ScaleOut Software

Implementing Operational Intelligence Using In-Memory Computing

In-Memory Computing Summit 2015

William L. Bain (wbain@scaleoutsoftware.com)

June 29, 2015

Agenda

ScaleOut Software

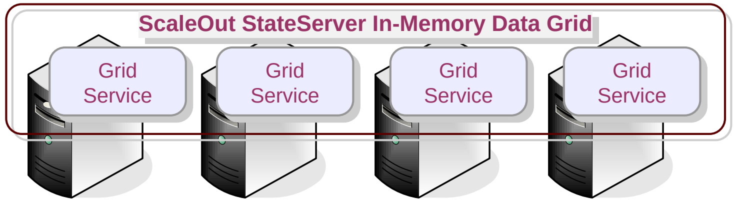
- What is Operational Intelligence?
- Example: Tracking Set-Top Boxes
- Using an In-Memory Data Grid (IMDG) for Operational Intelligence
  - Tracking and analyzing live data
  - Comparison to Spark
- Implementing OI Using Data-Parallel Computing in an IMDG
- A Detailed OI Example in Financial Services
  - Code Samples in Java
- Implementing MapReduce on an IMDG
- Optimizing MapReduce for OI
- Integrating Operational and Business Intelligence

About ScaleOut Software

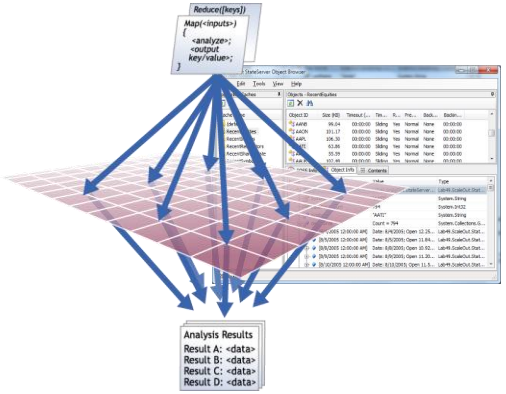
- Develops and markets **In-Memory Data Grids**, software middleware for:
  - **Scaling application performance** and
  - **Providing operational intelligence** using
  - **In-memory data storage and computing**
- Dr. William Bain, Founder & CEO
  - Career focused on parallel computing – Bell Labs, Intel, Microsoft
  - 3 prior start-ups, last acquired by Microsoft and product now ships as Network Load Balancing in Windows Server
- Ten years in the market; 400+ customers, 10,000+ servers
- Sample customers:

ScaleOut Software's Product Portfolio

- **ScaleOut StateServer® (SOSS)**
  - In-Memory Data Grid for Windows and Linux
  - Scales application performance
  - Industry-leading performance and ease of use

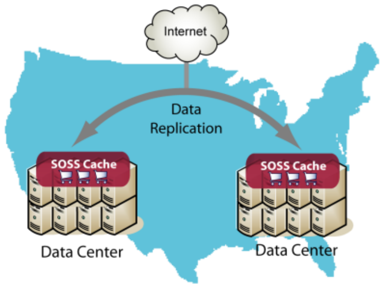


- **ScaleOut ComputeServer™** adds
  - Operational intelligence for "live" data
  - Comprehensive management tools



- **ScaleOut hServer®**
  - Full Hadoop Map/Reduce engine (>40X faster*)
  - Hadoop Map/Reduce on live, in-memory data

- **ScaleOut GeoServer®**
  - WAN based data replication for DR
  - Global data access and synchronization



*in benchmark testing

In-Memory Computing Is Not New

ScaleOut Software

- 1980's: SIMD Systems, Caltech Cosmic Cube

Thinking Machines Connection Machine 5

- 1990's: Commercial Parallel Supercomputers

Intel IPSC-2

IBM SP1

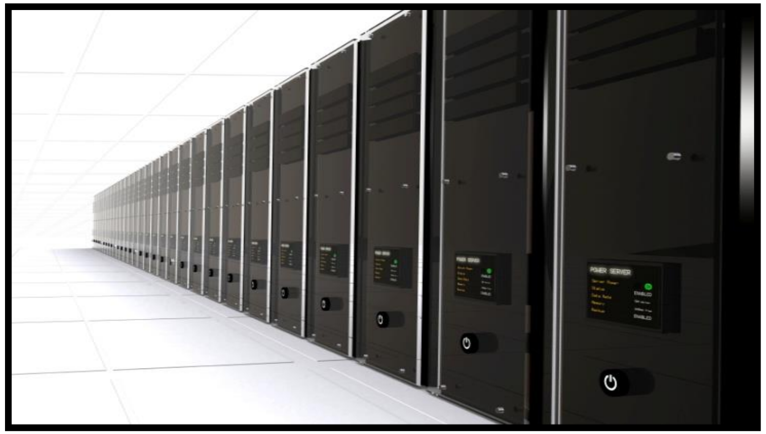
What's New: IMC on Commodity Hardware

ScaleOut Software

- 1990's – early 2000's: HPC on Clusters

HP Blade Servers

- Since ~2005: Public Clouds



Amazon EC2, Windows Azure

ScaleOut Software

Introductory Video:

What is Operational Intelligence

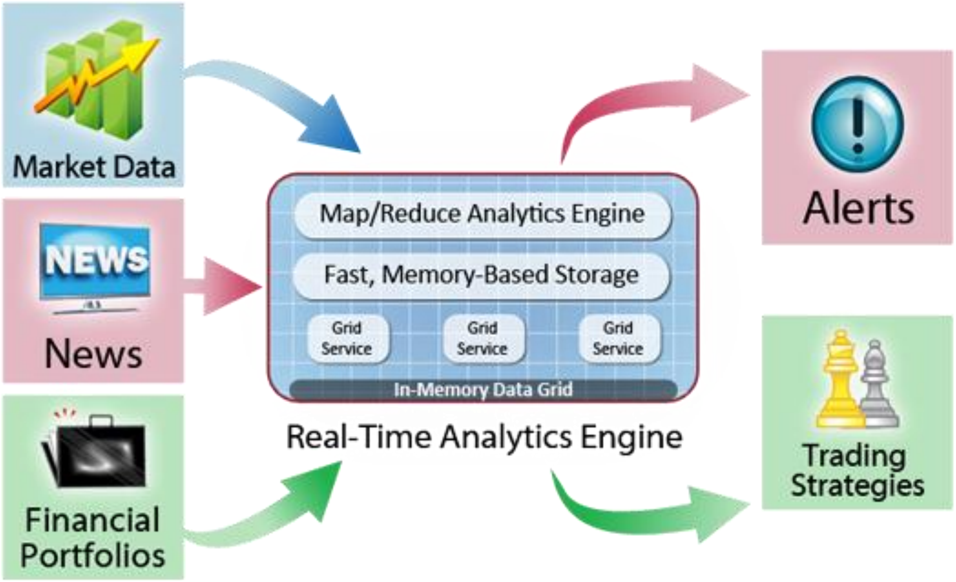
https://www.youtube.com/watch?v=H6OFzdIEy-g&feature=youtu.be

Online Systems Need Operational Intelligence

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

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**: optimize power distribution & detect issues



Operational vs Business Intelligence

Operational Intelligence

Real-time

Live data sets

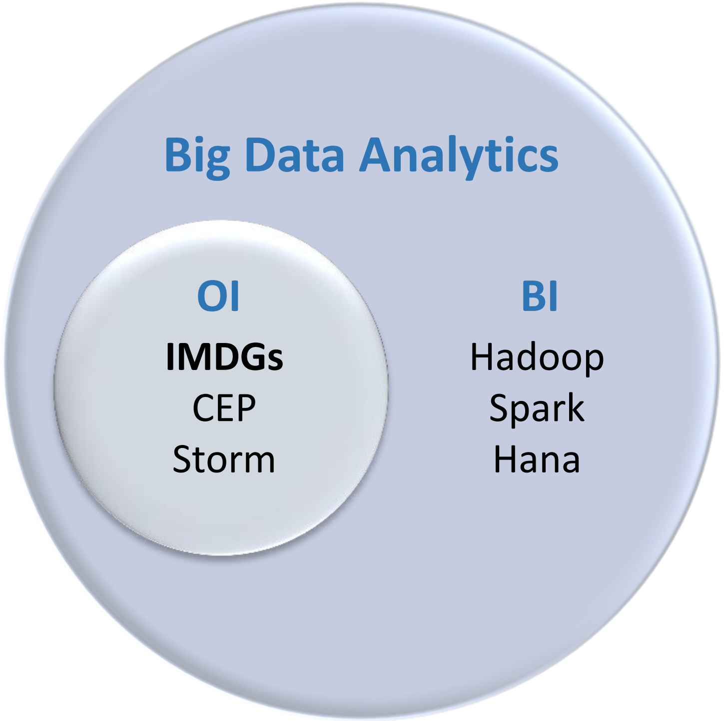
Gigabytes to terabytes

In-memory storage

Sub-second to seconds

Best uses:

