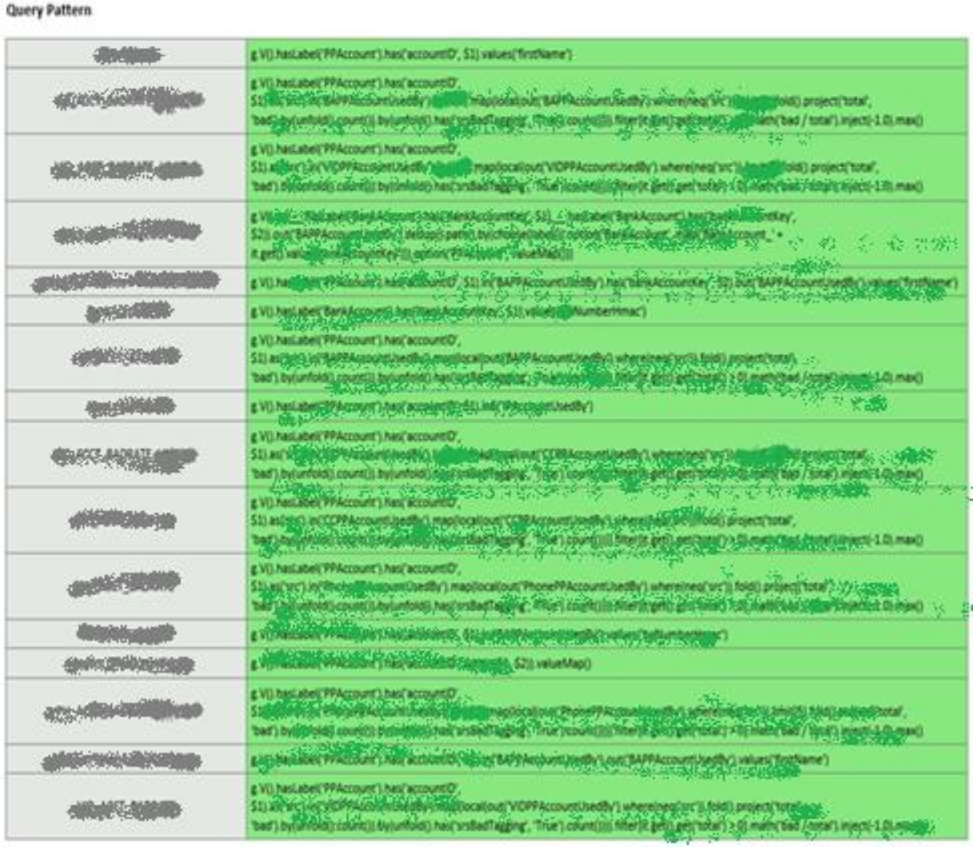
Overall Architecture Design



Graph Data Model DSL

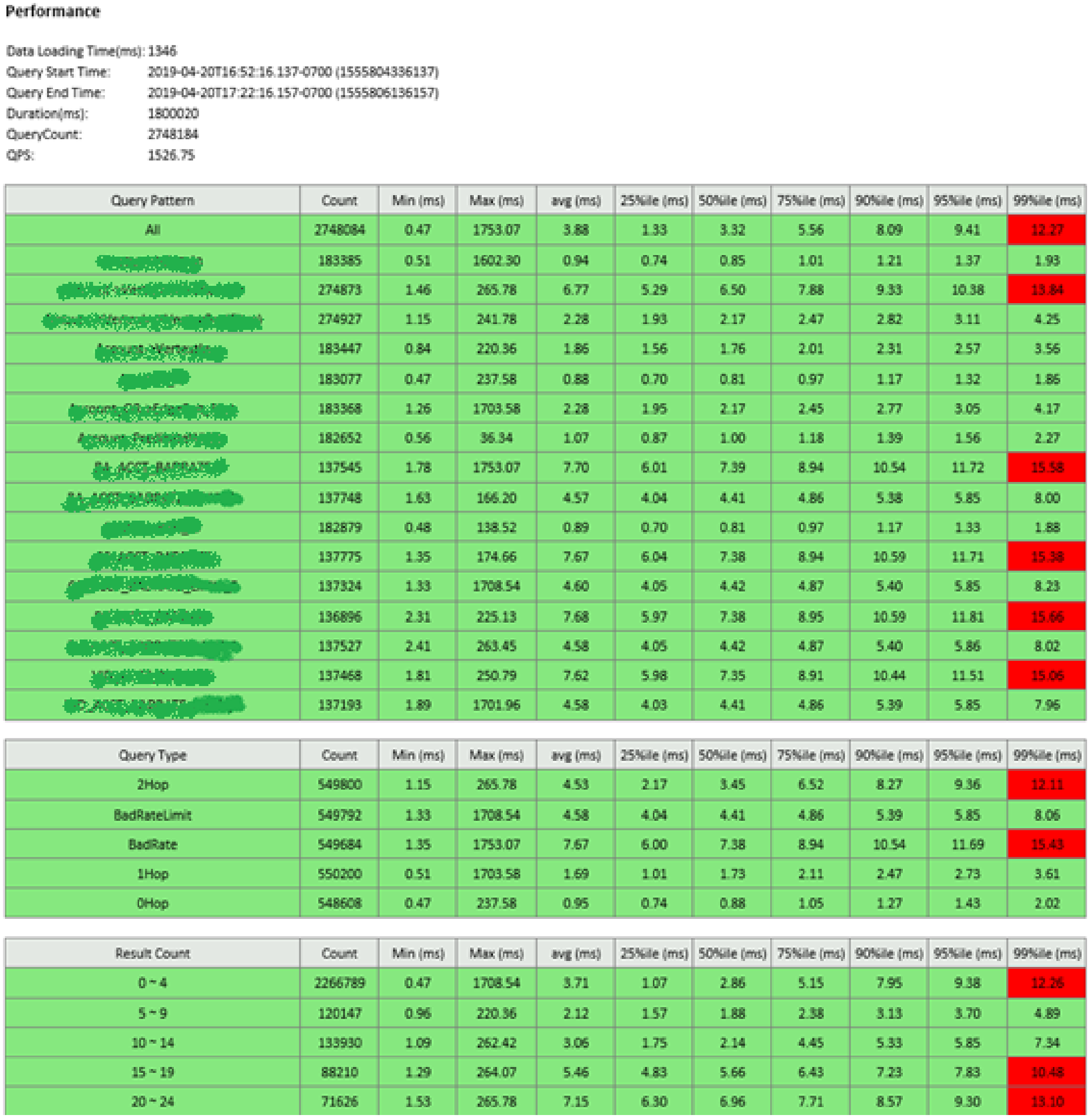
```
graph AccountLinkageGraph {
  metadata {
    index : 0;
    description : "";
    version : 1;
    lastModified : "2018/10/9";
  }

vertex Account {
  metadata {
    ttl : 63072000; // // 2 years, in seconds
    description : "";
    index : 0;
  };
  define properties {

    accountID : long not null as ID : 0;
    creationTime : long not null : 1;
    lastModifiedTime : long not null : 2;
    firstName : string not null : 3;
    lastName : string not null : 4;
    ...
```

