Eliminate disk access in the real time path

We Challenge the traditional RDBMS design NOT SQL

Too much I/O
Design roots don’t necessarily apply today
- Too much focus on ACID
- Disk synchronization bottlenecks

Buffers primarily tuned for IO
First write to Log
Second write to Data Files
‘Memory is the new bottleneck’

Hardware Changes: The Memory Wall

Trip to memory = 1000s of instructions!

Source: MonetDB
IMDG basic concepts

- Distributed memory oriented store
  - KV/Objects or JSON
  - Queryable, Indexable and transactional

- Multiple storage models
  - Replication, partitioning in memory
  - With synchronous copies in cluster
  - Overflow to disk and/or RDBMS

- Parallelize Java App logic
- Multiple failure detection schemes
- Dynamic membership (elastic)

- Vendors differentiate on
  - Query support, WAN, events, etc

Low latency for thousands of clients
Key IMDG pattern - Distributed Caching

- Designed to work with existing RDBs
  - Read through: Fetch from DB on cache miss
  - Write through: Reflect in cache IFF DB write succeeds
  - Write behind: reliable, in-order queue and batch write to DB
Traditional RDB integration can be challenging

Synchronous “Write through”
- Single point of bottleneck and failure
- Not an option for “Write heavy”
- Complex 2-phase commit protocol
- Parallel recovery is difficult

Asynchronous “Write behind”
- Cannot sustain high “write” rates
- Queue may have to be persistent
- Parallel recovery is difficult
Some IMDG, NoSQL offer ‘Shared nothing persistence’

- Append only operation logs
- Fully parallel
- Zero disk seeks

But, cluster restart requires log scan

- Very large volumes pose challenges
GemFire – How we got here
GemFire – The world as we see it
Our GemFire Journey Over The Years

- Massive increase in data volumes
- Falling margins per transaction
- Increasing cost of IT maintenance
- Need for elasticity in systems

2004
- Financial Services Providers (Every major wall street bank)
- Department of Defense

2008
- Real Time response needs
- Time to market constraints
- Need for flexible data models across enterprise
- Distributed development
- Persistence + In-memory

2014
- Global data visibility needs
- Fast Ingest needs for data
- Need to allow devices to hook into enterprise data
- Always on

- Hybrid Transactional/Analytics grids

- Largest Telcos
- Large mfrers
- Largest Payroll processor
- Auto insurance giants
- Largest rail systems on earth
- Largest travel Portal
- Airlines
- Trade clearing
- Online gambling
Why OSS? Why Now? Why Apache?

• Open Source Software is fundamentally changing buying patterns
  – Developers have to endorse product selection (No longer CIO handshake)
  – Community endorsement is key to product visibility
  – Open source credentials attract the best developers
  – Vendor credibility directly tied to street credibility of product

• Align with the tides of history
  – Customers increasingly asking to participate in product development
  – Resume driven development forces customers to consider OSS products
  – Allow product development to happen with full transparency

• Apache is where you go to build Open Source street cred
  – Transparent, meritocracy which puts developers in charge
  – Roman keeps shouting “Apache!” every few hours
Geode Will Be A Significant Apache Project

• Over a 1000 person years invested into cutting edge R&D
• Thousands of production customers in very demanding verticals
• Cutting edge use cases that have shaped product thinking
• Tens of thousands of distributed, scaled up tests that can randomize every aspect of the product
• A core technology team that has stayed together since founding
• Performance differentiators that are baked into every aspect of the product
GemFire – Architecture Designed For Speed & Scale
Gemfire High Level Architecture

As data changes, subscribers are pushed notification events

Data transparently replicated and/or partitioned; Redundant storage can be in memory and/or on disk

Increase/Decrease capacity on the fly

Shared Nothing disk persistence
Each cache instance can optionally persist to disk

Synchronous read through, write through or Asynchronous write-behind to other data sources and sinks

Many physical machine nodes appear as one logical system
What makes it fast?

• Minimize copying
  – Clients dynamically acquire partitioning meta data for single hop access
  – Avoid JVM memory pools to the extent possible

• Minimize contention points .. avoid offloading to OS scheduler
  – Highly concurrent data structures
  – Efficient data transmission – Nagle’s Algorithm

• Flexible consistency model
  – FIFO consistency across replicas but NO global ordering across threads
  – Promote single row transactions (i.e no transactions)
  – No lock escalation strategies … no Serializable transactions
What makes it fast?