- Tracking live data
- Immediately identifying trends and capturing opportunities
- Providing immediate feedback



Business Intelligence

Batch

Static data sets

Petabytes

Disk storage

Minutes to hours

Best uses:

- Analyzing warehoused data
- Mining for long-term trends

Example: Enhancing Cable TV Experience

ScaleOut Software

- **Goals:**
  - Make real-time, personalized upsell offers
  - Immediately respond to service issues
  - Detect and manage network hot spots
  - Track aggregate behavior to identify patterns, e.g.:
    - Total instantaneous incoming event rate
    - Most popular programs and # viewers by zip code

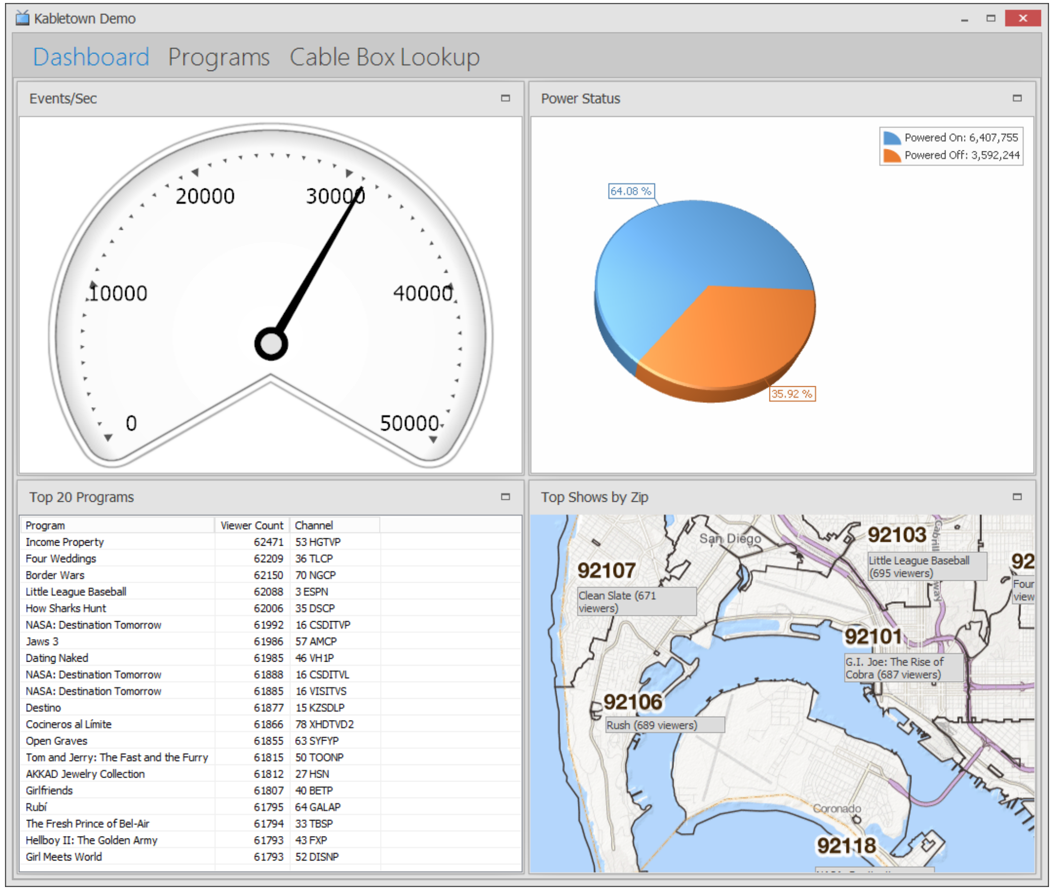
- **Requirements:**
  - Track events from 10M set-top boxes with 25K events/sec (2.2B/day)
  - Correlate, cleanse, and enrich events per rules (e.g. ignore fast channel switches, match channels to programs)
  - Be able to feed enriched events to recommendation engine within 5 seconds
  - Immediately examine any set-top box (e.g., box status) & track aggregate statistics

The Result: An OI Platform

ScaleOut Software

Based on a simulated workload for San Diego metropolitan area:

- Continuously correlates and cleanses telemetry from 10M simulated set-top boxes (from synthetic load generator)
- Processes more than 30K events/second
- Enriches events with program information every second
- Tracks aggregate statistics (e.g., top 10 programs by zip code) every 10 seconds

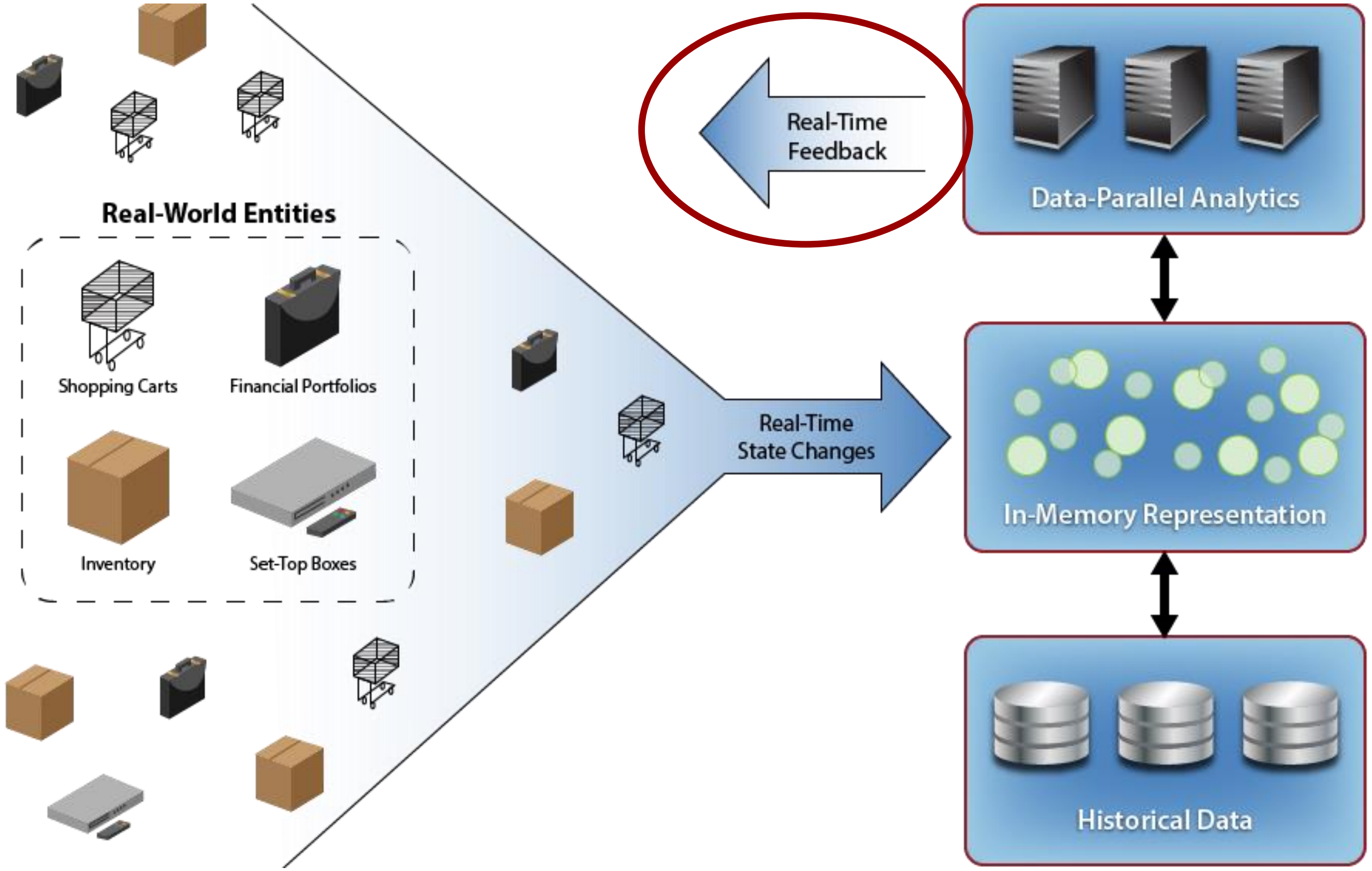


Real-Time Dashboard

Using an IMDG to Implement OI

ScaleOut Software

- IMDG models and tracks the state of a “live” system.
- IMDG analyzes the system’s state in parallel and provides real-time feedback.



