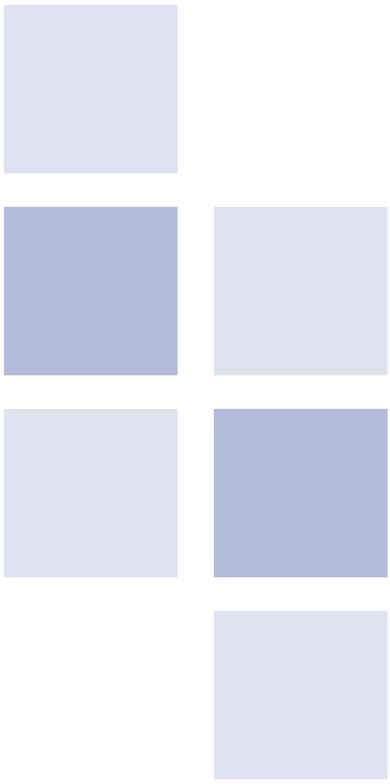


SCALEOUT SOFTWARE

Accelerating Hadoop MapReduce Using an In-Memory Data Grid

By David L. Brinker and William L. Bain, ScaleOut Software, Inc.

Hadoop has been widely embraced for its ability to economically store and analyze large data sets. Using parallel computing techniques like MapReduce, Hadoop can reduce long computation times to hours or minutes. This works well for mining large volumes of historical data stored on disk, but it is not suitable for gaining real-time insights from live operational data. Still, the idea of using Hadoop for real-time data analytics on live data is appealing because it leverages existing programming skills and infrastructure – and the parallel architecture of Hadoop itself.

As competitive pressures build, companies are increasingly pushed to find new ways to identify and capture business opportunities. Business intelligence (BI) has been around for years, and it continues to yield important business insights. However, as data volumes continue to explode, new methods of storage and analysis are needed. Hadoop has emerged as a popular way to handle both the storage and analysis of these increased volumes of data. Using Apache Hadoop or one of the many commercial distributions, companies can store petabytes of data in the Hadoop Distributed File System (HDFS) and analyze it using MapReduce. The efficiency gains have been impressive.

Even so, BI is only part of the story. As business velocity increases, companies have turned to real-time analytics on live, operational data to extend their competitive barriers. For example, e-commerce vendors need to target customers while they shop, financial risk managers need to quickly react to changing market conditions, manufacturers need to continuously analyze sensor data to tune production processes – the list of opportunities continues to grow. Although Hadoop's parallel architecture can accelerate analytics, when it comes to fast-changing data, Hadoop's batch processing and disk overheads are prohibitive. Other approaches, such as complex event processing (CEP), use highly specialized programming models and require incremental investments in infrastructure, skills, training, and software.

What if you could leverage the expertise of your existing Hadoop developers, as well as your current computing infrastructure, to perform real-time analytics on live, operational data? This would give you the means to quickly identify and act on "perishable" business opportunities while you take full advantage of your investment in Hadoop training and infrastructure. This paper describes how real-time analytics using Hadoop can be performed by combining an in-memory data grid (IMDG) with an integrated, stand-alone Hadoop MapReduce execution engine. This new technology delivers fast results for live data and also accelerates the analysis of large, static data sets.

The First Step: Scalable, In-Memory Data Storage for Live Data

For the last several years, IMDGs have been widely deployed to host live, fast-changing data within operational systems. Because of their low access latency, scalable capacity and throughput, and integrated high availability, they have proved to be useful in a wide range of

applications. For example, IMDGs are used to hold e-commerce shopping carts, financial market and trading data, airline reservations, etc.

Typically organized as a middleware software tier, IMDGs automatically store and load-balance data across an elastic cluster of 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.) An IMDG's cluster can seamlessly scale its capacity by adding servers to handle growing workloads. These servers also provide the computational capacity needed for performing real-time analytics. This enables the IMDG to incorporate a Hadoop MapReduce engine for analyzing its live data using standard Hadoop techniques, as illustrated in figure 1 below.

IMDGs need flexible storage mechanisms to handle widely varying demands on the data that they store. In many cases, they host complex objects with rich semantics to support features such as property-oriented query, dependencies, timeouts, pessimistic locking, and synchronized access from remote IMDGs. Other applications need to store and analyze huge numbers of very small objects, such as sensor data or tweet streams. Interestingly, Hadoop MapReduce applications have typically been used for large populations of simple objects.

