APACHE HADOOP PERFORMANCE-TUNING METHODOLOGIES AND BEST PRACTICES

Shrinivas Joshi, Advanced Micro Devices, Inc. {shrinivas.joshi@amd.com}

Introduction

Users who have deployed and tried tuning their Hadoop clusters for the first time will certainly attest to the fact that optimizing Hadoop clusters is a daunting task.

Apart from the nature and implementation of Hadoop jobs, hardware, network infrastructure, OS, JVM, and Hadoop configuration properties all have a significant impact on performance and scalability of Hadoop workloads.

Using TeraSort as a reference workload, this poster attempts to educate the audience on challenges involved in tuning performance of Hadoop setup, different tools available for monitoring and diagnosing performance bottlenecks, tuning best practices, empirical data on effect of various tunings on performance, and some future directions.

Challenges

• Hadoop is a large and complicated framework involving a number of entities interacting with each other across multiple hardware systems.
• Performance of Hadoop jobs is sensitive to every component of the cluster stack: Hadoop configuration, JVM, OS, network infrastructure, underlying hardware, and possibly BIOS settings.
• Hadoop supports a large number of configuration properties and a good chunk of these can potentially impact performance.
• As with any large software system, diagnosing performance issues is a complicated task.

Tools

• Ganglia and Nagios – effective in monitoring cluster-wide statistics.
• Hadoop Vaidya – offers guidelines on mitigating framework-level bottlenecks.
• Hadoop Core – offers simple and insightful information on performance-affecting events.
• AMD CodeAnalyst™, Oracle® Solaris Studio and Intel® VTune™ – profilers for in-depth source-level analysis.
• AMD MultEvent, Linux perf, and OProfile – platform-level profilers for in-depth analysis of hardware-level performance events.

Cluster A

• 6 DNs, 1 NN: 2 chips/6 cores per chip: AMD Opteron™ 2435 @2.6GHz
• 16GB DDR2 800 RAM per node
• 6 x 1TB Samsung Spinpoint F3 @7200
• Ubuntu 11.04 Server x64, Oracle JDK6 update 25 x64
• Dataset size – 64 GB

Cluster B

• 5 DNs: 2 chips/4 cores per chip: AMD Opteron 2356™ @2.3GHz
• 1 NN: 4 chips/4 cores per chip: AMD Opteron 8356™ @ 2.6GHz
• 64GB DDR2 667 and DDR2 800 RAM
• 6 x 1TB Samsung Spinpoint F3 @7200
• Ubuntu 11.04 Server x64, Oracle JDK6 update 25 x64
• Dataset size – 1 TB

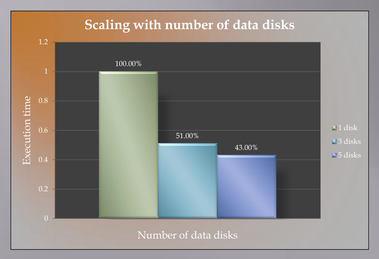
Hadoop Configuration Tuning

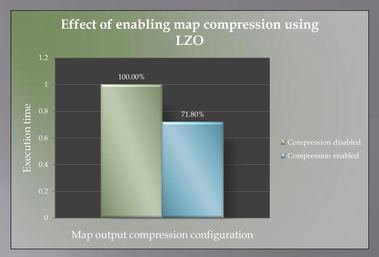
• Using maximum possible map and reduce slots, identify the optimal number of disks that maximizes I/O bandwidth.
• Experiment with different HDFS block sizes.
• Identify heap usage and GC characteristics of framework processes and lock in their heap and GC settings.

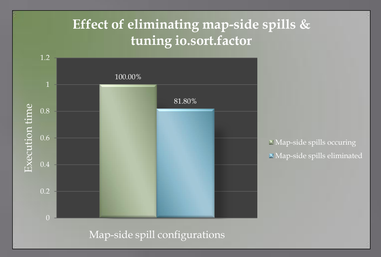
• Start with biggest-payoff properties.
• Enable map output compression.
• Experiment with different codec choices. In our experience, LZO codec performed better across multiple workloads.
• Experiment with JVM reuse policy.
• This helps reduce disk & network I/O wait and increase CPU utilization.

