High Performance Transaction Systems, 2011, Asilomar, CA

Flexible OLTP Data models in the future

Jags Ramnarayan

Disclaimer:
Any positions expressed here are my own and do not necessarily reflect the positions of my employer VMWare.

vmware

Agenda

- Perspective on some trends
- Basic concepts in VMWare GemFire/SQLFire
- Beyond key based partitioning
- Beyond the SQL Data model

Trends, Observations

- **High demand for low/predicable latency, handle huge load spikes, in-memory on commodity, big data**

- **Input is streaming in nature**
  - High, bursty rates ... structured and unstructured
  - continuous correlations and derived events

- **Increasingly data is bi-temporal in nature**
  - very high ingest rates that tend to be bursty
  - optimizations for inserts and mass migration of historical data to data warehouse.
  - occasional joins across in-memory and data warehouse

Trends, Observations

- **DB schema rapidly evolving**
  - Services are added/changed every week... DB model cannot be rigid
  - programmer drives the change
  - DBA only for operational support?

- **DB Instance is ACID but nothing ACID across the enterprise**
  - many silos and data duplicated across independent databases
  - Cleansing, de-duplication is fact of life and will never go away

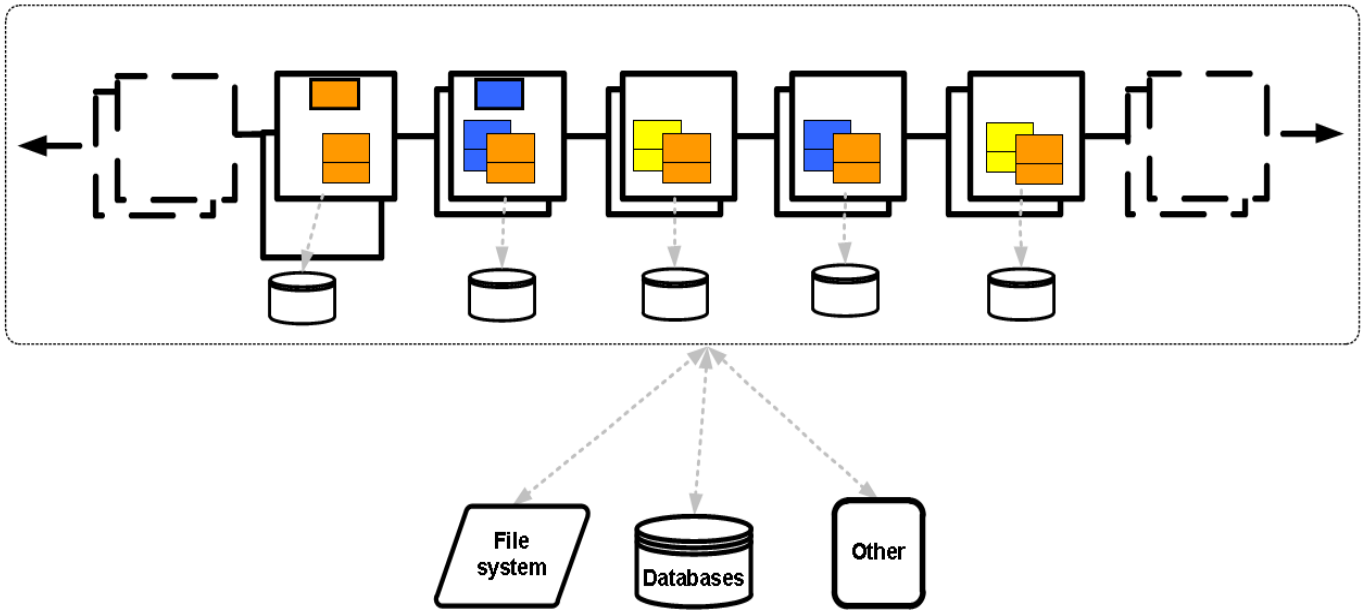
- **So, why is ACID so important for most use cases?**
  - Folks want deterministic outcome not ACID

VMWare offering - vFabric GemFire (GA), SQLFire (in beta)

- **GemFire: Distributed, memory oriented, Object (KV) data management**
- **SQLFire: Similar but SQL is the interface**
- **Target market today**
  - OLTP upto few TB range (all in memory)
  - real-time, low latency, very high concurrent load
  - Not focused on “big data” batch analytics

