Pivotal
A NEW PLATFORM FOR A NEW ERA
Evolution of Pivotal Gemfire

Which way might the "Apache Way" take it?

Roman Shaposhnik
rvs@apache.org
Director of Open Source, Pivotal Inc.
@rhatr

Milind Bhandarkar
milind@ampool.io
CEO, Ampool Inc.
@techmilind
Topics

- GemFire history, architecture, and use cases
- Geode as the open source core of GemFire
- Requirements of the modern data infrastructure
- Butterfly architecture
- Geode as an engine for in-memory data exchange layer
It’s not the size of DATA

it’s how you use 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

- Largest Telcos
- Large mfrers
- Largest Payroll processor
- Auto insurance giants
- Largest rail systems on earth

- Largest travel Portal
- Airlines
- Trade clearing
- Online gambling
Apps at Scale Have Unique Needs

Pivotal GemFire is the distributed, in-memory database for apps that need:

1. Elastic scale-out performance
2. High performance database capabilities in distributed systems
3. Mission critical availability and resiliency
4. Flexibility for developers to create unique applications
5. Easy administration of distributed data grids
1. Elastic Scale-Out Performance

Linear scalability

Elastic capacity

Latency-minimizing data distribution

China Railway Corporation

“The system is operating with solid performance and uptime. Now, we have a reliable, economically sound production system that supports record volumes and has room to grow”

Dr. Jiansheng Zhu, Vice Director of China Academy of Railway Sciences

- 4.5 million ticket purchases & 20 million users per day.
- Spikes of 15,000 tickets sold per minute, 40,000 visits per second.
2. High Performance Database Capabilities

Performance-optimized persistence

Configurable consistency

Partitioned  Replicated  Disabled

Distributed & continuous queries

Distributed transactions, indexing, archival

Newedge

“Our global deployment of Pivotal GemFire’s distributed cache gives me a single version of the trade – resolving hard-to-test-for synchronization issues that exist within any globally distributed business application architecture”

Michael Benillouche, Global Head of Data Management
3. Mission Critical Availability and Resiliency

Cluster resilience & failover

Cluster to cluster WAN connectivity

Gire

“We can track and collect money at our 4,000+ kiosks and branches – even without a reliable Internet connection. GemFire provides the core data grid and a significant amount of related functionality to help us handle this unreliable network problem”

Gustavo Valdez, Chief of Architecture and Development

- 19 million payment transactions per month
- 4000+ points of sale with intermittent Internet connectivity
4. Flexibility for Developers to Create Unique Apps

- **Data Structures:**
  - User-defined classes
  - Documents (JSON)

- **Native language support:**
  - Java, C++, C#

- **API's**
  - Java: Hashmap
  - Memcache
  - Spring Data GemFire
  - C++ Serialization API's

- **Embedded query authoring**
  - Object Query Language (OQL)

- **Publish & subscribe framework for continuous query & reliable asynchronous event queues**

- **App-server embedded functionality:**
  - Web app session state caching
  - L2 Hibernate
What makes it fast?

- Design center is RAM not HDD
- Really demanding customers
- Avoid and minimize, particularly on the critical read/write paths:
  - Network hops, copying data, contention, distributed locking, disk seeks, garbage
- Lots and lots of testing
  - Establish and monitor performance baselines
  - Distributed systems testing is difficult!
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
GemFire (Geode) 3.5-4.5X Faster Than Cassandra for YCSB
Apache Geode
(incubating)

The open source core of GemFire
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
Geode or GemFire?

- Geode is a *project*, GemFire is a *product*
- We donated everything but the kitchen sink*
- More code drops imminent; going forward all development happens OSS-style (“The Apache Way”)

* Multi-site WAN replication, continuous queries, and native (C/C++) client driver
Performance is key. Consistency is a must.

Geode is the open source distributed, in-memory database for scale-out applications.

geode.incubator.apache.org
Why OSS? Why Now? Why Apache?