IMDG analyzes in-memory data with integrated compute engine.

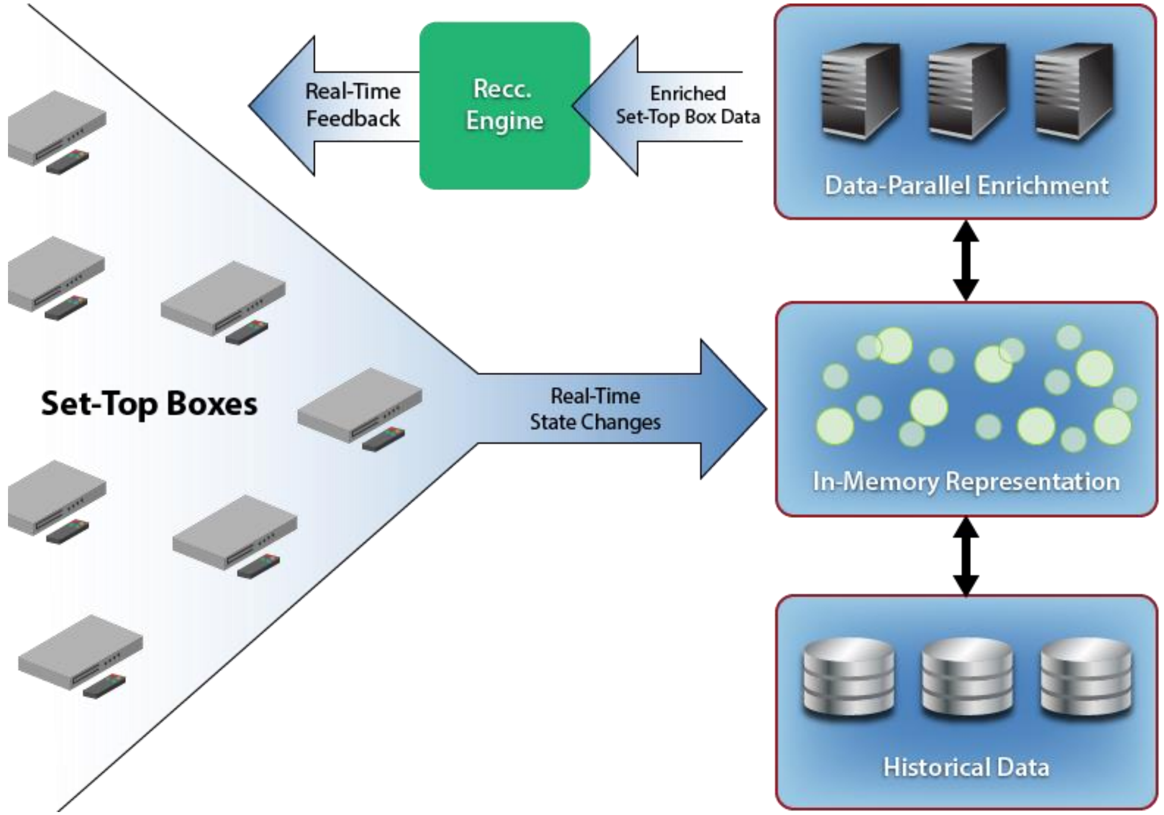
IMDG tracks live system’s state with an in-memory, object-oriented model.

IMDG enriches in-memory model from disk-based, historical data.

Example: Tracking Set-TopBoxes

ScaleOut Software

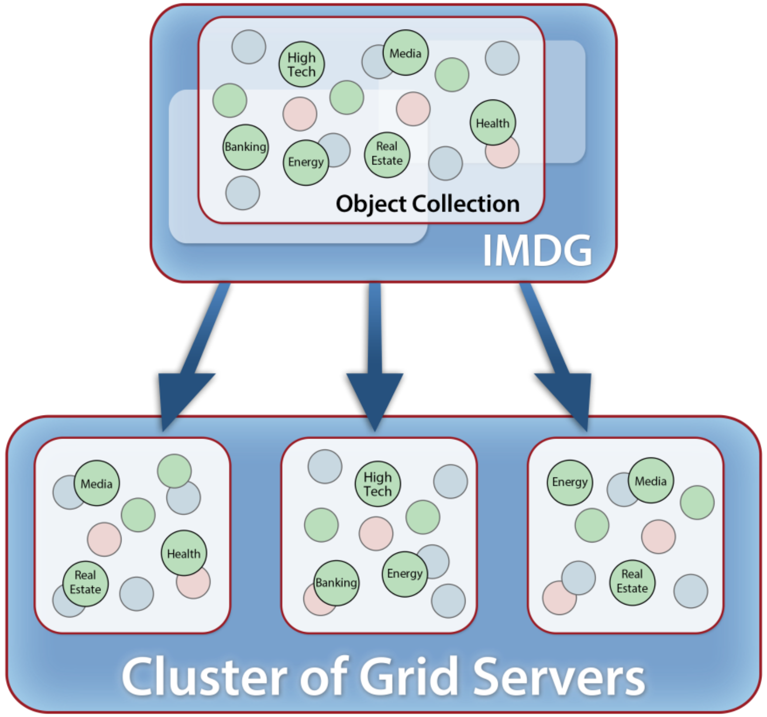
- Each set-top box is represented as an object in the IMDG
- Object holds raw & enriched event streams, viewer parameters, and statistics
- IMDG captures incoming events by updating objects
- IMDG uses data-parallel computation to:
  - immediately enrich box objects to generate alerts to recommendation engine, and
  - continuously collect and report global statistics



The Foundation: In-Memory Data Grids

ScaleOut Software

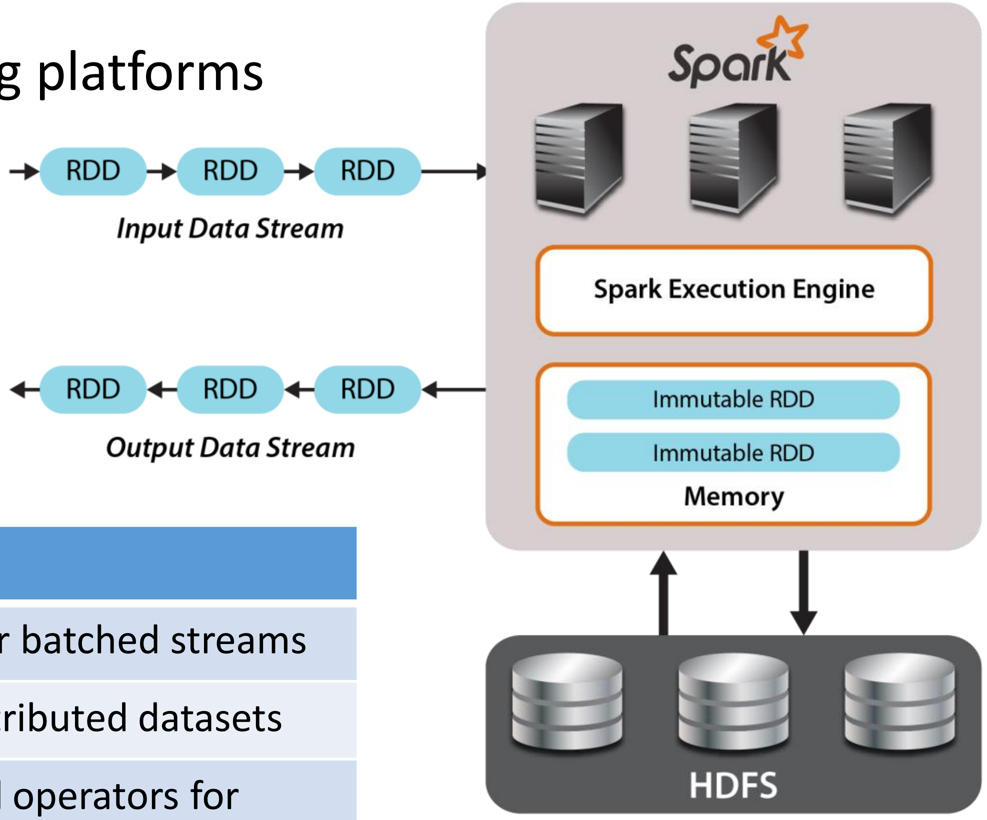
- In-memory data grid (IMDG) provides scalable, hi av storage for live data:
  - Designed to manage business logic state:
    - Object-oriented collections by type
    - Create/read/update/delete APIs for Java/C#/C++
    - Parallel query by object properties
    - Data shared by multiple clients
  - Designed for transparent scalability and high availability:
    - Automatic load-balancing across commodity servers
    - Automatic data replication, failure detection, and recovery
- IMDGs provide ideal platform for operational intelligence:
  - Easy to track live systems with large workloads
  - Appropriate availability model for production deployments



