With the ability to analyze virtually unlimited amounts of unstructured and semi-structured data, Apache Hadoop has the potential to transform data analytics. Yet with Apache Hadoop only recently becoming a more mainstream technology, benchmarking and deployment tools are still catching up, which can make realizing the full potential of a performance-optimized Apache Hadoop cluster challenging.

As with most new technologies, Apache Hadoop has gone from being an interesting concept to an established technology. And as new technologies become more established, they become easier to optimize. For example, rail companies, whose technologies and processes are well known, realize the value of optimization. From the performance of the locomotive engines, to the routes that mile-long trains take, optimization is a key factor in lowering the cost per pound of cargo and getting that cargo to its destination efficiently. The same principle applies to choosing the right computing platform that is optimized to squeeze the last bit of performance out of Apache Hadoop. A non-optimized Apache Hadoop cluster might still get the job done, but data analytics tasks will take longer and be less efficient.

Systems administrators can increase Apache Hadoop cluster performance by using existing Intel technologies, such as Intel® Solid State Drives (SSDs) and Intel® Ethernet 10 Gigabit Server Adapters. These technologies, combined with Intel® Xeon® processors, can improve a one terabyte (TB) MapReduce sort from four hours to seven minutes. Yet because Apache Hadoop is built on Java, one of the most effective ways to increase performance is to optimize Java itself to take advantage of Intel architecture enhancements.

**Intel and Oracle: Building a Better Java Foundation**

Through a collaborative effort between Intel and Oracle software engineers, organizations can realize significant performance gains using the latest Oracle Java Virtual Machine (JVM) running on servers powered by Intel Xeon processors. Intel and Oracle have worked together for the past six years to improve the overall performance of Java running on the Intel Xeon processor family. Intel software engineers work directly with Oracle Java engineers to identify and provide specific optimizations that take advantage of the latest Intel microarchitecture enhancements.

Each new generation of Intel microarchitecture introduces enhancements that increase software performance. Older versions of Java will inherently run faster on newer platforms due to increased clock speeds and more efficient instruction execution. But to push performance even further, software engineers can also optimize Java to take advantage of new microarchitecture enhancements. When Intel releases a new microarchitecture and platform, Intel and Oracle software engineers work together to identify specific areas within the JVM that can be tuned to take advantage of the new hardware advances. These optimizations can deliver the highest performance possible across a broad range of applications, including Apache Hadoop.
Apache Hadoop Architecture and Java Components

Built on Java, Apache Hadoop is a scalable, distributed data storage and analytics engine that can store and analyze massive amounts of unstructured and semi-structured data. The distributed architecture of Apache Hadoop is comprised of multiple nodes, which are individual servers that run an off-the-shelf operating system and the Apache Hadoop software. This design lets Apache Hadoop manipulate large datasets across clusters of commodity hardware using a simple programming model.

Three core components written in Java form the foundation of Apache Hadoop:

- **Apache Hadoop Distributed File System** (HDFS): HDFS provides a high-performance file system that sits on top of an Apache Hadoop node’s native file system and can span and replicate data across the nodes in the cluster. The file system can scale from a single node to thousands of nodes and provides resiliency and performance for large datasets.

- **MapReduce**: A processing framework that enables large-scale analytics across distributed, unstructured data. Apache Hadoop breaks down analytics jobs into smaller tasks that are then executed by MapReduce on individual nodes as close as possible to the data being analyzed.

- **NameNode**: a directory component of HDFS that keeps track of where data is located within each of the cluster’s slave nodes.

- **JobTracker**, which breaks the request down into multiple smaller tasks based on which slave nodes contain the data, and then assigns these tasks to those specific slave nodes.

These two services then communicate with additional Java services on the slave nodes. Each slave node utilizes three services that communicate with the master node and perform the actual analytics:

- **DataNode**, a component of HDFS that stores and retrieves data on the slave node at the request of NameNode.

- **TaskTracker**, which receives MapReduce tasks from JobTracker and runs the tasks within a separate JVM.

- **MapReduce**, the analytic service that performs the analytics on the data.

---

![Figure 1: Overview of Apache Hadoop architecture and components](image)
Once MapReduce completes the analytics tasks on the slave node, it returns the results to the master node. The master node waits for all of the tasks to complete, compiles the results, and then returns the results to the client.

The Apache Hadoop framework spawns a new JVM for each MapReduce function on each slave node, which can result in thousands of individual JVMs being created for a single analytics task. Since Apache Hadoop does not share memory resources across the various nodes, it is important that the JVM and the services written in Java perform as optimally as possible on each node because reduced performance on any single node can hamper data analytics performance across the cluster.

Apache Hadoop Performance Gains

Since 2007, Oracle and Intel have improved Java performance up to 14 times by tying specific Java optimizations to advancements in the underlying hardware.