Gremlin Query Benchmark in GCP



Query Pattern

[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).values('firstName')
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1)[redacted]('BAPPAccountUsedBy')[redacted].map(local(out('BAPPAccountUsedBy').where(neq('src'))[redacted]fold()).project('total', 'bad')[redacted]by(unfold()).count()).by(unfold().has('srsBadTagging'[redacted]math('bad / total').inject(-1.0).max()
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).a[redacted]in('VIDPPAccountUsedBy')[redacted].map(local(out('VIDPPAccountUsedBy').where(neq('src'))[redacted]fold()).project('total', 'bad').[redacted]
[redacted]	g.V[redacted] $2)).out('BAPPAccount[redacted] it.get().valu[redacted]
[redacted]	g.V().ha[redacted]
[redacted]	g.V().hasLabe[redacted]Number[redacted]
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).as([redacted] 'bad').by(unfold([redacted]).max()
[redacted]	g.V().has[redacted]
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).as('[redacted].project('total', 'bad').by(unfold()).count()).by(unfold()[redacted]srsBadTagging', 'True').count[redacted]inject(-1.0).max()
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).as([redacted]).in('CCPPAccountUsedBy').map(local(out('CCPPAccountUsedBy').[redacted].project('total', 'bad')[redacted]'True').count())).filter{it.get().[redacted]inject(-1.0).max()
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).as('src').in('Phone[redacted]AccountUsedBy').map(local(out('PhonePPAccountUsedBy').where(neq('src')).fold()).project('total', 'bad')[redacted]
[redacted]	g.V[redacted]
[redacted]	g.V[redacted], $2)).valueMap()
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $[redacted]map(local(out('PhonePPAccount[redacted]project('total', 'bad').by[redacted]'True').count[redacted]max()
[redacted]	g.[redacted]('BAPPAccountUsedBy').out('BAPPAccountUsedBy').values('firstName')
[redacted]	g.V().hasLabel('PPAccount').has('accountID', $1).as('src').in('VIDPPAccount[redacted]local(out('VIDPPAccountUsedBy').where(neq('src')).fold()).project('total[redacted] 'bad').by(unfold()).count()).by(unfold().has('srsBadTagging', 'True').count())).filter{it.get().get('total') > 0}.math('bad / total').inject(-1.0[redacted]



Performance

Data Loading Time(ms): 1346
Query Start Time: 2019-04-20T16:52:16.137-0700 (1555804336137)
Query End Time: 2019-04-20T17:22:16.157-0700 (1555806136157)
Duration(ms): 1800020
QueryCount: 2748184
QPS: 1526.75

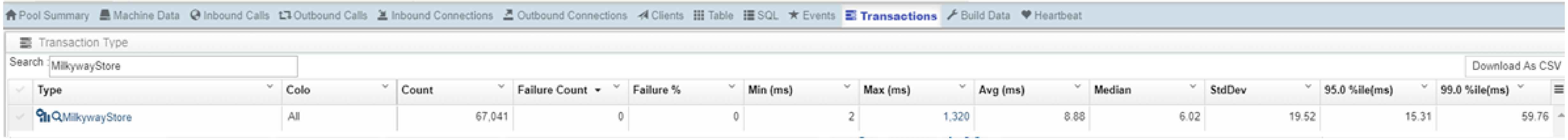
Query Pattern	Count	Min (ms)	Max (ms)	avg (ms)	25%ile (ms)	50%ile (ms)	75%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)
All	2748084	0.47	1753.07	3.88	1.33	3.32	5.56	8.09	9.41	12.27
[redacted]	183385	0.51	1602.30	0.94	0.74	0.85	1.01	1.21	1.37	1.93
[redacted]	274873	1.46	265.78	6.77	5.29	6.50	7.88	9.33	10.38	13.84
[redacted]	274927	1.15	241.78	2.28	1.93	2.17	2.47	2.82	3.11	4.25
[redacted]	183447	0.84	220.36	1.86	1.56	1.76	2.01	2.31	2.57	3.56
[redacted]	183077	0.47	237.58	0.88	0.70	0.81	0.97	1.17	1.32	1.86
[redacted]	183368	1.26	1703.58	2.28	1.95	2.17	2.45	2.77	3.05	4.17
[redacted]	182652	0.56	36.34	1.07	0.87	1.00	1.18	1.39	1.56	2.27
[redacted]	137545	1.78	1753.07	7.70	6.01	7.39	8.94	10.54	11.72	15.58
[redacted]	137748	1.63	166.20	4.57	4.04	4.41	4.86	5.38	5.85	8.00
[redacted]	182879	0.48	138.52	0.89	0.70	0.81	0.97	1.17	1.33	1.88
[redacted]	137775	1.35	174.66	7.67	6.04	7.38	8.94	10.59	11.71	15.38
[redacted]	137324	1.33	1708.54	4.60	4.05	4.42	4.87	5.40	5.85	8.23
[redacted]	136896	2.31	225.13	7.68	5.97	7.38	8.95	10.59	11.81	15.66
[redacted]	137527	2.41	263.45	4.58	4.05	4.42	4.87	5.40	5.86	8.02
[redacted]	137468	1.81	250.79	7.62	5.98	7.35	8.91	10.44	11.51	15.06
[redacted]	137193	1.89	1701.96	4.58	4.03	4.41	4.86	5.39	5.85	7.96