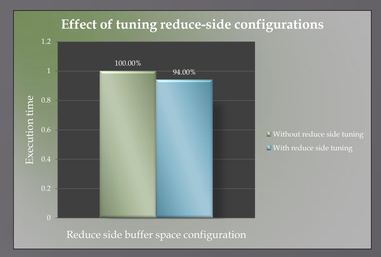
• If memory is not a bottleneck, try to eliminate map-side spills by tuning io.sort.mb. At the least, reduce the number of spills.
• In case there are no spills, tune io.sort.spill.percent.
•This also helps reduce disk I/O wait time and increase CPU utilization.

• Try to avoid or eliminate intermediate disk I/O operations on reduce side by tuning heap sizes of reduce JVMs.
• If the reduce functionality is not heap heavy, adjust map output buffer size.
• Tune framework-related resources such as task tracker threads, data node, and name node handler count.









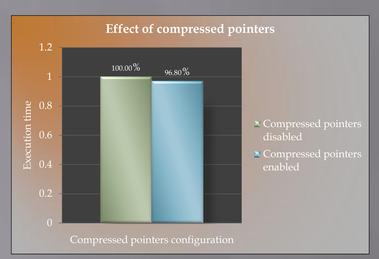
JVM Configuration Tuning

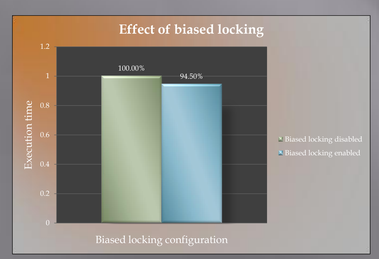
• On 64-bit Oracle JDK 6 update 25, compressed pointers are ON by default. If you are using an older version of JDK and compressed pointers are disabled, experiment with enabling them.
• Compressed pointers reduce memory footprint.

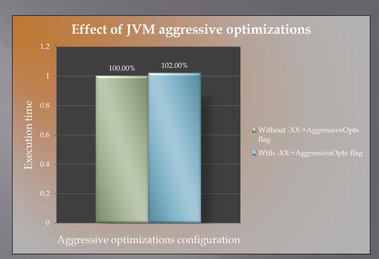
• Biased-locking feature of Oracle HotSpot JDK improves performance in scenarios with uncontended locks.
• Given the architecture of Hadoop framework, biased locking should generally improve performance.

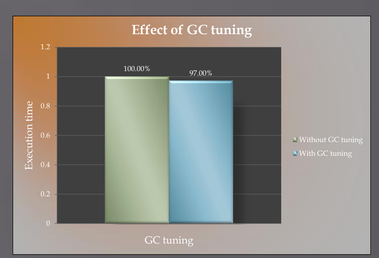
• Experiment with AggressiveOpts flag.
• Experiment with UseCompressedStrings flag.
• Experiment with UseStringCache flag.
• Experiment with UseNUMA flag.
•Experiment with UseLargePages flag.

• Verify whether the JVM is running out of code cache. Increase code cache size if necessary.
• Perform detailed GC log analysis and tuning.









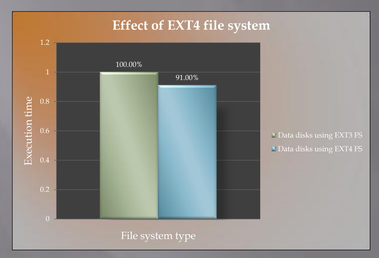
OS Configuration Tuning

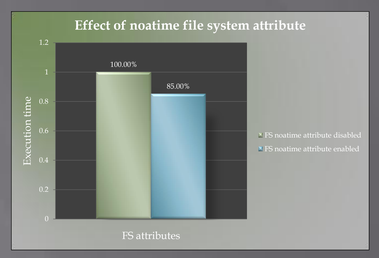
• Certain Linux distributions support EXT4 as the default file-system type.
• If you are using another type of file system, experiment with EXT4 file system.

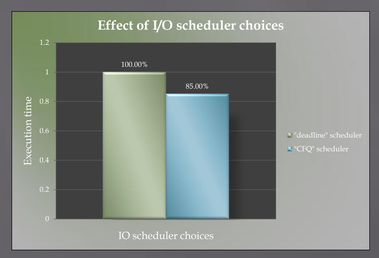
• By default, every file read operation triggers a disk write operation for maintaining last access time.
• Disable this logging using noatime, nodirtaime FS attributes.
• Experiment with other FS attribute tuning such as extent, flex_bg, barrier etc.