During the course of their partnership, Oracle’s and Intel’s software engineers have optimized key Java elements that help increase Apache Hadoop performance:

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Description</th>
<th>Apache Hadoop* Performance Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast cyclic redundancy check (CRC) computation</td>
<td>Soon to be released optimization that improves CRC performance by using the carry-less multiplication instruction and CRC32 instruction</td>
<td>Increases file checksum and compression/decompression checksum, which increases Apache Hadoop network and file system performance</td>
</tr>
<tr>
<td>Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI)³</td>
<td>Accelerates AES data encryption</td>
<td>Increases data encryption performance for data stored in HDFS* and for network traffic between nodes</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives (Intel® IPP) compression</td>
<td>Optimizes performance of common compression algorithms, including zlib and LZO</td>
<td>Increases compression performance, which reduces network and disk input/output (I/O) across the Apache Hadoop cluster</td>
</tr>
<tr>
<td>Intel® Advanced Vector Extensions/Advanced Vector Extensions 2 (Intel® AVX/AVX2) superword optimization</td>
<td>New 256-bit instruction set extension to Intel® Streaming SIMD Extensions (Intel® SSE) that dramatically increases floating point calculation performance</td>
<td>Improves the performance of MapReduce operations that contain array and string manipulation, such as sub-string or character searches; also improves integer and floating point calculations</td>
</tr>
<tr>
<td>ISO 8859-1 encoding</td>
<td>Uses the AVX/AVX2 single instruction, multiple data (SIMD) vector instructions to increase character encoding to 16 or 32 characters at a time instead of character by character</td>
<td>Improves the performance of applications that use international string encoding</td>
</tr>
<tr>
<td>Intel® Transactional Synchronization Extensions (Intel® TSX)</td>
<td>Soon to be released optimization that uses Intel transactional memory instructions for Java lock elision</td>
<td>Increases the performance of applications that use synchronization and concurrent data structures</td>
</tr>
<tr>
<td>Allocation prefetch</td>
<td>Caches additional memory ahead of the object being allocated</td>
<td>Increases critical code path performance</td>
</tr>
<tr>
<td>Large-page usage</td>
<td>Uses large memory pages for both code and data memory</td>
<td>Increases performance of large analytics jobs</td>
</tr>
</tbody>
</table>

![Figure 2: Performance gains from combined Intel hardware and Java Development Kit* (JDK*) optimizations since 2007](image-url)
### Optimizing Java* and Apache Hadoop* for Intel® Architecture

<table>
<thead>
<tr>
<th>Optimization Feature</th>
<th>Description</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorization</td>
<td>Loads, operates on, and stores multiple array elements within a single machine instruction</td>
<td>Increases execution performance of widely used operations, which improves overall Apache Hadoop service performance</td>
</tr>
<tr>
<td>Compressed references</td>
<td>Stores 32-bit compressed references for 64-bit pointers</td>
<td>Reduces the memory footprint and cache misses of Apache Hadoop while improving cycles per instruction (CPI)</td>
</tr>
<tr>
<td>Lock optimizations</td>
<td>Multi-tiered lock deferrals to avoid or delay inflation to fat locks for coincidental lock contention</td>
<td>Increases Apache Hadoop service scalability through reduction of fat locks</td>
</tr>
<tr>
<td>HashMap, TreeMap, BigDecimal</td>
<td>Optimizes commonly used functions</td>
<td>Improves performance by using caching and other optimizations to reduce path length and object allocation</td>
</tr>
<tr>
<td>String optimizations</td>
<td>Optimizes code generation for methods that are frequently used</td>
<td>Reduces intermediate object allocation for common operations such as string comparisons</td>
</tr>
<tr>
<td>Instruction decoding optimization</td>
<td>Optimizes instruction decoding by removing length changing prefixes and aligning branch targets.</td>
<td>Allows the JVM and Apache Hadoop to take advantage of new Intel microarchitecture performance features</td>
</tr>
<tr>
<td>Out-of-order execution optimization</td>
<td>Eliminates partial flag and partial register stalls</td>
<td>Allows the JVM and Apache Hadoop to take advantage of new Intel microarchitecture performance features</td>
</tr>
<tr>
<td>Intrinsic function optimization</td>
<td>Optimizes the most common code paths</td>
<td>Reduces instruction count and CPI, which increases overall Apache Hadoop service performance</td>
</tr>
</tbody>
</table>

The combined effect of these optimizations can increase the analytics and internode performance of Apache Hadoop on Intel hardware. In a performance benchmark conducted by Intel, a Hadoop cluster running on servers powered by Intel Xeon family processors and running the latest JDK experienced performance gains of 50 percent in the TeraSort* industry-standard benchmark, and 70 percent on a Hadoop Sort benchmark.\(^4\)

**Compression: Where the Wheels Meet the Track**

While many of the JVM and Intel microarchitecture optimizations can increase the performance of individual Apache Hadoop services, data compression can help Apache Hadoop realize significant performance gains across the entire cluster. Data that is compressed before being sent across the network can help reduce network traffic and disk input/output (I/O), which can improve overall performance between Apache Hadoop nodes.

