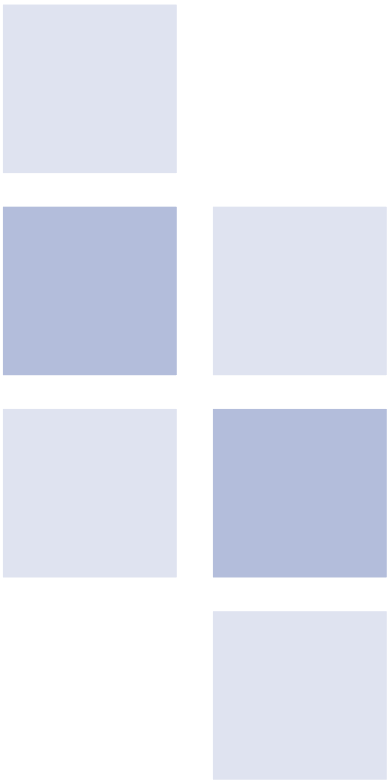


SCALEOUT SOFTWARE

Using In-Memory Computing to Simplify Big Data Analytics

by Dr. William Bain, ScaleOut Software, Inc.

The "big data" revolution is upon us, fed by the need in both the public and private sectors to quickly analyze large datasets for important patterns and trends. With big data analysis, ecommerce vendors can target customers more precisely, financial analysts can quickly spot changing market conditions, manufacturers can tune logistics planning, and the list goes on. They all need powerful, easy to use analysis tools to maintain a competitive edge.

Parallel computing techniques, such as "map/reduce," have opened the door to dramatically reducing analysis times and are now proliferating in platforms such as open source Hadoop. However, the history of parallel computing since the mid-1980s has demonstrated that the complexity of these techniques can be daunting, impeding adoption and inhibiting widespread use. How can this complexity be managed so that the benefits of parallel data analysis are fully realized?

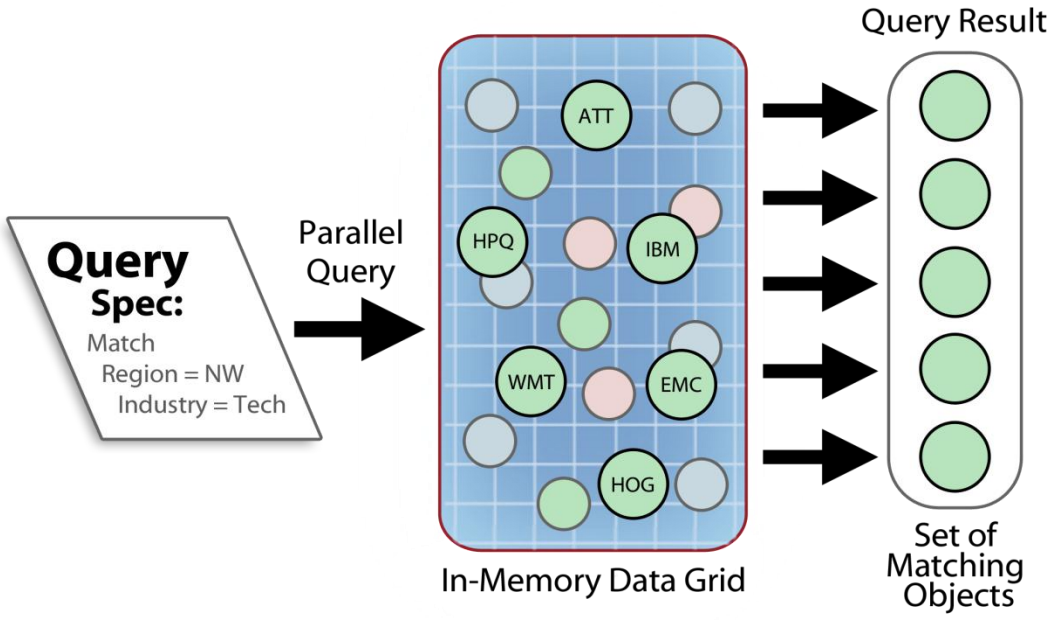
The technology of in-memory data grids (IMDGs) offers important breakthroughs in addressing the challenge. Having proven their value in storing fast-changing application data and scaling application performance, IMDGs have integrated map/reduce analytics into the grid to provide parallel analysis and enable near real-time decision making. By incorporating straightforward, well understood programming techniques with a fast execution engine, IMDGs lower the learning curve, shorten the development cycle, and reduce analysis times. This article explains how IMDGs reduce complexity and lower the barrier to big data analysis.