• Open Source Software is fundamentally changing buying patterns
  – Developers have to endorse product selection (No longer CIO handshake)
  – Open source credentials attract the best developers
  – Open Source has replaced standards

• Align with the tides of history
  – Customers increasingly asking to participate in product development
  – Allow product development to happen with full transparency

• Apache Way
  – “Community over code”
  – Use cases far beyond Pivotal Gemfire’s
Beyond Gemfire’s core
Roadmap

• HDFS persistence
• Off-heap storage
• Lucene indexes
• Spark integration
• Cloud Foundry service

…and other ideas from the Geode community!
Geode in modern data infrastructure
Infrastructure is increasingly Scale-Out
Memory throughput growth

[Graph showing memory bandwidth through time, with date of availability and module bandwidths indicated.]
Memory hierarchy getting deeper
One-size-fits-all Data Platform Era is Over
Analytics moving from Batch to Real-Time
What happened?
Web 2.0
And, then...
In a blink of an eye...
Tape is Dead
Disk is Tape
Flash is Disk
RAM Locality is King

Jim Gray
Microsoft
December 2006
The *missing* building block
Sharing operational data at the speed of RAM

In-Memory Data Exchange Layer

Spark

...frontend processing frameworks...

HAWQ

HDFS

...backend archival stores...

Isilon
Spark’s view on how to lock you in

- Spark
- HDFS
- In-Memory Data Exchange Layer
- HAWQ
- Isilon

...frontend processing frameworks...

...backend archival stores...
HDFS’s view on how to lock you in

Spark  …frontend processing frameworks…  HAWQ

HDFS  In-Memory Data Exchange Layer

…backend archival stores…  Isilon
How is open source solving this?
Short list of open source contenders

- Tachyon
- Infinispan
- Apache Ignite (incubating)
- Apache Geode (incubating)
Short list of open source contenders

- Tachyon
- Infinispan
- Apache Ignite (incubating)
- Apache Geode (incubating)
Geode’s secret sauce

• Community!
• Maturity
• Scalability
• Building blocks
A few key building blocks

- PDX Serialization
- Asynchronous Events
Fixed or flexible schema?

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>pet_id</th>
</tr>
</thead>
</table>

OR

```json
{
    id : 1,
    name : "Fred",
    age : 42,
    pet : {
        name : "Barney",
        type : "dino"
    }
}
```
But how to serialize data?

Portable Data eXchange

<table>
<thead>
<tr>
<th></th>
<th>header</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>header</td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>pdx</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>dsid</td>
<td>typeId</td>
</tr>
<tr>
<td></td>
<td>fields</td>
<td>offsets</td>
</tr>
</tbody>
</table>
Efficient for queries

```
SELECT p.name from /Person p WHERE p.pet.type = "dino"

{
  id : 1,
  name : "Fred",
  age : 42,
  pet : {
    name : "Barney",
    type : "dino"
  }
}
```
Easy to use

- Access from Java, C#, C++, JSON
- Domain objects not required
- Automatic type definition
  - No IDL compiler or schema required
  - No hand-coded read/write methods
Asynchronous Events – Design Goals

• High availability

• Low latency, high throughput

• Deliver events to a receiver without impacting the write path
Questions?

- http://geode.incubator.apache.org
- dev@geode.incubator.apache.org
- user@geode.incubator.apache.org
- http://github.com/apache/incubator-geode
DO WE HAVE ANY ACTIONABLE ANALYTICS FROM OUR BIG DATA IN THE CLOUD?

YES, THE DATA SHOWS THAT MY PRODUCTIVITY PLUNGES WHENEVER YOU LEARN NEW JARGON.

MAYBE IN-MEMORY COMPUTING WILL ACCELERATE YOUR APPLICATIONS.

PLUNGE, PLUNGE, PLUNGE.
Bonus Content
Distributed type registry

Person p1 = ...
region.put("Fred", p1);
Distributed type registry

Person p1 = ...
region.put("Fred", p1);

© Copyright 2015 Pivotal. All rights reserved.
Distributed type registry

Person p1 = ...
region.put("Fred", p1);

automatic definition
Distributed type registry

Person p1 = ...
region.put("Fred", p1);

automatic definition

replicate serialized data containing typeId
Schema evolution

v2 objects preserve data from missing fields

v1 objects use default values to fill in new fields

PDX provides forwards and backwards compatibility, no code required
Asynchronous Events
Serial Queues

Member 1
- Enqueue
- LOL!!