Comparing IMDGs to Spark

ScaleOut Software

- On the surface, both are surprisingly similar:
  - Both designed as scalable, in-memory computing platforms
  - Both implement data-parallel operators
  - Both can handle streaming data
- But there are key differences that impact use for operational intelligence:

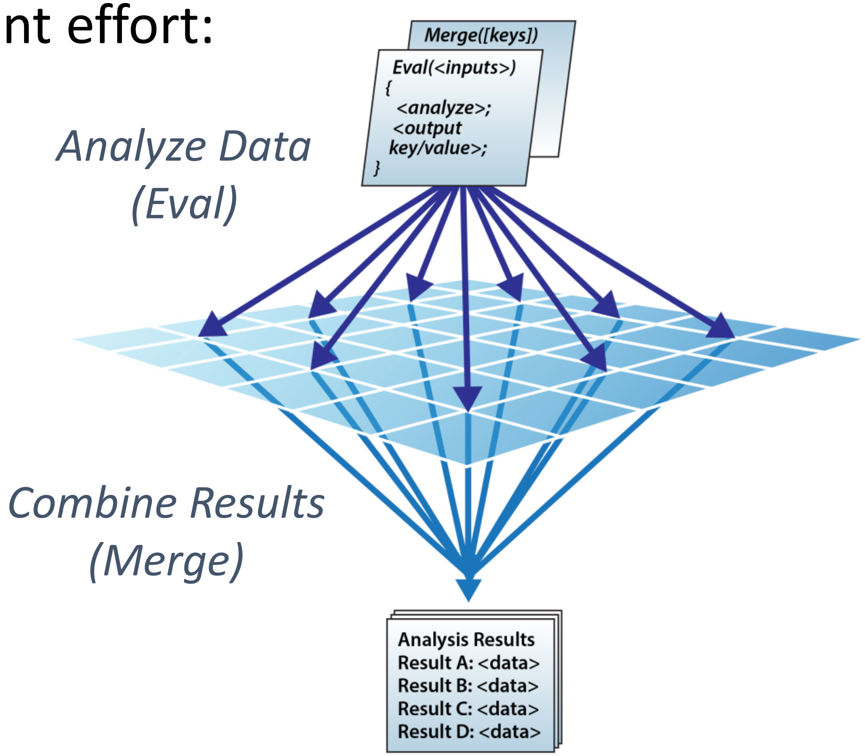


	IMDGs	Spark
Best use	Live, operational data	Static data or batched streams
In-memory model	Object-oriented collections	Resilient distributed datasets
Focus of APIs	CRUD, eventing, data-parallel computing	Data-parallel operators for analytics
High availability tradeoffs	Data replication for fast recovery	Lineage for max performance

Data-Parallel Computing on an IMDG

ScaleOut Software

- IMDGs provide powerful, cost-effective platform for data-parallel computing:
  - Enable integrated computing with data storage:
    - Take advantage of cluster’s commodity servers and cores.
    - Avoid delays due to data motion (both to/from disk and across network).
  - Leverage object-oriented model to minimize development effort:
    - Easily define data-parallel tasks as class methods.
    - Easily specify domain as object collection.

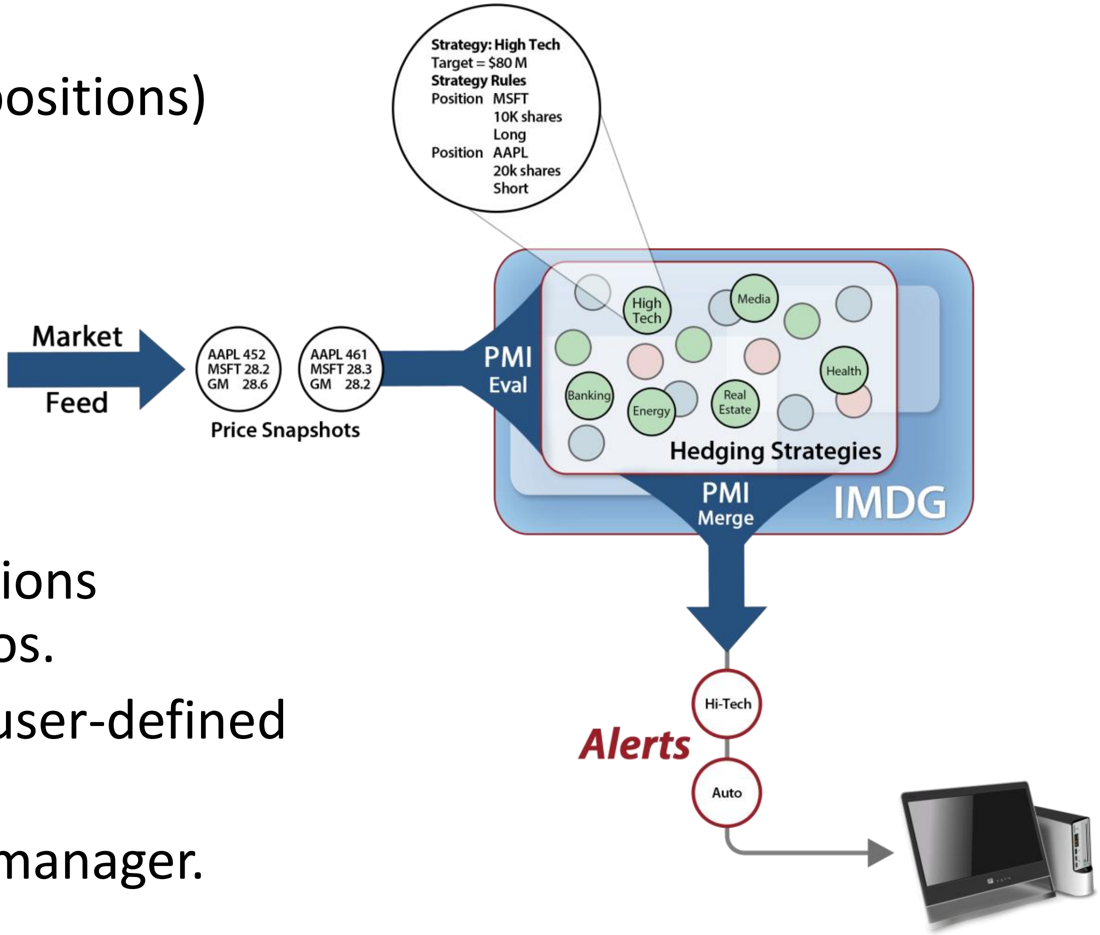


- Example: “Parallel Method Invocation” (PMI):
  - Object-oriented version of standard HPC model
  - Runs class methods in parallel across cluster.
  - Selects objects using parallel query of obj. collection.
  - Serves as a platform for implementing MapReduce and other data-parallel operators

PMI Example: OI in Financial Services

ScaleOut Software

- **Goal**: track market price fluctuations for a hedge fund and keep portfolios in balance.
- **How**:
  - Keep portfolios of stocks (long and short positions) in object collection within IMDG.
  - Collect market price changes in one-second snapshots.
  - Define a method which applies a snapshot to a portfolio and optionally generates an alert to rebalance.
  - Perform repeated parallel method invocations on a selected (i.e., queried) set of portfolios.
  - Combine alerts in parallel using a second user-defined method.
  - Report alerts to UI every second for fund manager.