Some random characteristics

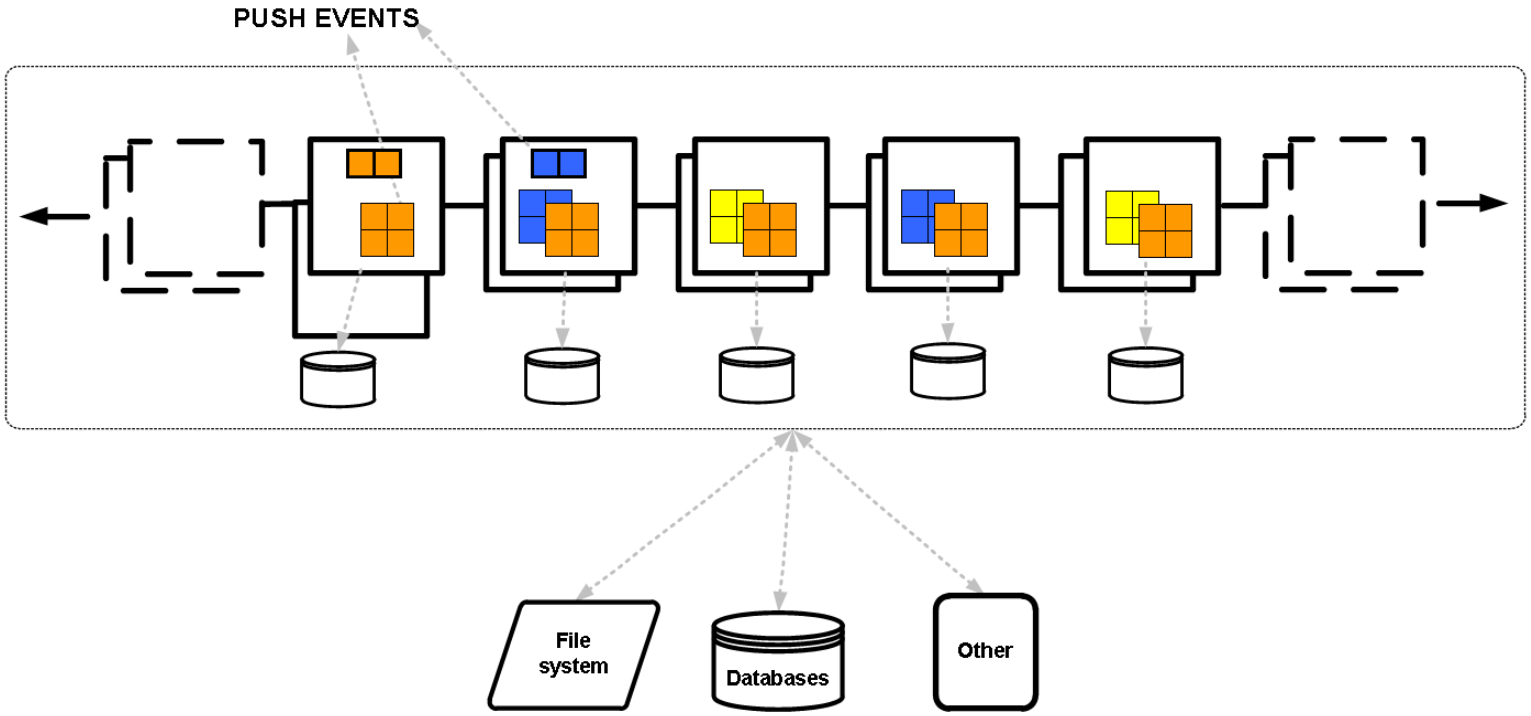
- Replicated or partitioned tables
- Dynamic membership based, allows dynamic, non-blocking changes to cluster size
- HA through synchronous ACK based replication protocol
- Multiple levels of failure detection for stronger consistency
- Replicas are active-active for reads but writes serialized through a "row" owner
- Highly optimized for "colocated transactions" but supports distributed transaction



What is different?