- **Avoid disk seeks**
  - Data kept in Memory – 100 times faster than disk
  - Keep indexes in memory, even when data is on disk
  - Direct pointers to disk location when offloaded (single IOP fetch)
  - Flush only to OS buffers
    - Mitigate failure risks by concurrent disk write on replicas

- **Tiered Caching**
  - Eventually consistent client caches
  - Avoid Slow receiver problems

- **Partition and parallelize everything**
  - Data. Application processing (procedures, callbacks), queries, Write behind, CQ/Event processing
GemFire – Common Usage Patterns
“low touch” Usage Patterns

HTTP Session management
- Simple template for TCServer, TC, App servers
- Shared nothing persistence, Global session state

Hibernate L2 Cache plugin
- Set Cache in hibernate.cfg.xml
- Support for query and entity caching

Memcached protocol
- Servers understand the *memcached* wire protocol
- Use any *memcached* client

Spring Cache Abstraction
- `<bean id="cacheManager" class="org.springframework.data.gemfire.support.GemfireCacheManager"/>`
“Write thru” Distributed caching

- Pre-load data into system
- Lazily load cache misses
- Configure LRU eviction or expiry for large data
- “Write thru” – participate in container transaction
Distributed caching with Async writes to DB

Buffer high write rate from DB

Writes can be enqueued in memory redundantly on multiple nodes

Or, also be persisted to disk on each node

Batches can be conflated and written to DB

Pattern for “high ingest” into Data Warehouse
As a scalable OLTP data store

Shared nothing persistence to disk
Backup and recovery

No Database to configure and be throttled by
As embedded, clustered Java database

Just deploy a JAR or WAR into clustered App nodes

Data can be sync’d with DB is partitioned or replicated across the cluster

Low cost and easy to manage
To process app behavior in parallel

```java
@ParititionedTable(TableName="trades")
public List AnalyzeTrades(@FilterKey Set<String> months, String portfolio) {
...
}
```

Map-reduce but based on simpler RPC
To make data visible across sites in real time
Real Time Analytics With GemFire

- Data stored within GemFire in a “sliding window”
- GemFire map-reduce style in-memory analytics can be performed with data locality
  - Ex: Violation of known trading patterns
- **Benefit**: Early-warning indicators can be identified faster than waiting for analysis on just Pivotal HD
- **Benefit**: Real-time analytics can better influence what kind of big data analytics need to be performed
Analytics on HDFS
The Pivotal Data Fabric (core platform)
Mapping to Products

Disparate input sources

SQL
Objects
JSON

GemFire

SpringXD

HDFS

HAWQ

Deep Scale SQL

PXF
Use case: Telemetry – Net optimization, Location based Svc

- **Revenue Generation**
  - Real-time Location based Mobile Advertising (B2B2C)

- **Revenue Protection**
  - Customer experience management to reduce churn
  - Customers Sentiment analysis

- **Network Efficiency**
  - Network bandwidth optimisation
  - Network signalling maximisation

- **Network optimization**
  - E.g. re-reroute call to another cell tower if congestion detected

- **Location based Ads**
  - Match incoming event to Subscriber profile; If ‘Opt-in’ show location sensitive Ad

- **Challenge: Too much streaming data**
  - Many subscribers, lots of 2G/3G/4G voice/data
  - Network events: location events, CDRs, network issues
Scalable Big Data Architecture for Real time Network analytics

Network sources

1). Raw data

Stream ingestion
- Filter (‘opt-in’), normalize
- Dispatch real time events

In-Memory Cluster

Spring Framework

Subscriber Profile, etc

2). Derived data

Profiles, models

HDFS

Business Rules Engine

Rule Execution
Rule Triggering

3). Analyze billions of events

Batch analytics – Trending, Subscriber location based analytics, etc

HAWQ

© Copyright 2013 Pivotal. All rights reserved.
Demo
Social Network

People Region
Partitioned

Person
Name: String
Description: String

Post Region
Partitioned

Post
Id: PostId(name, date)
Text: String
Basic Save Code

```java
public interface PersonRepository extends CrudRepository<Person, String> {
}
```

```java
@Autowired
PersonRepository people;

public static void main(String[] args) {
    people.save(new Person(name));
}
```
Configuration

```xml
<bean id="pdxSerializer"
  class="com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer">
  <constructor-arg value="io.pivotal.happysocial.model.*"/>
</bean>

<gfe:cache pdx-serializer-ref="pdxSerializer"/>

<gfe:partitioned-region id="people" copies="1"/>
```
public interface PostRepository extends GemfireRepository<Post, PostId> {