Defining the Dataset

ScaleOut Software

- Simplified example of a portfolio class (Java):
  - Note: some properties are made query-able.
  - Note: the evalPositions method analyzes the portfolio for a market snapshot.

```java
public class Portfolio {
    private long         id;
    private Set<Stock> longPositions;
    private Set<Stock> shortPositions;
    private double       totalValue;
    private Region       region;
    private boolean      alerted;  // alert for trading

    @SossIndexAttribute       // query-able property
    public double getTotalValue() {…}
    @SossIndexAttribute       // query-able property
    public Region getRegion() {…}

    public Set<Long> evalPositions(MarketSnapshot ms) {…};
}
```

Strategy: High Tech
Target = $80 M
Strategy Rules
Position MSFT
10K shares
Long
Position AAPL
20k shares
Short

Defining the Parallel Methods

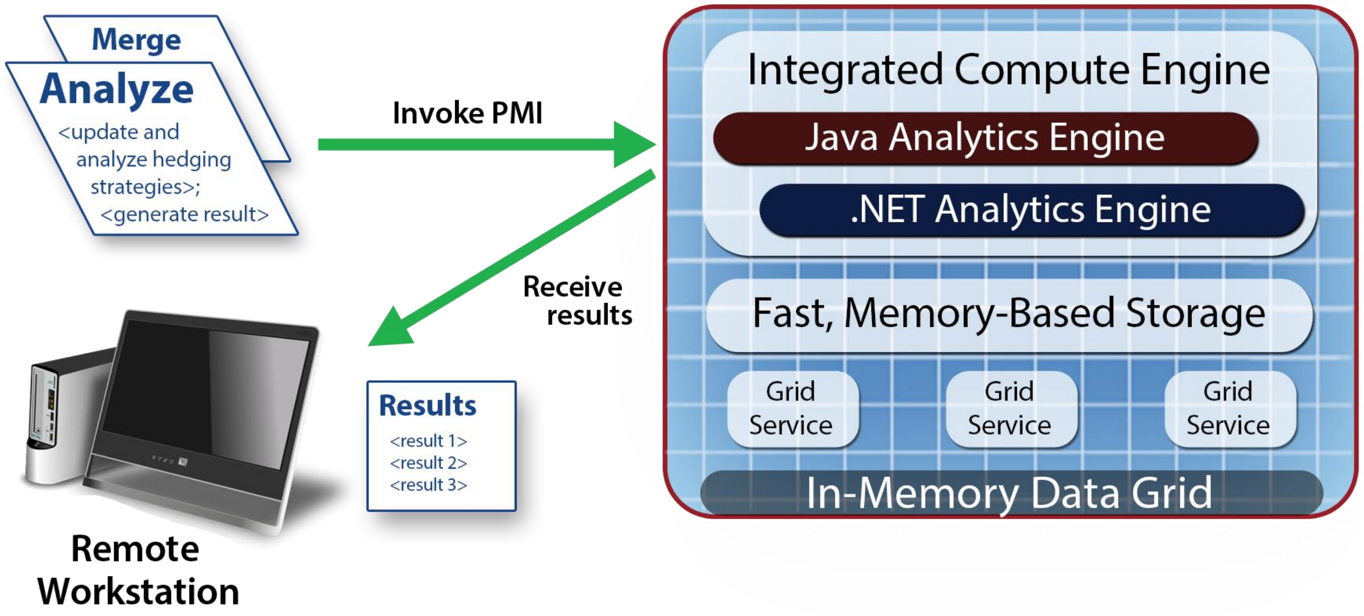
ScaleOut Software

- Implement PMI interface to define methods for analyzing each object and for merging the results:

```
public class PortfolioAnalysis implements
    Invokable<Portfolio, MarketSnapshot, Set<Long>>
{
  public Set<Long> eval(Portfolio p, MarketSnapshot ms)
                  throws InvokeException {

      // update portfolio and return id if alerted:
      return p.evalPositions(ms);
  }

  public Set<Long> merge(Set<Long> set1, Set<Long> set2)
                         throws InvokeException {
      set1.addAll(set2);
      return set1;  // merged set of alerted portfolio ids
  }}
```

Running the Analysis

- PMI can be run from a remote workstation.
- IMDG ships code and libraries to cluster of servers:
  - Execution environment can be pre-staged for fast startup.
- In-line execution minimizes scheduling time.
  - Avoids batch scheduling delays.
- PMI automatically runs in parallel across all grid servers:
  - Uses software multicast to accelerate startup.
  - Passes market snapshot parameter to all servers.
  - Uses all servers and cores to maximize throughput.



Spawning the Compute Engine

ScaleOut Software

- First obtain a reference to the IMDG's object collection of portfolios:

```
NamedCache pset = CacheFactory.getCache(“portfolios");
```

- Create an "invocation grid," a re-usable compute engine for the application:
  - Spawns a JVM on all grid servers and connects them to the in-memory data grid.
  - Stages the application code on all JVMs.
  - Associates the invocation grid with an object collection.

```
InvocationGrid grid = new InvocationGridBuilder("grid")
          .addClass(DependencyClass.class)
          .addJar("/path/to/dependency.jar")
          .setJVMParameters("-Xmx2m")
          .load();

pset.setInvocationGrid(grid);
```

Invoking the PMI

ScaleOut Software

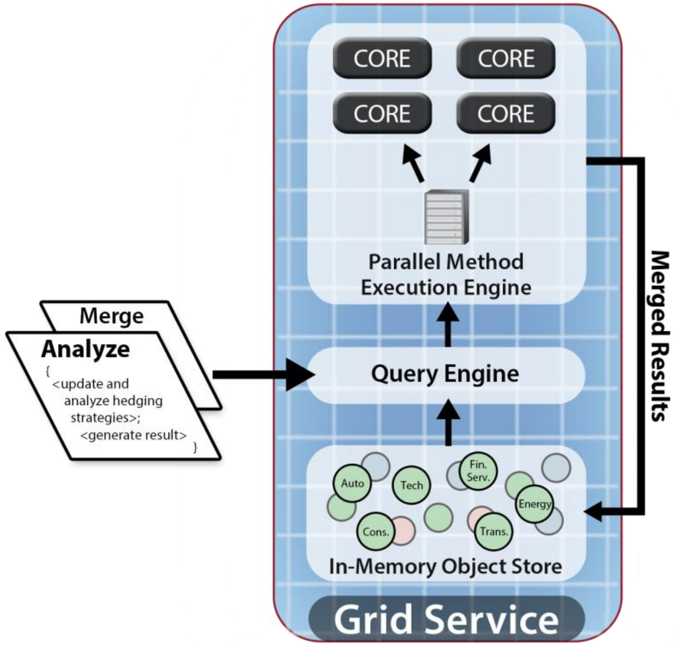
- Run the PMI on a queried set of objects within the collection:
  - Multicasts the invocation and parameters to all JVMs.
  - Runs the data-parallel computation.
  - Merges the results and returns a final result to the point of call.

```
InvokeResult alertedPortolios = pset.invoke(
    PortfolioAnalysis.class,
    Portfolio.class,
    and(greaterThan(“totalValue”, 1000000),  // query spec
        equals(“region”, Region.US)),
    marketSnapshot,                          // parameters
    ...
    );

System.out.println("The alerted portfolios are" +
                   alertedPortfolios.getResult());
```