To handle these divergent storage requirements and efficiently use memory and network resources, IMDGs can employ multiple storage APIs, such as the *Named Cache* and *Named Map* APIs shown in the figure. In both named caches and named maps, applications can create, read, update, and delete objects to manage live data. This gives the application developer the choice to store and analyze heavyweight objects with rich metadata or lightweight objects with highly optimized storage, depending on the type of data being analyzed.

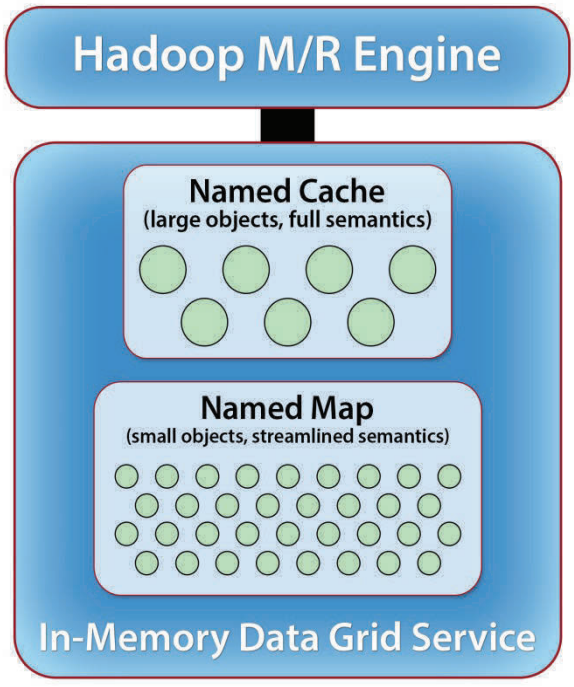


Figure 1

Because IMDGs typically are integrated into operational systems which process live data, they can immediately access in-memory data for analysis and provide real-time feedback to optimize operations and identify exceptional conditions. Integrating a Hadoop MapReduce engine into an IMDG minimizes analysis time because it avoids data motion during processing by analyzing data in place. In contrast, hosting data in the Hadoop Distributed File System (HDFS) requires data to be moved to and from disk, increasing both access latency and I/O overhead and significantly lengthening analysis time.

Running Hadoop MapReduce in the IMDG

When run using one of the popular, open source distributions, Hadoop MapReduce introduces numerous overheads that extend analysis times to minutes (and even hours for very large data sets). These overheads are prohibitive when running real-time analytics that must return results in milliseconds or seconds. For example, in financial trading systems, every second lost can materially affect the return on a trading decision.

Advanced IMDGs include parallel computing capabilities that overcome many of these limitations and enable the semantics of Hadoop MapReduce to be emulated (and optimized), yielding the same results as standard Hadoop MapReduce only much faster. This enables standard Hadoop MapReduce code to be run with essentially no changes at all. There is no need to learn anything new. If your MapReduce program is designed to analyze both live data and historical data, it could be used in both the IMDG-based real-time environment as well as standard Hadoop.

How can an IMDG reduce Hadoop's execution time to enable real-time analytics? The first step is to eliminate batch scheduling overhead, which can take upwards of 30 seconds using Hadoop's standard batch scheduler. Instead, IMDGs can pre-stage a Java-based execution environment on all grid servers and reuse it for multiple analyses. This execution environment consists of a set of Java Virtual Machines (JVMs), one on every server within the cluster alongside each grid service process. These JVMs form the IMDG's Hadoop MapReduce engine, as shown in figure 2. Also, the IMDG can automatically deploy all necessary executable programs and libraries for the execution of MapReduce across the JVMs, greatly reducing startup time down to milliseconds.

