In-Memory Computing Brings Operational Intelligence to Business Challenges

Dr. William L. Bain
ScaleOut Software, Inc.
About the Speaker

• Dr. William Bain, Founder & CEO of ScaleOut Software:
  • Email: wbain@scaleoutsoftware.com
  • Ph.D. in Electrical Engineering (Rice University, 1978)
  • Career focused on parallel computing – Bell Labs, Intel, Microsoft
  • 3 prior start-ups, last acquired by Microsoft and product now ships as Network Load Balancing in Windows Server

• ScaleOut Software develops and markets **In-Memory Data Grids**, software for:
  • Scaling application performance with in-memory data storage
  • Analyzing live data in real time with in-memory computing

• Thirteen+ years in the market; 450+ customers, 12,000+ servers
Agenda

The evolution of in-memory computing for operational intelligence:
• The foundation: in-memory data grids (IMDGs)
• The challenges: using IMDGs for caching with parallel query
• Data-parallel computing: delivering operational intelligence
  • Examples in financial services
• The next step: data-parallel computing with method invocations
  • Examples in financial services and cable media
• Evolution into stream-processing: the digital twin model
  • Examples in ecommerce, logistics, IOT, medical device tracking, and more
• Combining stream-processing and data-parallel computing with an IMDG
In-Memory Data Grid (IMDG)

IMDGs provide fast, scalable, distributed in-memory data storage.

What is an IMDG?

- IMDG stores live, object-oriented data:
  - Uses a key/value storage model for large object collections.
  - Maps objects to a cluster of commodity servers with location transparency.
  - Has predictably fast (<1 msec.) data access and updates.
  - Designed for transparent scaling and high availability

Basic “CRUD” APIs:
- Create(key, obj, tout)
- Read(key)
- Update(key, obj)
- Delete(key)
Wide Range of Applications for IMDGs

IMDGs are typically used as a distributed cache.

<table>
<thead>
<tr>
<th>Vertical</th>
<th>Selected Use Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Financial &amp; Insurance</td>
<td>• Online banking • Brokerage</td>
</tr>
<tr>
<td></td>
<td>• Loan apps • Trading • Data services</td>
</tr>
<tr>
<td></td>
<td>• Portfolio analysis • Risk management</td>
</tr>
<tr>
<td></td>
<td>• Position updating</td>
</tr>
<tr>
<td>Ecommerce</td>
<td>• Shopping carts • User state</td>
</tr>
<tr>
<td></td>
<td>• Reward programs • Instant offers</td>
</tr>
<tr>
<td></td>
<td>• Product catalogs • Sale spikes</td>
</tr>
<tr>
<td>Online Services</td>
<td>• Patient records • SaaS • User preferences</td>
</tr>
<tr>
<td></td>
<td>• Service processing • Online education</td>
</tr>
<tr>
<td></td>
<td>• Internet provisioning • Legal analysis</td>
</tr>
<tr>
<td>Entertainment &amp; Communication</td>
<td>• Online game state • Streaming media</td>
</tr>
<tr>
<td></td>
<td>• Online bill pay • Services catalog</td>
</tr>
<tr>
<td></td>
<td>• Gambling analysis</td>
</tr>
<tr>
<td>Travel &amp; Transportation</td>
<td>• Ticketing system • Reservation system</td>
</tr>
<tr>
<td></td>
<td>• Reservation analysis</td>
</tr>
</tbody>
</table>
Using IMDG as a Cache: Parallel Query

- Users often have a database mindset and rely on query.
  - Query retrieves a set of objects with selected properties and/or tags.
    - Uses all grid servers to access queried data.
- Challenges:
  - Cost = $O(N)$ for $N$ servers (vs. $O(1)$ for CRUD)
  - Can create excessive network traffic
- Intermediate solution: filter methods:
  - Run Boolean method on objects to refine search.
  - Example (C#):
    - (select stocks where region == NW)
    - .Filter(EvalPriceChanges());
  - Reduces number of objects returned to client.
  - Provides a bridge to data-parallel computing.
The Next Step: Operational Intelligence (OI)