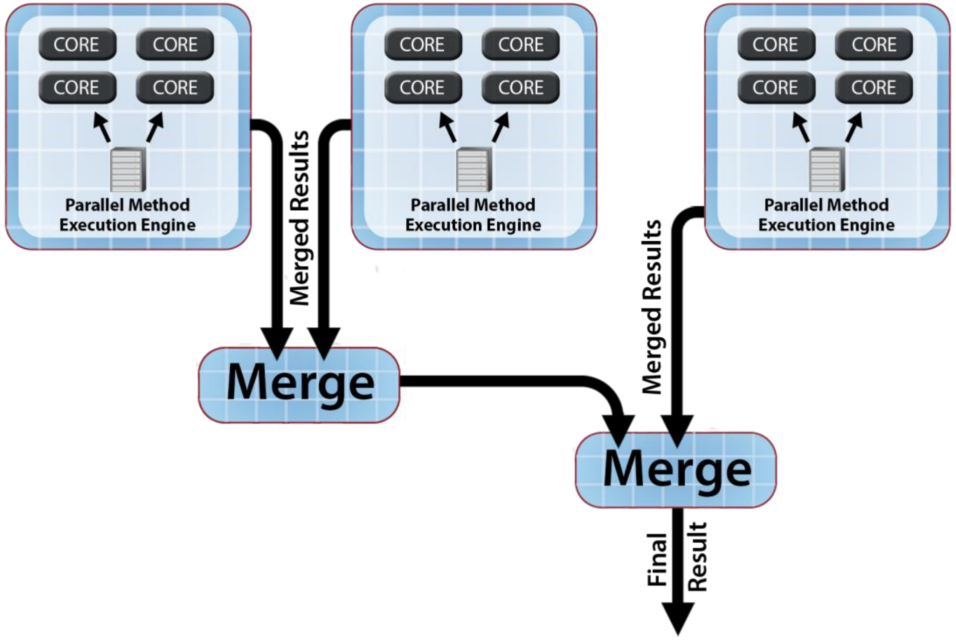
Execution Steps

ScaleOut Software

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



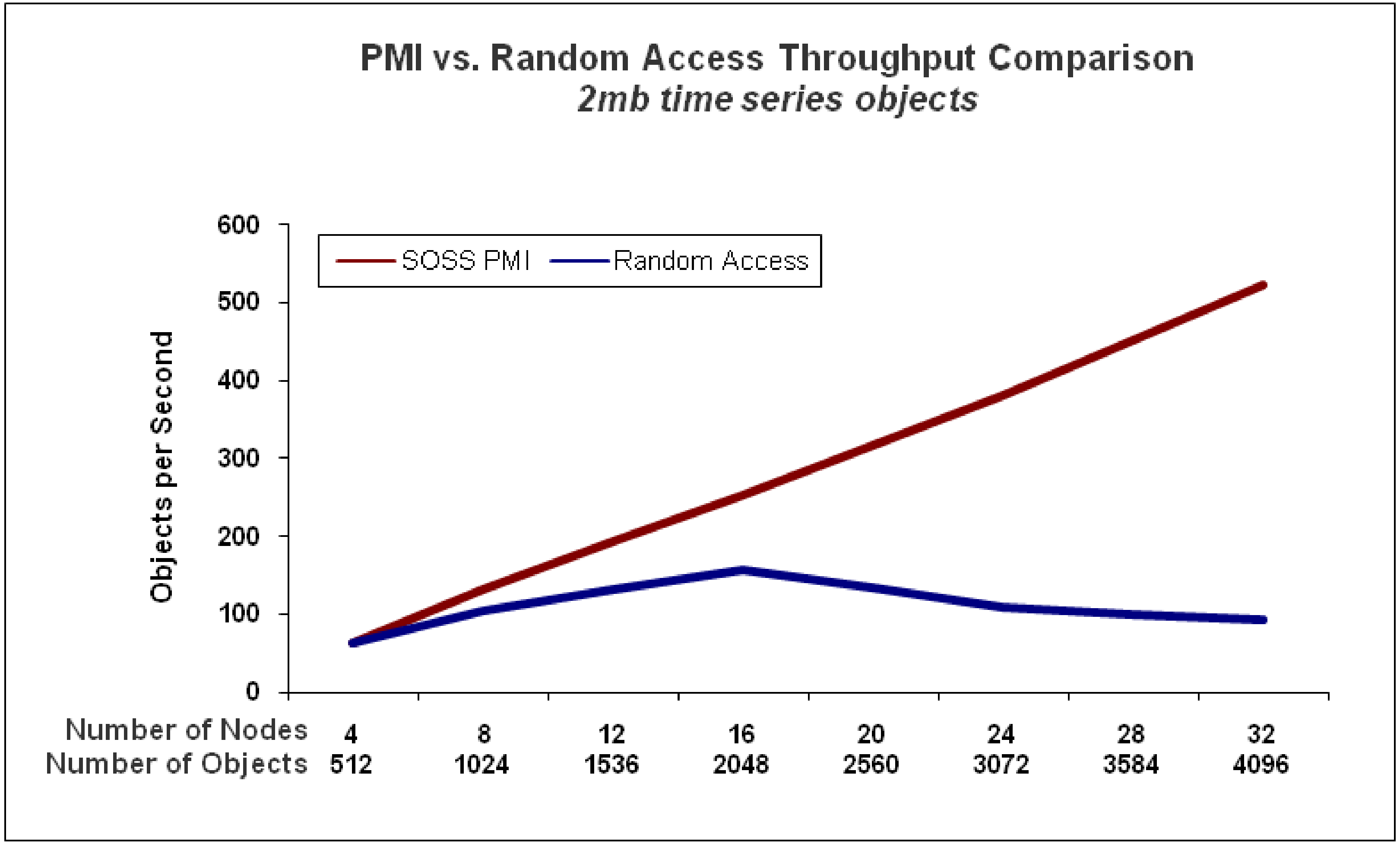
- **Merge phase**: all servers perform distributed merge to create final result:
  - Merge runs in parallel to minimize completion time.
  - Returns final result object to client.



Importance of Avoiding Data Motion

ScaleOut Software

- Local data access enables linear throughput scaling.
- Network access creates a bottleneck that limits throughput.

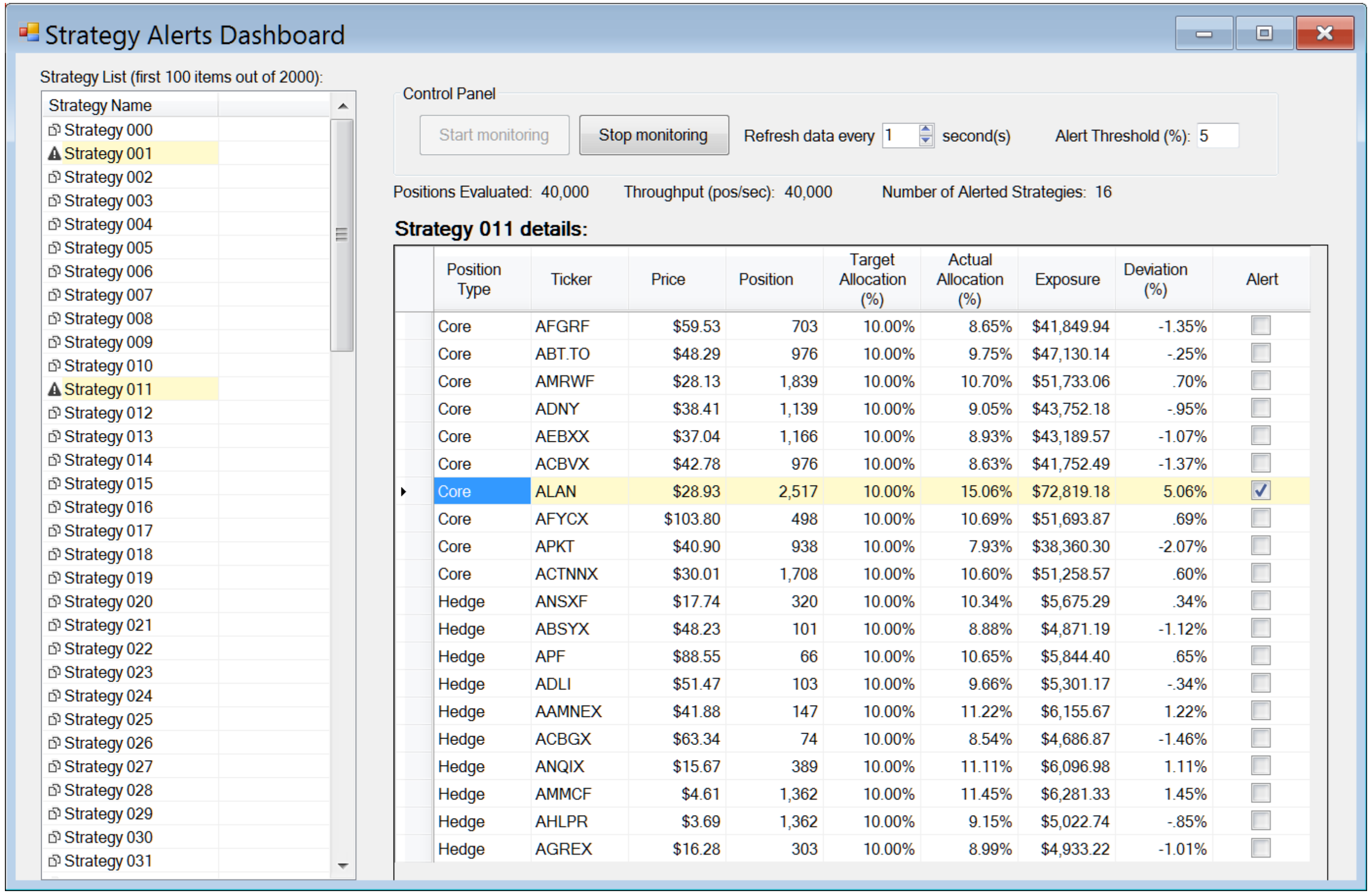


Outputting Continuous Alerts to the UI

ScaleOut Software

- PMI runs every second; it completes in **350 msec**. and immediately refreshes UI.