- **Keys and indexes are always in memory**
- **Persistence is just rolling logs with automatic compression**

  Each copy locally persists to disk. Membership changes reliably stored and used to ensure consistency
- **Allow clients to register CQs - push events reliably as updates occur**
- **Framework for read-through, write through and write behind**
- **Async WAN replication**



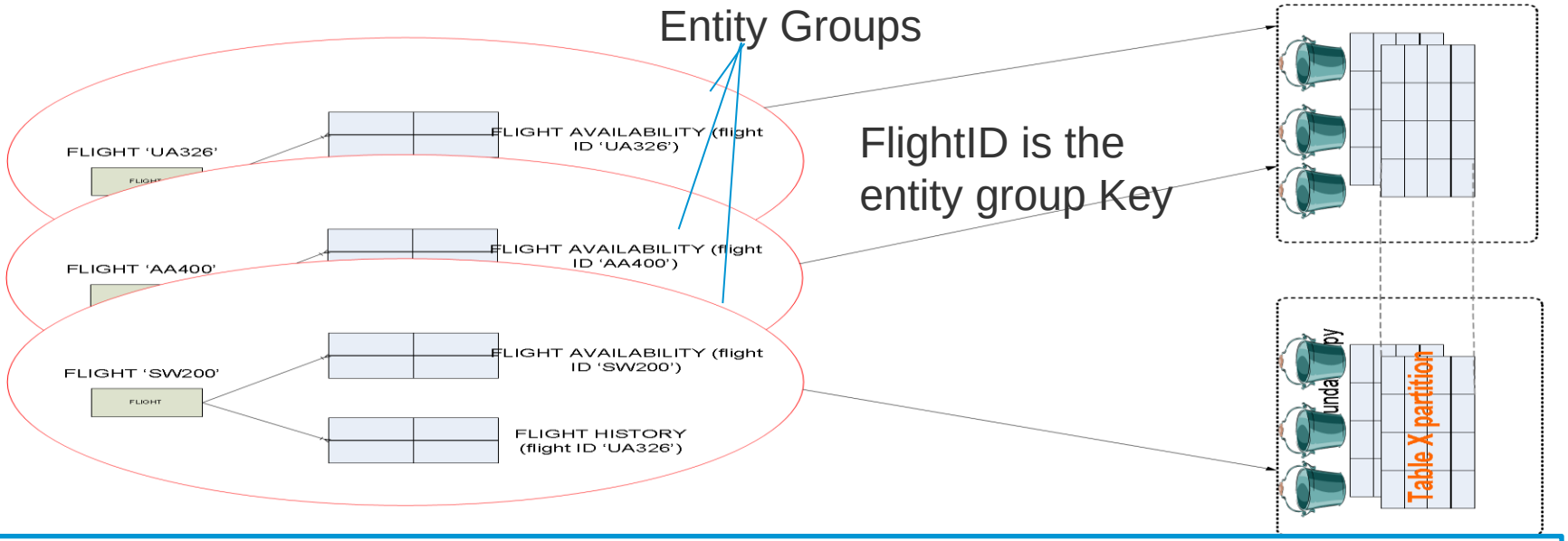
Beyond Key based Hash Partitioning

- We all know Hash partitioning provides uniform load balance
- List, range, or using custom application expression

- Exploit OLTP characteristics for partitioning
  - Often it is the number of entities that grows over time and not the size of the entity.
    - *Customer count perpetually grows, not the size of the customer info*
  - Most often access is very restricted to a few entities
    - *given a FlightID, fetch flightAvailability records*
    - *given a customerID, add/remove orders, shipment records*
  - Root entity frequently fetched with its immediate children

Grouping entities

- Related entities share a "entity group" key and are colocated
- Grouping based on foreign key relationships: look for FK in the compound PK
  - advantage here is that not all entities in group have to share the same key



CreateTable *FlightAvailability(..) partitioned by FlightID colocated with Flights*

Why does this scale?

- requests pruned to a single node or subset of cluster
  - Transactional "write set" is mostly confined to a single entity group
  - Unit of serializability now confined to a single "primary" member managing the entity group
  - Common query joins: across tables that belong to the same group
  - If all concurrent access were to be uniformly distributed across the "entity group" set then you can linearly scale with cluster size

Invariably, access patterns are more complex

- Scalable joins when entity grouping is not possible
  - Reference tables
  - M-M relationships
- Distributed joins impedes scaling significantly
  - pipelining intermediate data sets impacts other concurrent activity

- Answer today:
  - Use replicated tables for reference data
  - one side in the M-M

  - Assumptions
    - update rate on reference data is low
    - one side of the M-M related tables is small and infrequently changing

It doesn’t end here

- realizing a "partition aware" design is difficult
- 80-20 rule: 80% of access at a point in time is on 20% of the data