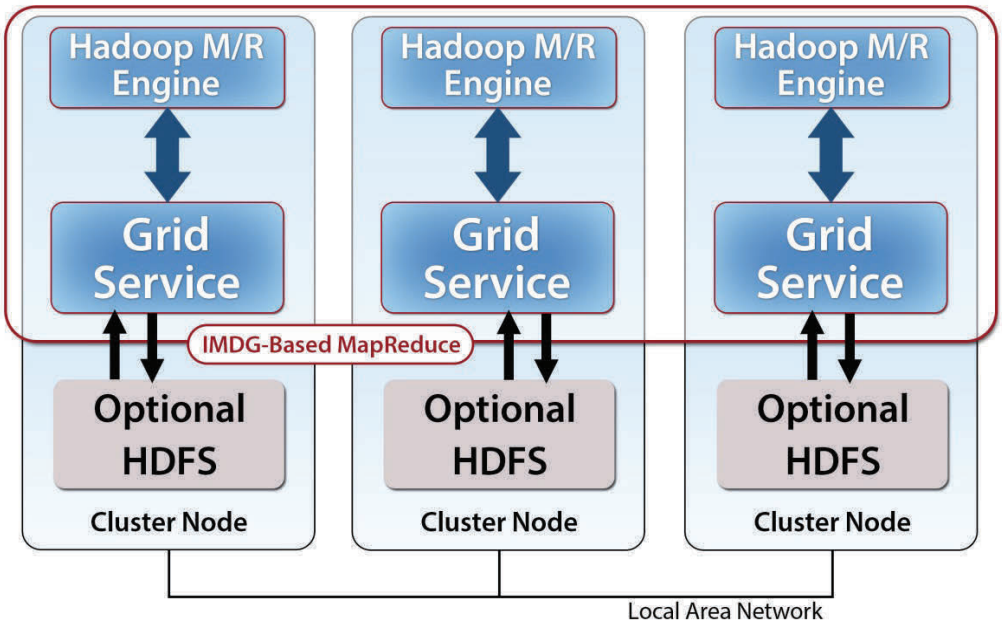


Figure 2

The next step in reducing MapReduce analysis time is to eliminate as much data motion as possible. Because an IMDG hosts fast-changing data in memory, MapReduce applications can input data directly from the grid (and output results back to the grid). This speeds up analysis by avoiding delays in accessing secondary storage. Also, because the execution engine is integrated with the IMDG, key/value pairs hosted within the IMDG can be efficiently read into the execution engine to minimize access time, as shown in figure 3. A special record reader, called a *grid record reader*, can be used to automatically pipeline the transfer of key/value pairs from the IMDG's in-memory storage into the mappers. Its input format automatically creates splits of the specified input key/value collection to avoid network overhead when retrieving key/value pairs on all grid servers. Likewise, a *grid record writer* enables pipelined output of results from Hadoop's reducers back to IMDG storage. This technique also can be employed to minimize data motion between the mappers and the reducers by storing intermediate data within the IMDG's memory-based storage.

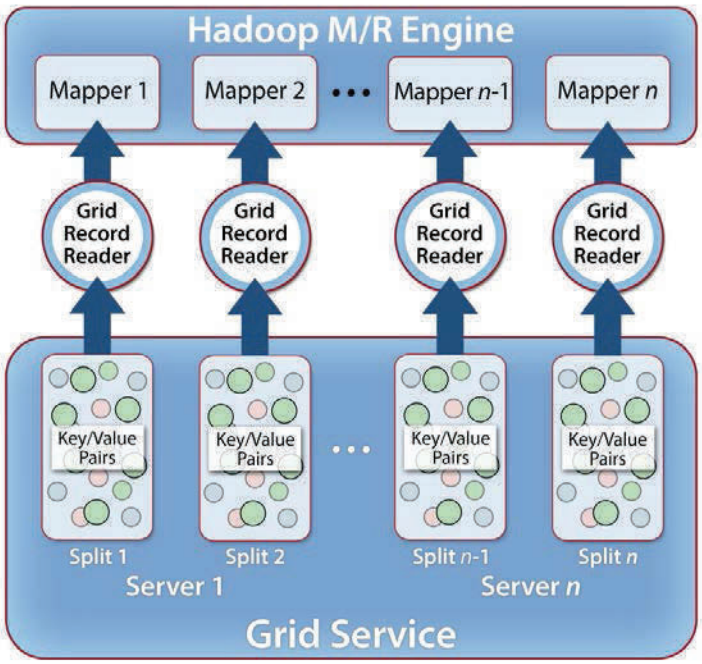


Figure 3

Integration of a Hadoop MapReduce execution engine within an IMDG offers additional performance optimizations and simplifications:

- To reduce data motion between the mappers and reducers, the execution engine can store intermediate results within the grid instead of in the local file system. These results also can be efficiently pipelined to and from grid storage to accelerate MapReduce performance.
- The use of grid-based storage for the input data set and results enables the IMDG to automatically determine several Hadoop parameters, including the number of splits and partitions. This simplifies application development and tuning.

- Because the IMDG hosts its own MapReduce engine, sorting can be made optional to accelerate applications that do not need it. Many uses in real-time analytics do not need sorting.
- The MapReduce engine's performance can be scaled linearly just by adding more servers to the cluster. Because the IMDG automatically load-balances stored data, the analysis workload is automatically redistributed to additional JVMs within the cluster.

Some applications may elect to input static data sets from HDFS (as shown in figure 2) to take advantage of the fast MapReduce execution environment provided by the IMDG. In this case, the IMDG should be installed on the same cluster of servers as HDFS to minimize network overhead. Many uses of Hadoop MapReduce in real-time analytics do not need HDFS and are not subjected to the complexity of installing one of the Hadoop distributions. These applications can take advantage of the simplicity offered by the IMDG's integrated MapReduce engine.