- UI alerts trader to portfolios that need rebalancing.
- UI allows trader to examine portfolio details and determine specific positions that are out of balance.
- **Result: in-memory computing delivers operational intelligence.**



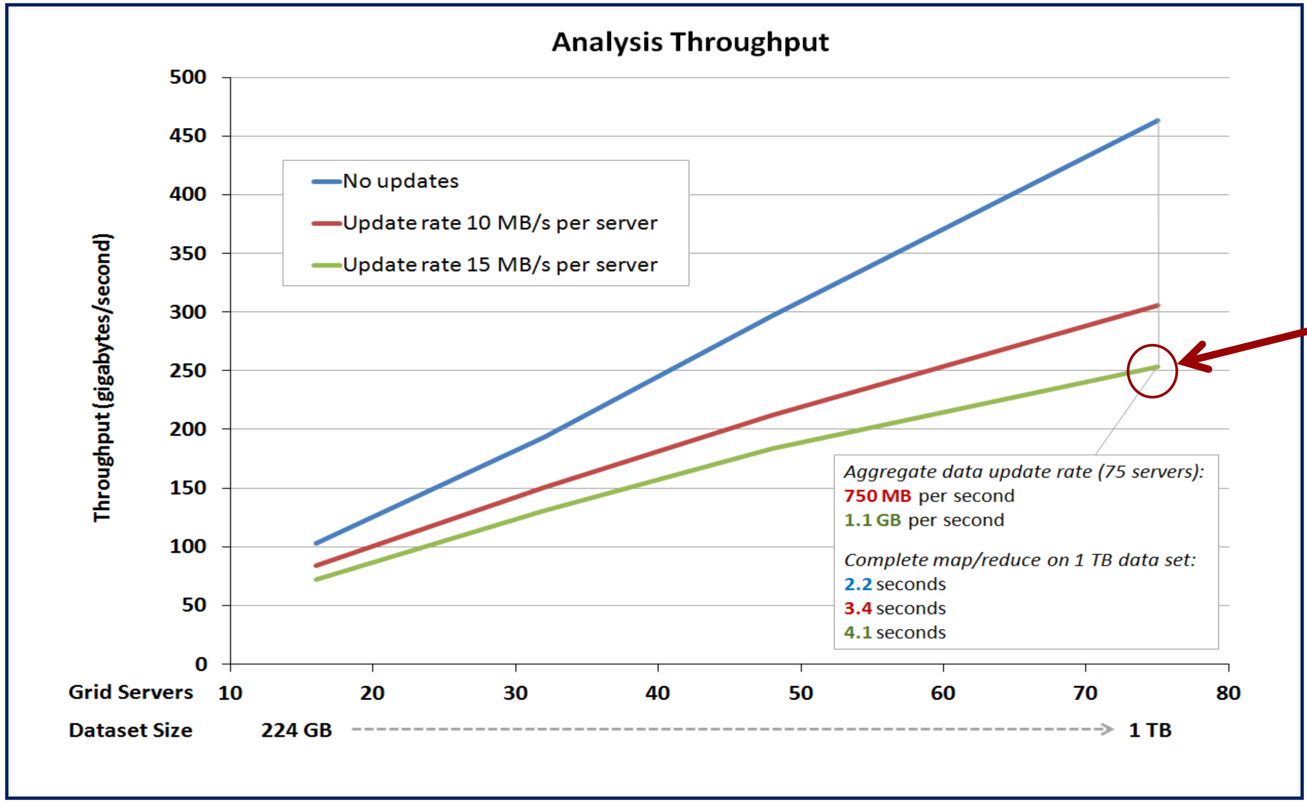
Demonstration Video:
[Comparison of PMI to Apache Hadoop](https://www.youtube.com/watch?v=8JTsqp_-Gnw)

https://www.youtube.com/watch?v=8JTsqp_-Gnw

PMI Scales for Large In-Memory Datasets

ScaleOut Software

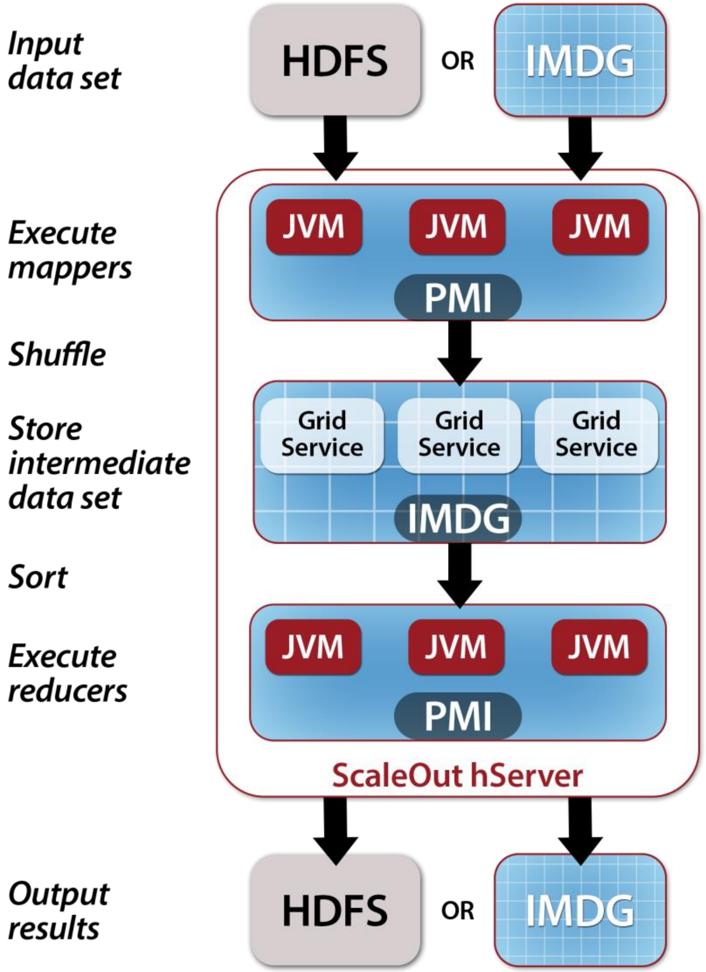
- Measured a similar financial services application (back testing stock trading strategies on stock histories)
- Hosted IMDG in Amazon EC2 using 75 servers holding **1 TB** of stock history data in memory
- IMDG handled a continuous stream of updates (**1.1 GB/s**)
- Results: analyzed 1 TB in **4.1 seconds** (250 GB/s).
- Observed linear scaling as dataset and update rate grew.



Using PMI to Implement MapReduce for OI

ScaleOut Software

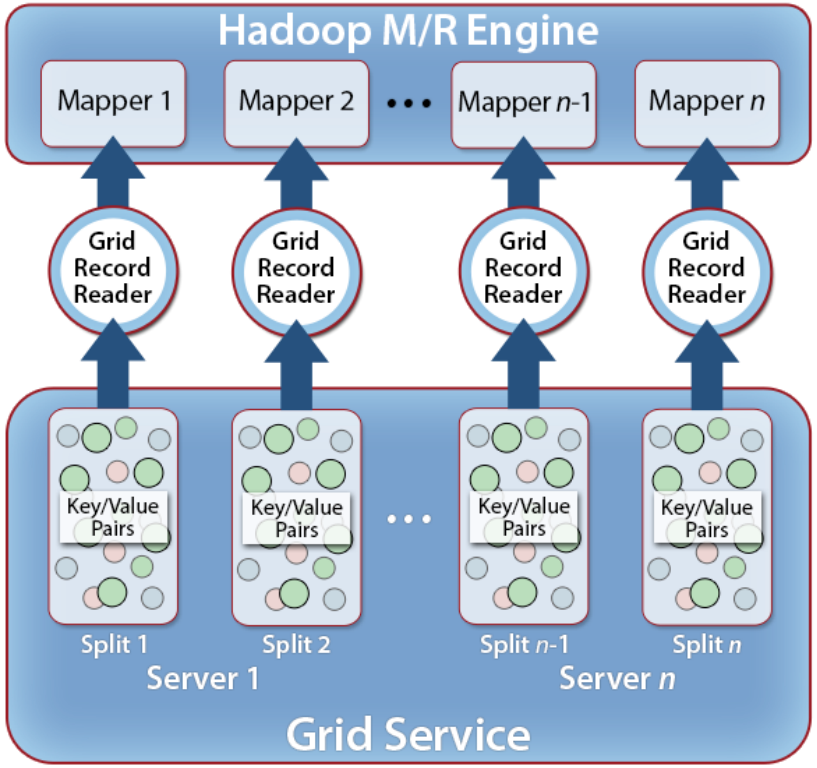
- PMI serves as foundational platform for MapReduce and other parallel operators.
- Implement MapReduce with two PMI phases:
  - Runs standard Hadoop MapReduce applications.
  - Data can be input from either the IMDG or an external data source.
    - Works with any input/output format.
  - IMDG uses PMI phases to invoke the mappers and reducers.
    - Eliminates batch scheduling overhead.
  - Intermediate results are stored within the IMDG.
    - Minimizes data motion in shuffle phase.
    - Allows optional sorting.
  - Note: output of a single reducer/combiner optionally can be globally merged.