Query Type	Count	Min (ms)	Max (ms)	avg (ms)	25%ile (ms)	50%ile (ms)	75%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)
2Hop	549800	1.15	265.78	4.53	2.17	3.45	6.52	8.27	9.36	12.11
BadRateLimit	549792	1.33	1708.54	4.58	4.04	4.41	4.86	5.39	5.85	8.06
BadRate	549684	1.35	1753.07	7.67	6.00	7.38	8.94	10.54	11.69	15.43
1Hop	550200	0.51	1703.58	1.69	1.01	1.73	2.11	2.47	2.73	3.61
0Hop	548608	0.47	237.58	0.95	0.74	0.88	1.05	1.27	1.43	2.02

Result Count	Count	Min (ms)	Max (ms)	avg (ms)	25%ile (ms)	50%ile (ms)	75%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)
0 ~ 4	2266789	0.47	1708.54	3.71	1.07	2.86	5.15	7.95	9.38	12.26
5 ~ 9	120147	0.96	220.36	2.12	1.57	1.88	2.38	3.13	3.70	4.89
10 ~ 14	133930	1.09	262.42	3.06	1.75	2.14	4.45	5.33	5.85	7.34
15 ~ 19	88210	1.29	264.07	5.46	4.83	5.66	6.43	7.23	7.83	10.48
20 ~ 24	71626	1.53	265.78	7.15	6.30	6.96	7.71	8.57	9.30	13.10

Gremlin Performance in Production

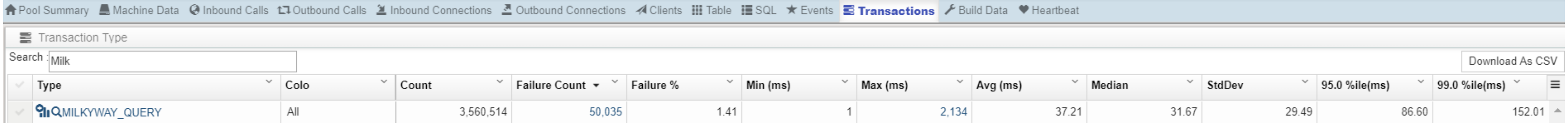
Write performance:



Search : MilkywayStore

Type	Colo	Count	Failure Count	Failure %	Min (ms)	Max (ms)	Avg (ms)	Median	StdDev	95.0 %ile(ms)	99.0 %ile(ms)
MilkywayStore	All	67,041	0	0	2	1,320	8.88	6.02	19.52	15.31	59.76

Read performance:



Search : Milk

Type	Colo	Count	Failure Count	Failure %	Min (ms)	Max (ms)	Avg (ms)	Median	StdDev	95.0 %ile(ms)	99.0 %ile(ms)
MILKYWAY_QUERY	All	3,560,514	50,035	1.41	1	2,134	37.21	31.67	29.49	86.60	152.01

Q & A