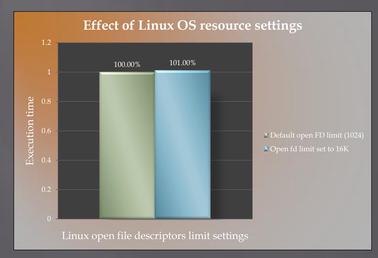
• Linux kernels support 4 different types of I/O schedulers – CFQ, deadline, no-op, and anticipatory.
• Experiment with different choices of I/O scheduler, especially CFQ and deadline.

• Linux OS limits such as max open file descriptors and epoll limits can affect performance.
• Experiment with these limits.









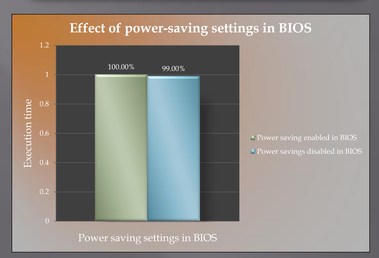
BIOS Configuration Tuning

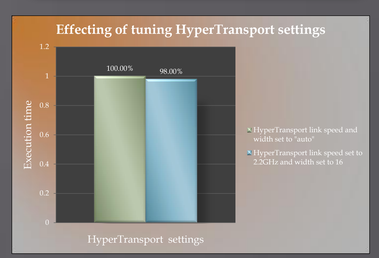
• Native command queuing feature of modern hard drives helps improve I/O performance by optimizing drive head movement.
• Experiment with AHCI option in BIOS, which can be used to enable NCQ mode.

• When all the CPU cores on the hardware are not fully utilized, the processor could be downgrading CPU frequency.
• Experiment with ACPI and other power-related BIOS options.

• On some AMD processor-based systems, HyperTransport™ link speed and width are dynamically adjusted to reduce power consumption.
• If memory bandwidth is a bottleneck, experiment with this option.

• Modern AMD processors support a feature called HT assist (a.k.a. probe filters). This feature reduces traffic on memory interconnects at the expense of some portion of L3 cache.
• Experiment with this feature. Try disabling it if your job is cache-sensitive.





Best Practices

Hadoop:

• Identify the right number of data disks your job requires.
• Observe Hadoop framework heap usage and GC patterns and lock in heap and GC JVM flags for these processes.
• Using default settings, start with Hadoop configuration tuning.
• Focus on properties shown to have a big impact, such as map/reduce output compression and JVM reuse policy.
• Avoid or eliminate map-side spills.
• Avoid or eliminate reduce-side disk I/O by tuning reduce JVM heap size, map output copier threads, merge factor, and tasktracker threads handling reduce-side requests.
• If reduce functionality is not heap-heavy, try to maximize the map output storage buffers.
• Try to maximize CPU usage by tuning number of map and reduce JVMs.
• Tune other framework-related components such as datanode and namenode handler count.

JVM:

• Experiment with performance-affecting JVM flags such as biased locking, compressed pointers, AggressiveOpts, etc.
• Perform a thorough Java GC analysis and tune GC related flags.

Operating System:

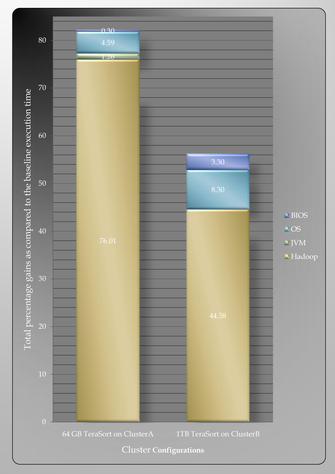
• Experiment with various OS-level tunings such as choice of FS, choice of I/O scheduler, and limits on OS resources.

BIOS:

• Finally, verify if any default BIOS settings are negatively affecting performance of your Hadoop jobs.

Tuning Highlights

• On ClusterA, we achieved a speed-up factor of 5.6X.
• On ClusterB, we achieved a speed-up factor of 2.2X.



Conclusion & Future Direction

• Configuration tuning of all the components of Hadoop stack is an important exercise and can offer a huge performance payoff.
• Different Hadoop workloads will have different characteristics, so it is important to experiment with different tuning options.
• We want to perform a similar study in multi-tenant environments and in the cloud environment.
• We want to explore JVM and JDK optimizations targeting peculiar characteristics of Hadoop framework and jobs running on top of it.