MapReduce for OI Requires New Data Model

ScaleOut Software

- IMDGs historically implement a feature-rich data model:
  - Efficiently manages large objects (KBs-MBs).
  - Supports object timeouts, locking, query by properties, dependency relationships, etc.
- MapReduce typically targets very large collections of small key/value pairs:
  - Does *not* require rich object semantics.
  - Does require efficient storage (minimum metadata) and highly pipelined access.
- **Solution**: a new IMDG data model for MapReduce:
  - Uses standard Java named map APIs for access.
  - MapReduce uses standard input/output formats.
  - Stores data in chunks and pipelines to/from engine.
  - Automatically defines splits for mappers and holds shuffled data for reducers.



Optimizing MapReduce for OI: simpleMR

ScaleOut Software

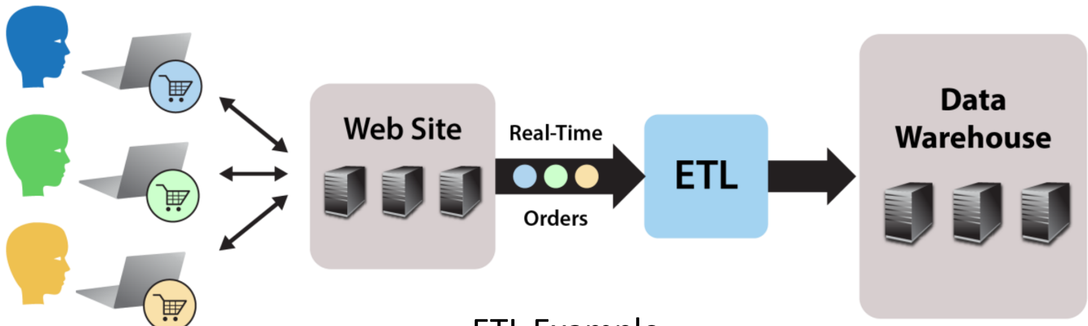
- Integrate in-memory named map with MapReduce to minimize execution time.
- Use new API (**simpleMR** in Java, C#) to simplify apps and remove Hadoop dependencies.

```
public class Mapper : IMapper<int, string, string, int>
{
    void IMapper<int, string, string, int>.Map(int key,
        string value, IContext<string, int> context)
    {
        ...
        context.Emit(Encoding.ASCII.GetString(...), 1);
}}

inputMap  = new NamedMap<int, string>("Input_Map");
outputMap = new NamedMap<string, int>("Output_Map");

inputMap.RunMapReduce<string, int, string, int>(outputMap,
             new Mapper(), new Combiner(), new Reducer(), ...);
```

Integrating OI and BI in the Data Warehouse

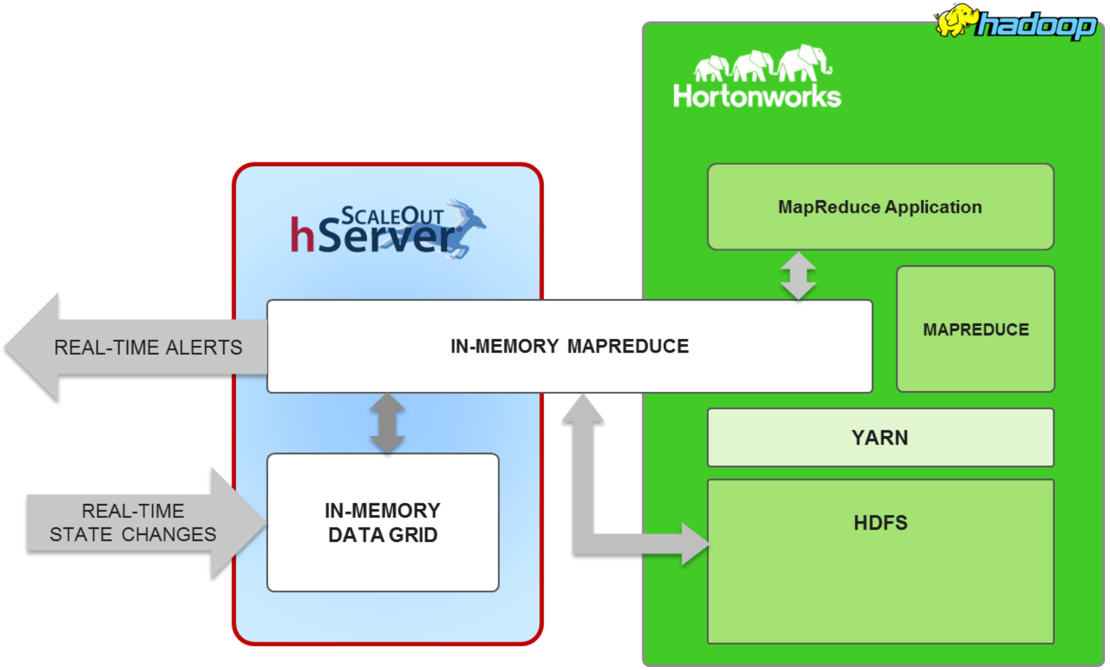
ScaleOut Software

- In-memory data grids can add value to a BI platform, e.g.:
  - Transform live data and store in HDFS for analysis.
  - Provide immediate feedback to live system pending deep analysis.



ETL Example

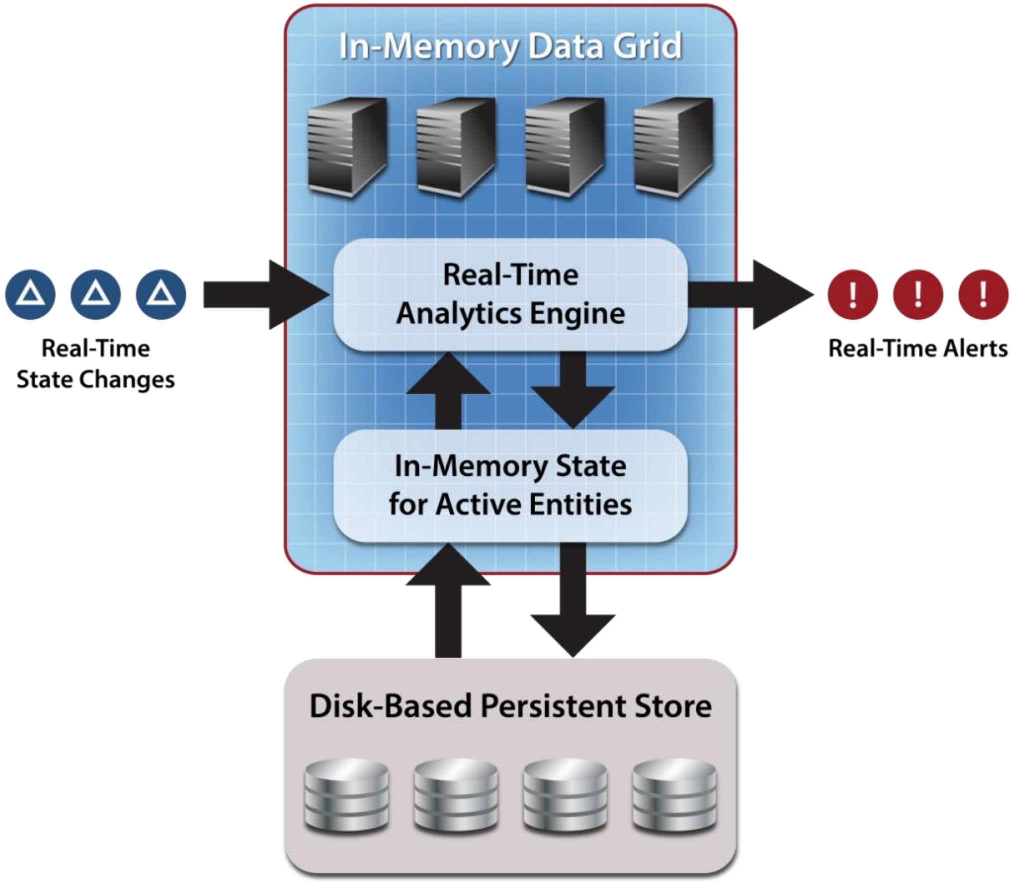
- Using YARN, an IMDG can be directly integrated into a BI cluster:
  - The IMDG holds fast-changing data.
  - YARN directs MapReduce jobs to the IMDG.
  - The IMDG can output results to HDFS.



Recap: In-Memory Computing for OI

ScaleOut Software

- **Online systems need operational intelligence** on "live" data for immediate feedback.
  - Creates important new business opportunities.
- Operational intelligence can be implemented using **standard data-parallel computing techniques**.
- **In-memory data grids provide an excellent platform** for operational intelligence:
  - Model and track the state of a "live" system.
  - Implement high availability.
  - Offer fast, data-parallel computation for immediate feedback.
  - Provide a straightforward, object-oriented development model.



www.scaleoutsoftware.com