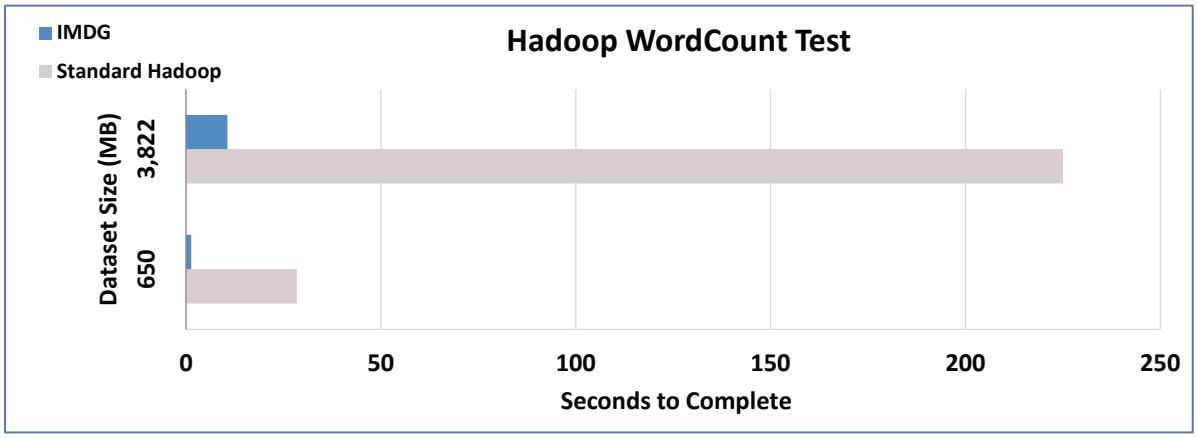
Real-Time Performance

To demonstrate the performance advantage of the IMDG's integrated MapReduce engine, measurements were made of the familiar Hadoop WordCount sample application. This program was run both on the standard Apache Hadoop distribution and on an IMDG that included a built-in Hadoop MapReduce execution engine. In this test, ScaleOut hServer™ from ScaleOut Software was used as the IMDG and MapReduce engine.

ScaleOut hServer integrates a Hadoop MapReduce execution engine with its in-memory data grid. Its open source Java API library includes several components: a Hadoop MapReduce execution engine, which runs MapReduce jobs without using Hadoop job trackers or task trackers, and input/output formats to pass data between the IMDG and a MapReduce application, allowing the application to use the IMDG as a data source and/or result storage.

The benchmark test was run on four Intel Core 2 Quad CPU servers configured with 16GB memory. When running in the standard Apache Hadoop distribution, the application input data from HDFS; when running on ScaleOut hServer, it input data from the IMDG. Two tests were run with different data set sizes (650MB and 3.8GB), and ScaleOut hServer demonstrated a 21X performance gain on both, as shown in figure 4 below.

Figure 4



Beyond just providing a significant speed reduction in analysis time, the IMDG also allows the input data set to be updated while the MapReduce analysis is in progress. Using the standard Hadoop MapReduce distribution, live updates are not possible since data in HDFS can only be appended and not updated.

Beyond Live Data Analysis

We have seen how an IMDG can enable Hadoop MapReduce to analyze live, operational data. IMDGs also can provide significant advantages for analyzing static data sets. Here are some additional ways an IMDG can be employed to accelerate Hadoop MapReduce applications and simplify code development.

Stream Data in from HDFS

By using alternative input formats, IMDG-based MapReduce applications can be connected to HDFS or other data sources and input data sets for processing by the MapReduce engine. (See figure 5.) Likewise, output from MapReduce applications can be sent directly to HDFS or other storage systems instead of to the IMDG.

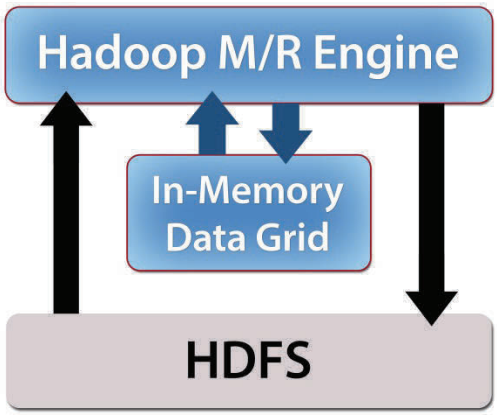


Figure 5

As data streams in from HDFS, it is fed to the mappers, processed, and then output as an intermediate data set which is shuffled and then sent to the reducers. Some IMDGs (such as ScaleOut hServer) can store this intermediate data set within the IMDG to minimize data motion. In this case, as long as this intermediate data set fits within the IMDG's memory, the IMDG's MapReduce engine can process very large data sets that otherwise would not fit within the IMDG.