**Goal**: Provide *immediate* (sub-second) feedback to a system handling live data.

- An IMDG hosts live data and can introspect on that data in real time.
- This delivers much greater value that just using the grid as a cache.
- A few example use cases requiring immediate feedback within a live system:
  - **Ecommerce**: personalized, real-time recommendations
  - **Healthcare**: patient monitoring, predictive treatment
  - **Equity trading**: minimize risk during a trading day
  - **Reservations systems**: identify issues, reroute, etc.
  - **Credit cards & wire transfers**: detect fraud in real time
  - **IoT, smart grids**: predictive analytics & optimization
Operational vs Business Intelligence

**Operational Intelligence**

- Real-time
- Live data sets
- Gigabytes to terabytes
- In-memory storage
- Sub-seconds to seconds

**Best uses:**
- Tracking live data
- Immediately identifying trends and capturing opportunities
- Providing immediate feedback

**Business Intelligence**

- Batch
- Static data sets
- Petabytes to exabytes
- Disk-based storage
- Minutes to hours

**Best uses:**
- Analyzing warehoused data
- Mining for long term trends
- Detecting patterns of strategic importance
Operational Intelligence Reduces Time to Action

In-Memory Computing Delivers OI in Three Ways:

1. Captures live data with extremely low latency.
2. Continuously analyzes a live system to identify opportunities.
3. Makes automated decisions before the moment is lost.
IMDG can have simple, fast APIs for scalable, data-parallel computing:

- “Parallel Method Invocation” (PMI)
  - Follows IMDG’s object-oriented storage model.
  - Defines data-parallel tasks as class methods.
  - Runs class methods in parallel across the cluster and performs a distributed merge of results.

- Advantages:
  - Uses standard, well understood “eval/merge” paradigm from parallel supercomputing.
  - Takes advantage of cluster’s servers and cores.
  - Moves the code to the data; avoid delays due to data motion.

- Can be used to build more complex data-parallel operators (e.g., MapReduce)
Comparing Query with Data-Parallel Computing

- Data-parallel computing moves the analysis to the IMDG and reduces data motion:
Example of PMI for OI in Financial Services

Back-testing stock trading strategies on stock histories:

- A widely used application - “embarrassingly parallel”
- Hosted an IMDG in Amazon EC2 using 75 servers holding 1 TB of stock history data in memory
- IMDG handled a continuous stream of updates (1.1 GB/s)
- Results: analyzed 1 TB in 4.1 seconds (250 GB/s).
- Observed near-linear scaling as dataset and update rate grew.
The Benefit of Computing in the IMDG

• Just using IMDG as a cache causes data motion for every operation.

• Network access creates a bottleneck that limits throughput and increases latency.

• Avoiding data motion enables linearly scalable throughput for growing workloads => predictable, low latency.
Data-Parallel Execution Steps

- **Eval** phase: each server queries local objects and runs eval and merge methods:
  - Accessing local objects avoids data motion.
  - Completes with one result object per server.

- **Merge** phase: all servers perform binary, distributed merge to create final result:
  - Merge runs in parallel to minimize completion time.
  - Returns final result object to client.
Example in Financial Services

• **Goal**: track market price fluctuations for a hedge fund to keep portfolios in balance across market sectors.

• **Solution**:
  - Keep portfolios of stocks (long and short positions) in an object collection within IMDG.
  - Collect market price changes in one-second snapshots.
  - Define a method which applies a snapshot to each portfolio and optionally generates an alert to rebalance.
  - Perform periodic (1/sec) parallel method invocations on the collection of portfolios.
  - Combine alerts in parallel using a second user-defined merge method.
  - Report alerts to UI every second for fund manager.
Outputs Continuous Alerts to the UI

- PMI runs every second; it completes in 350 msec. and immediately refreshes UI.
- Encapsulates proprietary analysis algorithm.
- UI alerts trader to portfolios that need rebalancing.
- UI allows trader to examine portfolio details and determine specific positions that are out of balance.
**Goal:** Match orders to inventory in real time and report issues before committing orders for perishable goods.

- **Customer’s approach:**
  - Use IMDG as a cache with orders and inventory changes stored as objects by SKU in separate name spaces.
  - Perform nightly reconciliation; for each SKU:
    - Query all orders by SKU.
    - Query inventory changes by SKU.
    - Run proprietary reconciliation algorithm and generate alerts.