Member 2
- Replicate
- Primary Queue
- Secondary Queue

Member 3
- sup
- sup

put
Serial Queues

Member 1

Member 2

Member 3

Dispatch Events from Primary

AsyncEventListener
{
  processBatch()
}

LOL!!

sup

LOL!!

sup

LOL!!

sup

sup
Parallel Queues

Primary Queue (Partition 1)

Primary Queue (Partition 2)

Secondary Queue (Partition 1)

LOL!!

sup

LOL!!

sup

Word

sup

Word

LOL!!

sup

put
Parallel Queues

AsyncEventListener
{
    processBatch()
}

AsyncEventListener
{
    processBatch()
}
“There are only two hard things in Computer Science: cache invalidation, naming things, and off-by-one errors.”

– the internet
Client driver

• Intelligent – understands data distribution for single hop network access

• Caching – can be configured to locally cache data for even faster access

• Events – `registerInterest()` in keys to receive push notifications
Client subscriptions

- Useful for refreshing client cache, pushing events ("topics") to multiple clients
- Highly available & scalable via in-memory replicated queues
- Events are ordered, at-least-once delivery
- Durable subscriptions, conflation optional
Client subscriptions

Client

region.registerInterest("Fred");
Client subscriptions

```java
Person p1 = ...
region.put("Fred", p1);

region.registerInterest("Fred");
```
Client subscriptions

Client

Client

Person pl = …
region.put("Fred", pl);

region.registerInterest("Fred");
Client subscriptions

Client

Client

Person p1 = ...
region.put("Fred", p1);

region.registerInterest("Fred");
Client subscriptions

Person p1 = ...
region.put("Fred", p1);

region.registerInterest("Fred");
Use cases and 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"`
Application Patterns

• Caching for speed and scale
  – Read-through, Write-through, Write-behind

• Geode as the OLTP system of record
  – Data in-memory for low latency, on disk for durability

• Parallel compute engine

• Real-time analytics
Development Patterns

• **Configure** the cluster programmatically or declaratively using cache.xml, gfsh, or SpringDataGemFire

```xml
<cache>
  <region name="turbineSensorData" refid="PARTITION_PERSISTENT">
    <partition-attributes redundant-copies="1" total-num-buckets="43"/>
  </region>
</cache>
```
Development Patterns

• **Write** key-value data into a Region using \{ create | put | putAll | remove \}
  
  – The value can be flat data, nested objects, JSON, ...

Region sensorData = cache.getRegion("TurbineSensorData");

SensorKey key = new SensorKey(31415926, "2013-05-19T19:22Z"); // turbineId, timestamp
sensorData.put(key, new TurbineReading()
  .setAmbientTemp(75)
  .setOperatingTemp(80)
  .setWindDirection(0)
  .setWindSpeed(30)
  .setPowerOutput(5000)
  .setRPM(5));
Development Patterns

- **Read** values from a Region by key using \{ get | getAll \}

  ```java
  TurbineReading data = sensorData.get(new SensorKey(31415926, "2013-05-19T19:22Z"));
  ```

- **Query** values using OQL

  ```java
  // finds all sensor readings for the given turbine
  SELECT * from /TurbineSensorData.entrySet WHERE key.turbineId = 31415926

  // finds all sensor readings where the operating temp exceeds a threshold
  SELECT * from /TurbineSensorData.entrySet WHERE value.operatingTemp > 120
  ```

- **Apply indexes to optimize queries**
Development Patterns

- Execute **functions** to operate on local data in parallel
- Respond to updates using **CacheListeners** and **Events**

- Automatic redundancy, partitioning, distribution, consistency, network partition detection & recovery, load balancing, …