HDFS Distributed Cache

To decrease the access time for reading data into Hadoop from HDFS, the input data set can be cached within the IMDG's memory-based storage. Some IMDGs provide a distributed caching feature that speeds access times by capturing data from HDFS or other data sources during MapReduce processing. This feature is intended for use with data sets which fit within the memory of the IMDG.

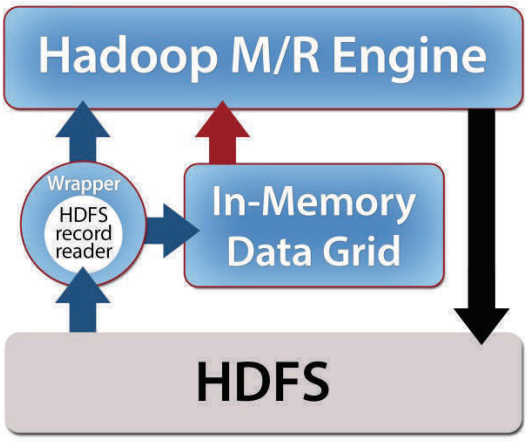


Figure 6

Here's how it works. Using a simple change to the Hadoop program, a special input format

"wraps" the application's input format and intercepts key/value pairs input by its record reader. This wrapper automatically stores the key/value pairs in the IMDG while forwarding them to the application's original record reader. On subsequent MapReduce runs, the IMDG verifies that the HDFS data set has not changed and then supplies key/value pairs to the mappers directly from the IMDG. Figure 6 shows how this wrapper is used.

Rapid Code Development

Development of Hadoop MapReduce code can be time consuming and error prone. Just the management of the Hadoop software stack is challenging, not to mention lengthy execution times during debugging. Using an IMDG with an integrated Hadoop MapReduce engine can simplify development and shorten the time required to create a tested application. This approach avoids the need to install and configure the standard Hadoop distribution just to develop and test a MapReduce application. It also dramatically shortens execution times, especially if the input data set is loaded into the IMDG. This means that you can rapidly iterate on your MapReduce code until you're getting the results you expect before taking your code into production.

Fast Iterative Runs

Using an IMDG also enables rapid, iterative "what-if" analyses of static data held in the grid. This can be useful in applications, such as financial or process modeling, which require running multiple runs on the same data set. For example, fast MapReduce execution times enable stock trading strategies to be repeatedly tested and honed by multiple simulations across price histories held in memory. Likewise, e-commerce recommendation engines can be developed and tested in a fraction of the time required by standard Hadoop.

Summary

Using an IMDG with an integrated Hadoop MapReduce engine opens the door to real-time analytics on live, operational data. By storing data in memory, eliminating scheduling overhead, and reducing data motion, an IMDG can dramatically shorten analysis time from minutes to seconds, demonstrating a 20x speedup in benchmark tests. And when the data sets get bigger, the performance of the IMDG's Hadoop MapReduce engine automatically scales as servers are added to the cluster. This enables Hadoop MapReduce to be employed in operational systems which manage fast-changing data and need to quickly identify patterns and trends, even as the data changes.

The IMDG's integrated MapReduce engine also eliminates the need to install, configure, and manage a full Hadoop distribution. Developers can write and run standard Hadoop MapReduce applications in Java, and these applications can be executed stand-alone by the execution engine. This enables Hadoop skill sets to be easily transferred to real-time analytics, while making full use of existing investments in infrastructure for managing fast-changing data. Many applications in e-commerce, financial services, logistics, and other areas now can benefit from the power of real-time analytics by leveraging the widespread expertise focused on Hadoop MapReduce.

About the Authors

David L. Brinker is the Chief Operating Officer of ScaleOut Software, Inc. Dave has over 25 years of software industry experience in a variety of operational, financial, and executive roles in both public and private companies. Prior to joining ScaleOut Software, Dave was CEO/Chairman at Webridge, Inc., a pioneer in secure extranets. As an early employee of Mentor Graphics, he launched Asian operations and grew Mentor's Pacific business while stationed in the Japan headquarters. Near the end of his twelve years at Mentor he managed the worldwide field organization. In addition to raising capital and selling two private businesses, Dave holds an Oregon CPA Certificate and spent his early years at KPMG and Price Waterhouse Coopers.

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.