- lumpy distribution causes hotspots
  - hash partitioning solves this but doesn't help range searches
  - some help: Multi-attribute Grid declustering
  - rebalancing may not help as the entity group (the lump) is a unit of redistribution

- Static grouping vs dynamic grouping
  - e.g online gaming: multiple players that all have to be grouped together lasts only for a game (http://www.cs.ucsb.edu/~sudipto/papers/socc10-das.pdf)

“Good enough” scalable transactions

- Assumptions
  - Small in duration and “write set”
  - Conflicts are rare
- Single row operations always atomic and isolated
- No statement level read consistency for queries
  - Writers almost never block readers
- Single phase commit protocol
  - Eagerly “write lock(local)” on each cohort.
  - “Fail fast” if lock cannot be acquired
  - Transaction isolation at commit time is guaranteed on "write set" in a single partition

Rough thoughts on “Schema flexibility”

- New generation of developers don’t seem to like Schemas ☺
- Drivers
  - Many source of data: it is semi-structured and changing rapidly
  - DB model changes are frequent
  - Adding UDTs and altering tables seen as "rigid“
- E.g.
  - E-commerce app introduces a few products with a stable schema

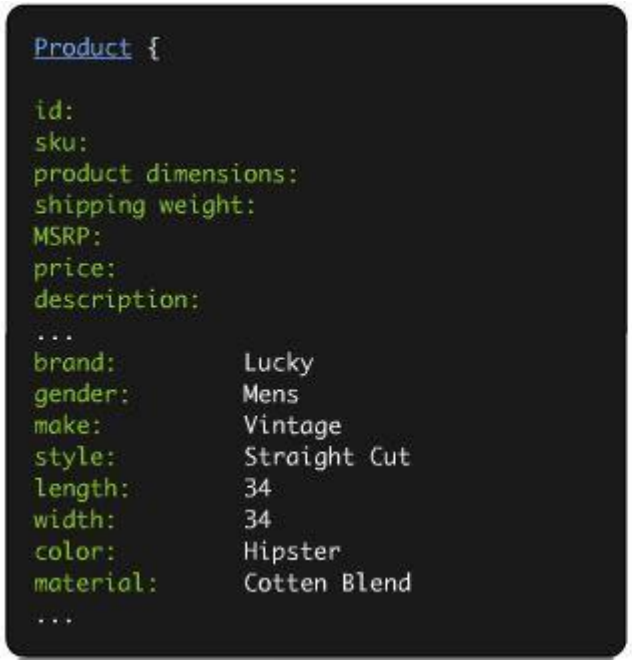
General Product attributes

Book Specific attributes

Source: http://www.nosqldatabases.com/main/2011/4/11/augmenting-rdbms-with-mongodb-for-ecommerce.html

“Schema free”, “Schema less”, etc

- Then, keeps adding support for new products
  - Or, keeps removing products



General Product attributes stay the same

Jeans specific attributes are totally different ... and not consistent across brands & make

- XML datatypes or UDTs or organizing tables in a hierarchy is unnatural and complex
- JSON is considered fat free alternative to XML

The “Polyglot” Data store

- Current thinking

Single OLTP data store for:

1. complex, obese, perpetually changing object graphs
   session state, workflow state

2. Highly structured, transactional data
   sourced from enterprise DBs

3. semi-structured, self describing, rapidly evolving data
   syndicated content, etc

Distributed data store that supports Objects, SQL and JSON ?

Object columns with dynamic attributes

- Extend SQL with dynamic, self describing attributes contained in **Object columns**
- Object columns are containers for self describing K-V pairs (think JSON)
  - values can be objects themselves supporting nesting (composition)
- Can contain collections
- Very easy in most object environments
  - Reflection provides dynamic type under the covers
  - And, hence the object fields become queriable. For interoperability, the type system could be JSON

Some Examples with Object columns

1. Session State- Object tables easily integrate with session state modules in popular app servers

```
create table sessionState (key String, value Object) hash partitioned redundancy level 1;
```

2. Semi-structured docs

```
create table myDocuments (key varchar, documentID varchar, creationTime date, doc Object, tags Object) hash partitioned redundancy level 1;
```

- doc could be a JSON object with each row having different attributes in the object

- tags is a collection of strings

More information at
http://communities.vmware.com/community/vmtn/appplatform/vfabric_sqlfire

Q & A