- **Problems:**
  - Very poor performance (1+ hours) due to parallel queries and data motion
  - Results not available in real time
Data-Parallel Solution

Key challenge for data-parallel computing: choose the right domain

**Solution:** Store data by SKU in the IMDG and perform data-parallel reconciliation on all SKU objects.
- Merge operation returns alerts.

**Advantages:**
- Pre-joined orders and inventory to avoid queries.
- Avoided network bottleneck from sending objects to external compute cluster.
- Eliminated need for a compute cluster.
- Reduced reconciliation time to <1 minute.
- Enabled real-time alerting.
Single Method Invocation (SMI)

- Another form of data-parallel computation
- Created for a financial services application performing column-oriented computations.
- Invokes user-defined method on a single, selected object:
  - Ships parameters to invoking method.
  - Enables object to be updated.
  - Efficiently returns results to invoking client.
- **Benefits:**
  - Avoids data motion to/from client.
  - Encapsulates application code and stages code in the grid.
  - Minimizes latency to invoke method and return results.
How an IMDG Runs Computations

• Each grid host runs a worker process which executes application-defined methods on stored objects.
  • The set of worker processes is called an *invocation grid (IG)*.
  • IG usually runs language-specific runtimes (JVM, .NET).
  • IMDG can ship code to the IG workers.

• Key advantages for IGs:
  • Follows object-oriented model.
  • Avoids network bottlenecks by moving computing to the data.
  • Leverages IMDG’s cores & servers.
IMDG Executes Data-Parallel Methods

Method execution implements a parallel op. on an object collection:

- Client runs a single method on all objects in a collection.
- Execution runs in parallel across the grid.
- Results are merged and returned to the client.
IMDG Executes Methods for Single Objects

Method execution runs independently for each request:

- IMDG directs request to a specific object for execution with low latency.
- IMDG executes multiple methods in parallel for high throughput.
OI Example: Tracking Cable Viewers

• **Cable Company’s Goals:**
  • Make real-time, personalized upsell offers.
  • Immediately respond to service issues & hotspots.
  • Track aggregate behavior to identify patterns, e.g.:
    • Total instantaneous incoming event rate from set-top boxes
    • Most popular programs and # viewers by zip code

• **Requirements:**
  • Track channel-change events from 10M set-top boxes with 25K events/sec (2.2B/day).
  • Correlate, cleanse, and enrich events per rules (e.g. ignore fast channel switches, match channels to programs) within 5 seconds (from current 6+ hours).
  • Refresh aggregate statistics every 10 seconds.
Implementation with both SMI and PMI

- Each set-top box is represented as an object in the IMDG
- Object holds raw & enriched event streams, viewer parameters, and statistics
- IMDG captures incoming events by updating objects
- IMDG uses both forms of data-parallel computation to:
  - immediately update box objects to generate alerts to recommendation engine using SMI, and
  - continuously collect and report global statistics using PMI across box objects
- Demonstrates the use of an IMDG for stream processing.
Built a POC to demonstrate performance and scalability for cable vendor:

- Based on a simulated workload for San Diego metropolitan area
- Continuously correlated and cleansed telemetry from 10M simulated set-top boxes (using a synthetic load generator)
- Processed more than 30K events/second
- Enriched events with program information every second
- Tracked aggregate statistics (e.g., top 10 programs by zip code) every 10 seconds
IMDG Processes Events with ReactiveX

ReactiveX reduces latency for each request compared to SMI:

• IMDG directs events to a specific object for handling by ReactiveX observers.

• IMDG handles multiple events in parallel for high throughput.
Stream Processing for Fast Queries

Challenge: How to query stock histories that are being continuously updated from a ticker feed?

• Requirements:
  • Must hold all prices from today’s and yesterday’s ticker feed.
  • Must support 1000s of simultaneous queries.
  • Each query must see the latest price updates.
  • Queries may have hotspots due to popular stocks.

• Current solution:
  • Replicate all stock price data across 12+ databases for simultaneous access.
  • Use a compute cluster.
  • Not clear how to keep databases coherent; expensive
Solution Using an IMDG

• Store each stock’s price history as a pair of objects (today’s and yesterday’s prices).
• Apply updates to today’s stock prices as streaming events.
• Query stock prices with fast key/value reads.
• Client caches host latest values for objects using 2+ objects per stock minimizes data motion.
• Implement a special read mode that always reads cache and asynchronously applies updates to refresh.
• Advantages:
  • All queries have fast, predictable latency.
  • Hotspots do not affect latency.
Stateful Stream-Processing on an IMDG

• IMDG is well suited to use the “digital twin” model created by Michael Grieves; popularized by Gartner.