Intel and Oracle have optimized Java to take advantage of Intel® Integrated Performance Primitives (Intel® IPP), a library of optimized software functions that improve the performance of common compression algorithms, in addition to multimedia processing, data processing, and communications applications. With Intel IPP enabled in Java, zlib and LZO compression performance increased TeraSort benchmark results.\(^5\)

Intel is also working to optimize compression CPU utilization with Intel® QuickAssist Acceleration Technology (Intel® QAT). This technology embeds compression algorithms in the hardware that accelerate common compression tasks while lowering CPU utilization, which frees the CPU to focus on other Apache Hadoop functions such as MapReduce tasks.

By utilizing Intel QAT, the performance of Apache Hadoop sort operations can increase by up to 60 percent while CPU utilization can be reduced by 20 percent.\(^6\)

![TeraSort* and Hadoop Sort* Performance Gains](image-url)

*Figure 3: TeraSort* and Hadoop Sort* performance gains running the latest optimized JDK on Intel hardware*
Conclusion

Apache Hadoop is transforming the way organizations analyze and store data, enabling new use cases for data analytics. Massive amounts of semi-structured and unstructured data can now be easily manipulated, and the scalability of Apache Hadoop means that organizations can expand from one to thousands of nodes as their needs grow.

The Intel architecture provides a solid, high-performance foundation for Apache Hadoop clusters. Intel’s and Oracle’s work on improving performance with Java has yielded demonstrable improvements to Apache Hadoop performance. By combining servers based on the latest Intel Xeon processor family with Oracle’s Java Virtual Machine, organizations can realize significant performance benefits across their Apache Hadoop clusters.

Intel® Integrated Performance Primitives (Intel® IPP) compression increases TeraSort* benchmark performance

![Benchmark Performance Chart](image)

For more information on how your Apache Hadoop cluster can benefit from Intel architecture, visit www.intel.com/bigdata and hadoop.intel.com.

1 Performance baseline consisted of a server configured with an Intel® Xeon® 5690 processor, a 7200 rpm SATA hard drive, and a single gigabit Ethernet adapter. The enhanced configuration that resulted in a one terabyte sort performance improvement from four hours to seven minutes consisted of a server configured with an Intel Xeon E5-2690 processor, Intel® SSD 520 series solid state drive, an Intel® Ethernet 10 Gigabit Server Adapter, and the Intel® Distribution for Apache Hadoop®.


3 Intel® Advanced Encryption Standard–New Instructions (Intel® AES-NI) requires a computer system with an AES-NI-enabled processor, as well as non-Intel software to execute the instructions in the correct sequence. AES-NI is available on select Intel® Core™ processors. For availability, consult your system manufacturer. For more information, see http://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html.

4 The Apache Hadoop* cluster consisted of a master server and four slave servers over a 10 gigabit Ethernet network. The master server was configured with dual Intel® Xeon® X5570 processors, and 64 GB of RAM. The slave nodes each consisted of dual Intel® Xeon® E5-2680 processors; 128 GB RAM, two dual-core Intel® QuickAssist Technology compression cards with hardware version C1 SKU A, firmware version 1.0.0, and driver version 1.2.0; seven 300 GB solid state drives in a RAID 0 configuration; and an external SAS enclosure with 24 64 GB hard drives in a RAID 0 configuration. Each server was running CentOS Release 6.3 with Linux kernel 2.6.32-150.1.3.el6.x86_64, 64, and Hadoop 1.0.4. The TeraSort and Sort benchmarks were run against a 500 GB dataset. The 50 percent TeraSort and 30 percent Sort performance increases were seen between running the benchmarks on JDK 1.6.0_14 and JDK 1.7.0_13.

5 The Apache Hadoop cluster* consisted of a master server and four slave servers connected over a gigabit Ethernet network. The master server was configured with dual Intel® Xeon® X5570 processors, 64 GB RAM, a single 64 GB solid state drive, and a single 500 GB hard drive. Each slave server was configured with dual Intel® Xeon® X5570 processors. Each server was running Ubuntu 11.10 64-bit with Linux kernel 3.0.0-12.20, Hadoop 0.20.203.0, and the Oracle Java HotSpot® 64-bit server VM version 1.7.

6 The Apache Hadoop cluster* consisted of a master server and four slave servers connected over a gigabit Ethernet network. The slave servers where the benchmark was run were each configured with dual Intel® Xeon® E5-2680 processors, 64 GB RAM, two dual-core Intel® QuickAssist Technology compression cards, and three 300 GB solid state drives. Each server was running CentOS Release 6.3 64-bit and the Oracle Java HotSpot™ 64-bit server VM version 1.7.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, go to: http://www.intel.com/PerformanceResources/benchmark_limitations.htm.