The Starting Point: Object-Based Data Storage and Query

IMDGs store fast-changing data within a middleware software tier shared across a pool of application servers. This enables applications to seamlessly scale access to fast-changing data as servers are added to handle growing workloads. To maximize scalability, IMDGs automatically load-balance data across servers on which the grid is hosted. They also redundantly store data on multiple servers to ensure high availability in case a server or

network link fails. Additional capabilities, including eventing and distributed locking, make IMDGs a powerful data storage platform.

The key to the IMDG's ease of use is its integration into the object-oriented data model used by business logic, typically written in Java or C#. IMDGs store data as a collection of logically related objects which are accessible either by specifying an identifying key or by querying object properties. For example, an ecommerce Web site can store a collection of shopping cart objects identified by user IDs and queryable based on properties such as dollar value or time of last change. A financial application analyzing trading strategies can store a large collection of stock histories, one for each stock symbol being analyzed and containing the price history of the stock; each object can be queried by properties such as sector, market cap, or other criteria. The following diagram illustrates the use of parallel query for selecting stock history data based on region and industry sector:



Parallel Query of an IMDG

Storing data as object-based collections offers the IMDG several advantages. First, it provides a straightforward means for identifying data to be analyzed. In contrast, applications running on file-based analysis platforms such as Hadoop must read and parse files to generate key/value pairs for analysis; this adds complexity to the application. The IMDG's object-oriented storage and query mechanism eliminates the need for applications

to implement this code and can quickly scan a large data set for objects whose properties match a query specification. Moreover, it also enables the IMDG to load-balance the objects in memory across a pool of grid servers. This eliminates the overhead of file I/O otherwise required to input data for analysis, significantly speeding up processing.

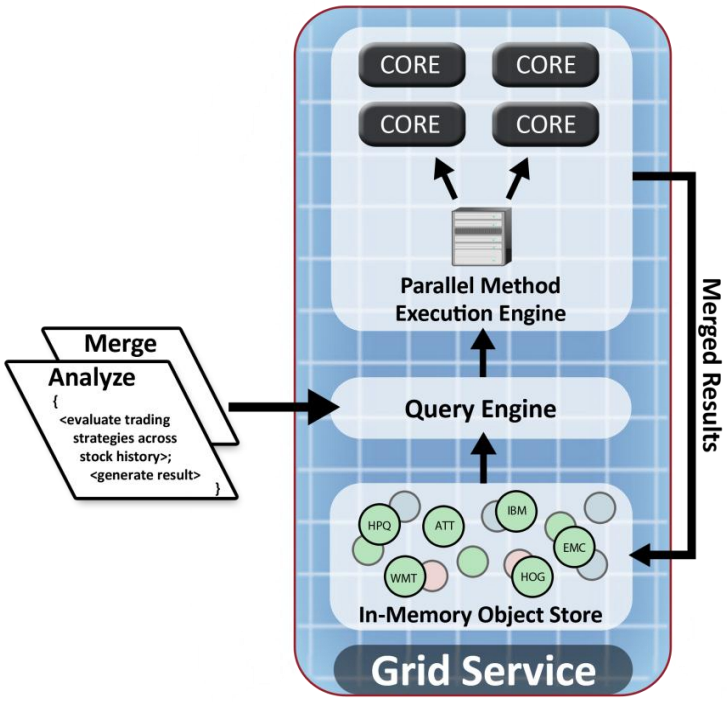
The Next Step: Object-Based Data Analysis

Because IMDGs store and query data as a collection of logically related objects, applications can easily create analysis code using methods defined for the object's type. This simplifies the application by taking full advantage of the language's typing system to identify code to be executed during parallel analysis. The developer typically defines two methods, an "analyze" method that analyzes all queried objects and a "merge" method that combines the results generated by the analyze methods. Developing these methods requires no special knowledge of parallel programming because they are written exactly as if they were to be sequentially executed on an object collection hosted on a single workstation.

The IMDG can fully leverage this object-oriented view of data by automatically deploying these "analyze" and "merge" methods on the grid servers for execution. For example, ScaleOut Analytics Server™ from ScaleOut Software lets developers pre-stage these methods and any supporting libraries on the grid prior to starting the parallel analysis.

As illustrated in the following diagram, the analysis proceeds in three steps on each grid server in parallel:

- The grid server's query engine identifies locally stored objects for analysis based on selected properties of interest from the query. These objects are held in memory on the grid servers.
- The grid server feeds the locally selected objects to a local multi-core execution engine for analysis using the pre-staged analyze method. Execution automatically uses all processors and cores for maximum performance. (Data never moves between servers.)
- The execution engine combines all analysis results by running the user's merge method, again taking full advantage of multi-core speedup. It stores the merged result with the server's in-memory object store.