• This model represents each data source with a grid object that holds:
  • An event collection
  • State information about the data source
  • Logic for analyzing events, updating state, and generating alerts

• Benefits:
  • Offers a structured approach to stateful stream processing.
  • Automatically correlates incoming events by data source.
  • Integrates all relevant context (events & state).
  • Enables easy deployment of application-specific logic (e.g., ML, rules engine, etc.) for analysis and alerting.
  • Provides domain for aggregate analysis and feedback.

In-Memory Computing Summit North America 2018
Example: Tracking a Fleet of Vehicles

• **Goal:** Track telemetry from a fleet of cars or trucks.
  - Events indicate speed, position, and other parameters.
  - Digital twin object stores information about vehicle, driver, and destination.
  - Event handler alerts on exceptional conditions (speeding, lost vehicle).

• Periodic data-parallel analytics determines aggregate fleet performance:
  - Computes overall fuel efficiency, driver performance, vehicle availability, etc.
  - Can provide feedback to drivers to optimize operations (e.g., avoiding congested areas).
Example: Heart-Rate Watch Monitoring

Tracks heart-rate for a large population of runners:

• Heart-rate events flow from smart watches to their respective digital twin objects for analysis.
• The analysis uses wearer’s history, activity, and aggregate statistics to determine feedback and alerts.
Data Parallel Analysis Across all Digital Twins

• Uses IMDG’s in-memory compute engine to create aggregate statistics in real time.

• Results can be reported to analysts and updated every few seconds.

• Results can be used as feedback for event analysis in digital twin objects and/or reported to users.
**Example: Ecommerce Recommendations**

**Goal:** Deliver real-time recommendations to 1000s of online shoppers.

- Each shopper generates a clickstream of products searched.

- **Stream-processing system must:**
  - Correlate clicks for each shopper and associate with shopper’s preferences.
  - Maintain a history of clicks during a shopping session.
  - Analyze clicks to create new recommendations within 100 msec.

- **Analysis must:**
  - Take into account the shopper’s preferences and demographics.
  - Create and use aggregate feedback on collaborative shopping behavior.
Providing Recommendations in Real Time

• Requires scalable stream-processing to analyze each click and respond in <100ms:
  • Accept input with each event on shopper’s preferences.
  • Provide aggregate feedback on best-selling products.
Implementation Using Digital Twin Objects

Tracks shoppers as digital twins and makes real-time recommendations:

- Each DT object holds clickstream of browsed products, preferences, and demographics.
- Event handler analyzes this data and immediately updates recommendations.
  - Product descriptions are kept in second object collection in the IMDG.
  - These descriptions are uploaded from the product database.
- Periodic data-parallel, batch analytics across all shoppers determine aggregate trends:
  - Examples include best selling products, average basket size, etc.
  - Used for analysis and real-time feedback

In-Memory Computing Summit North America 2018
Providing Key Aggregate Metrics

• Periodic data-parallel computation generates aggregate statistics across DT objects for all shoppers:
  • Tracks real-time shopping behavior.
  • Charts key purchasing trends.
  • Enables merchandizer to create promotions dynamically.

• Aggregate statistics can be shared with shoppers:
  • Allows shoppers to obtain collaborative feedback.
  • Examples include most viewed and best selling products.
In-memory computing enables operational intelligence.

- **Challenge**: using an IMDG solely as a cache does not take advantage of its ability to introspect on live data and return results in real time.
  - Users tend to view IMDGs as in-memory databases and rely heavily on queries.

- **Operational intelligence can capture new opportunities** that boost competitive value.

- **Data-parallel computing for OI in an IMDG** offers several key benefits:
  - It boosts application performance by moving code to the data, avoiding network bottlenecks.
  - It can be implemented using object-oriented constructs, which cleanly separate application code from the IMDG’s orchestration mechanisms.
  - It delivers results in real time for live data.

- **Stream processing in an IMDG** allows deeper introspection than previously possible:
  - IMDGs provide an excellent platform for the digital twin model, which has many applications.
In-Memory Computing for Operational Intelligence

www.scaleoutsoftware.com