    @Query("select * from /posts where id.person=$1")
    public Collection<Post> findPosts(String personName);
}

Collection<Post> posts = postRepository.findPosts(personName);
Indexes

```java
public interface PostRepository extends GemfireRepository<Post, PostId> {
    @Query("select * from /posts where id.person=$1")
    public Collection<Post> findPosts(String personName);
}
```

```java
Collection<Post> posts = postRepository.findPosts(personName);
```

```
<gfe:index id="postAuthor" expression="id.person" from="/posts"/>
```
Colocation

<gfe:partitioned-region id="posts" copies="1" colocated-with="people">
    <gfe:partition-resolver ref="partitionResolver"/>
</gfe:partitioned-region>

Related Posts Are colocated
Functions

Data is Colocated

String personName = pe
Collection<Post> posts =
String sentiment = sentim
return new SentimentR

Behavior is sent to
data (with filter)

client.getSentiment(filter);
Sample Function – Client Side

```java
@Component
@OnRegion(region = "posts")
public interface FunctionClient {
    public List<SentimentResult> getSentiment(@Filter Set<String> people);
}
```
@Autowired private PostRepository postRepository;
@Autowired SentimentAnalyzer sentimentAnalyzer;

@GemfireFunction
public SentimentResult getSentiment(@Filter Set<String> personNames) {
    String personName = personNames.iterator().next();
    Collection<Post> posts = postRepository.findPosts(personName);
    String sentiment = sentimentAnalyzer.analyze(posts);
    return new SentimentResult(sentiment, personName);
}
Parallel, Highly Available Queues
Shared Nothing Persistence

Put k6->v6

Modify k1->v5
Create k6->v6

Modify k1->v5
Create k2->v2
Modify k1->v3
Create k4->v4

Operation Logs with compaction

© Copyright 2013 Pivotal. All rights reserved.
GemFire (Geode) 3.5-4.5X Faster Than Cassandra for YCSB
Horizontal Scaling for GemFire (Geode) Reads With Consistent Latency and CPU

- Scaled from 256 clients and 2 servers to 1280 clients and 10 servers
- Partitioned region with redundancy and 1K data size
Southwest Airlines Technology

Fueling Fast Data At Southwest Airlines: Adopting Gemfire and Cross-Domain Integration
Integrated Data = Better Decisions

If we had **fast access** to **more information**, could we make better gate assignments?

Yes!

- Flight Times
- Passenger Connections
- Crew Connections
- Connecting Bags
- Gate Proximity
- Aircraft Maintenance
NETWORK OPERATIONS CONTROL (NOC)
From

RELATIONAL
ONE ACTIVE DB
NORMALIZED TABLES
ROW LOCKS
SQL JOINS
To

DATA FABRIC

KEY, VALUE STORE
NO JOINS
CAP
DISTRIBUTED GRID
PARTITIONED DATA
BUCKETS
ACTIVE / ACTIVE
Tips: Adopting Gemfire

**DATA PLACEMENT**
Spread across multiple availability zones
Multiple data centers
Number of copies

**CAP THEOREM (insights on choose 2)**
Partitions       = Slow or no ACKS
                 (usually not the network)
Consistency      = Your use case wins
                 (you probably have several different ones)
Convergence      = Some write wins
Tips: Adopting GemFire

DATA LOCKING
“This lock does not mean what you think it means.”

TRANSACTIONS
Data on the same node only

PUT
BEWARE! Stale reads on concurrent puts!
Tips: Adopting Gemfire

PDX
Use it.
Don’t rename enumerated options

SNAPSHOTS
In 7.0.x, you can’t mix PDX IDs between caches

FILESYSTEMS
Shared less = Good!
Cross-Domain Integration

COMPLEX DATA DOMAINS

10M EVENTS DAILY

CREW FLIGHTS PASSENGERS MAINTENANCE
Cross-Domain Integration

AIRPORT SUPPORTING DOMAIN

FLIGHT SCHEDULE CORE DOMAIN

USER AND ACCESS GENERIC DOMAIN

EVERYTHING HAS A CLEAR HOME

PASSENGER SUPPORTING DOMAIN
Resources

Google:

Implementing Domain-Driven Design by Vaughn Vernon

Reactive Enterprise by Vaughn Vernon (published this summer)

CAP Theorem

The Dynamo Paper

Reactive Streams
Thank You!

Brian Dunlap
Technical Lead

bdunlap22@gmail.com

@brianwdunlap