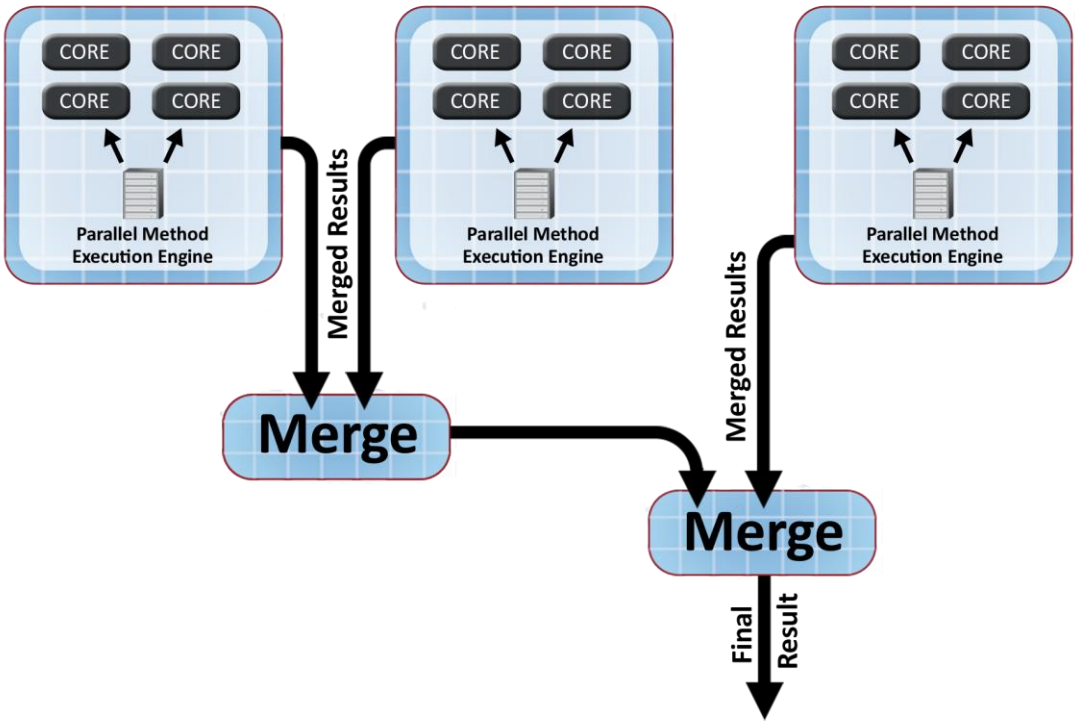
Performing a Parallel Method Invocation on a Grid Server

To ensure maximum performance, all grid servers automatically perform the above analysis in parallel. This leaves merged results in each grid server's memory-based store. How can these results be efficiently combined to create a final result?

The Final Step: Efficient, Object-Based Merging of Results

A great deal of complexity in parallel data analysis can be found in the merging of results from the analysis phase. While the map/reduce programming pattern offers powerful semantics, its inherent complexity adds to development time and requires careful tuning. Many useful analyses can be performed using a subset of this pattern which combines results in a pairwise manner following a well understood technique from parallel computing called "global data aggregation." This avoids the need for applications to define key/value spaces for results and implement reduction algorithms to process them. For example, when analyzing stock trading strategies, the set of results generated by analyzing all queried stock symbols can be combined pairwise to generate a final result; simultaneous access to all results, while allowed in fully general data reduction, is not required for merging results here.

IMDGs can take advantage of this straightforward merging technique to simplify application development and automatically merge results across grid servers. Once the application defines a merge method for combining two results from the analysis phase, the IMDG stores all result objects as a collection within the grid and merges them into a final result to be delivered back to the user. As described above, the IMDG's parallel execution engine combines all results generated within a single grid server. It then automatically combines results across all grid servers using a distributed combining tree, as shown in the following diagram:



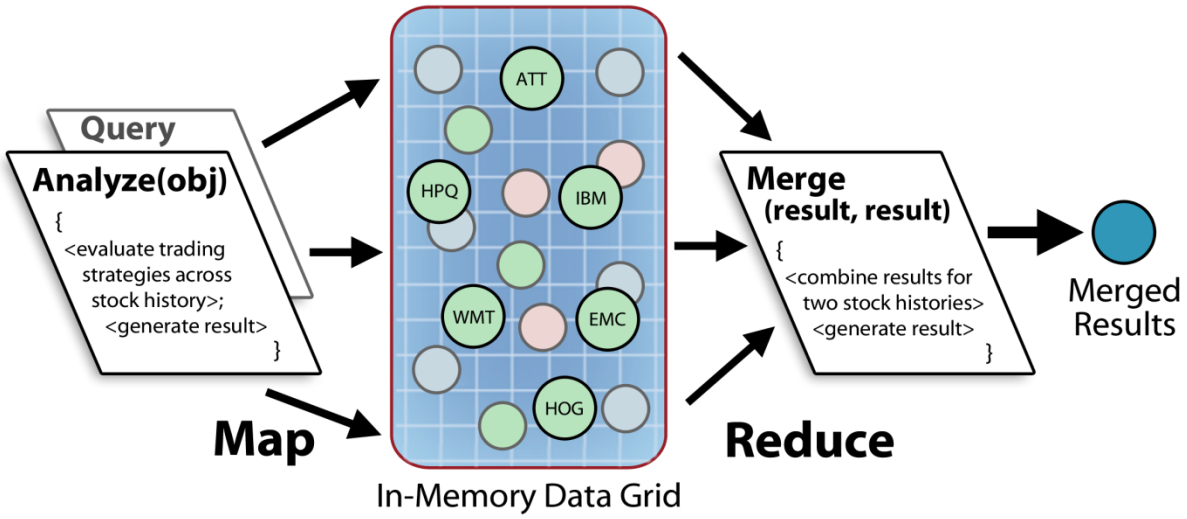
Merging PMI Results Across all Grid Servers in an IMDG

To ensure the fastest possible merging of results, some IMDGs, such as ScaleOut Analytics Server, use a binary merge tree that asynchronously combines results as they are generated by the grid servers. This minimizes the overall time to merge results across the grid servers, ensuring excellent scalability. More important, this approach avoids the need for application developers to write additional code and tune performance. It also delivers the final result to the user in memory instead of requiring that a set of results be retrieved from the file system.

Pulling It All Together: Big Data Simplified – and Faster

As we have seen, IMDGs can take the complexity out of parallel data analysis by hosting memory-based data as object collections and performing a simplified form of map/reduce analysis using familiar object-oriented techniques. The application developer selects data for analysis by querying properties of interest and then writes an "analyze" method to independently analyze each selected object along with a "merge" method to combine the results in a pairwise manner. The IMDG takes care of the rest, including automatically shipping the code to the grid servers, querying and analyzing the objects in parallel, and combining the results.

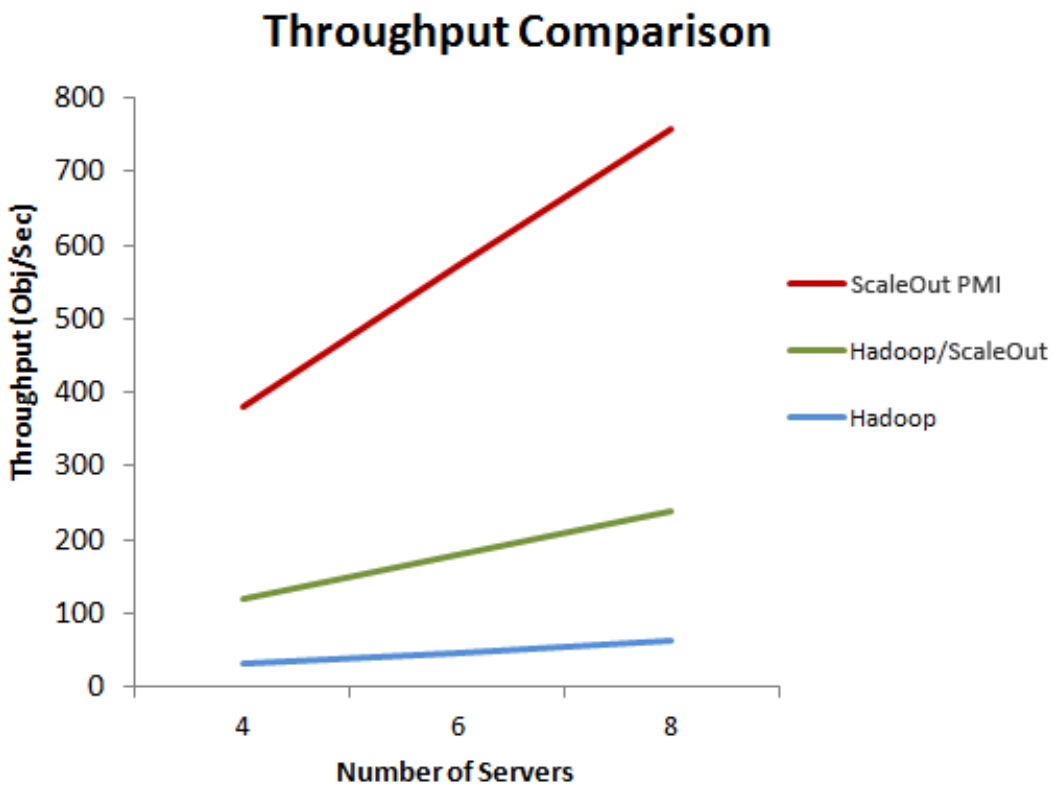
The following diagram illustrates a map/reduce analysis of stock trading strategies across a set of stock histories held in the IMDG. A parallel query selects stocks for analysis, and the IMDG analyzes the stocks and merges the results using the supplied methods:



Example of Parallel Method Invocation for Stock "Back Testing"

This simplified approach to map/reduce eliminates the complexity of popular map/reduce platforms which require the developer to define key/value spaces and implement record readers and reducers. It also eliminates the need to deploy and tune a complex parallel computing infrastructure.

However, this simplicity does not imply lower performance. For datasets that fit in memory, IMDGs offer substantial performance gains over file-based map/reduce platforms. By avoiding file I/O and data motion to combine results and feed reducers, they eliminate substantial overhead and boost throughput. To measure the effect of these optimizations, performance measurements were made for the stock trading analysis described above using ScaleOut Analytics Server as the number of stock histories and grid servers was proportionally increased. As the graph below illustrates, the IMDG delivered linearly scalable throughput (shown as the red line in the graph). An alternative implementation of this application was measured using Hadoop's map/reduce environment. Hadoop provided linear scaling with about 16X lower throughput (shown as the blue line in the graph) due to significant overhead introduced by file I/O, combining, reducing, and batch scheduling.



Throughput Comparison of an IMDG to Hadoop

To see the impact of file I/O, data was staged in the IMDG instead of the Hadoop file system (HDFS). Hadoop's throughput was increased by about 6X (shown as the green line).

The remaining difference is largely due to file I/O between the map and reduce phases. Although Hadoop is working to reduce this additional file I/O, significant data motion and lower performance due to the use of multiple reducers is unavoidable.

Many Datasets are Memory-Based

The complexity and overhead of today's disk-based map/reduce platforms can be too high for applications which must quickly analyze fast-changing data sets measured in hundreds of gigabytes or terabytes. (Estimates by some analysts indicate that as much as sixty percent of data sets are smaller than ten terabytes.) In many situations, an answer in hours or minutes is not acceptable. For example, an e-commerce Web site may need to monitor online shopping carts to see which products are selling; a financial services company might need to hone its equity trading strategy as it optimizes its response to fast-changing market conditions; an airline may need to continuously analyze its reservations system for delayed flights or lost bags.

Many useful datasets fit into an IMDG's scalable memory storage and can benefit from the simple, object-based analysis techniques and fast analysis times offered by the IMDG's integration of data storage with an object-oriented parallel analysis engine. IMDGs open up the opportunity to analyze fast-changing data with near real-time performance, giving businesses important new tools for maximizing competitiveness.

In Summary

With the ever increasing explosion in data for analysis and the need for fast insights on emerging trends, IMDGs offer a highly attractive platform for hosting map/reduce analysis. By simplifying the development model and automating execution, IMDGs shorten the learning curve in developing analysis codes and eliminate the tuning steps required by more complex platforms. Because IMDGs run the analysis on data already staged in memory and load-balanced across grid servers, file I/O is eliminated and data motion is minimized. IMDGs also provide the infrastructure needed to automatically run analysis code on all grid servers in parallel and then combine the results with minimum latency. The net result is that by using an IMDG, application developers can easily analyze fast-

changing, memory-based data and discover data patterns and trends that are vital to a company's success.

Dr. William L. Bain *is Founder and CEO of ScaleOut Software, Inc. Bill has a Ph.D. in electrical engineering/parallel computing from Rice University, and he has worked at Bell Labs research, Intel, and Microsoft. Bill founded and ran three start-up companies prior to joining Microsoft. In the most recent company (Valence Research), he developed a distributed Web load-balancing software solution that was acquired by Microsoft and is now called Network Load Balancing within the Windows Server operating system. Dr. Bain holds several patents in computer architecture and distributed computing. As a member of the Seattle-based Alliance of Angels, Dr. Bain is actively involved in entrepreneurship and the angel community.*

■ ■ ■

www.scaleoutsoftware.com