Architecture Support for Big Data Analytics

Ahsan Javed Awan
EMJD-DC (KTH-UPC)
(http://uk.linkedin.com/in/ahsanjavedawan/)
Supervisors: Mats Brorsson(KTH), Eduard Ayguade(UPC), Vladimir Vlassov(KTH)
Why should we care?

Data Growing Faster than Technology

- Technology improvement (Moore's Law)
- Largest publicly-reported data warehouse size

Growing technology gap

WinterCorp Survey, www.wintercorp.com

*Source: Babak Falsafi slides
• A mismatch between the characteristics of emerging workloads and the underlying hardware.
  - Z. Jia et-al, “Characterizing and subsetting big data workloads,” in IISWC 2014
Performance Characterization of In-Memory Data Analytics on a Modern Cloud Server

x86 Core Explosion

- 1 core
- 4 core
- 8 core
- 64 core

1971 2000 2010 2013

Data Volume Growth

- 130 Exa bytes
- 1.2 Zetta bytes
- 0.8 Yotta bytes

2005 2010 2015

*Source: SGI*
Our Focus

Improve the single node performance in scale-out configuration

*Source: http://navcode.info/2012/12/24/cloud-scaling-schemes/*
Which Scale-out Framework?

DStream | Shark | MLBase | GraphX

Spark

Mesos, YARN

HBase | HDFS

[Picture Courtesy: Amir H. Payberah]
Our Approach

- A three fold analysis method at Application, Thread and Micro-architectural level
  - Tuning of Spark internal Parameters
  - Tuning of JVM Parameters (Heap size etc..)
  - Concurrency Analysis
  - General Architectural Exploration
Benchmarks

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Transformations</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro-benchmarks</td>
<td>Word count</td>
<td>map</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reduceByKey</td>
</tr>
<tr>
<td></td>
<td>Grep</td>
<td>filter</td>
</tr>
<tr>
<td></td>
<td>Sort</td>
<td>map</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sortByKey</td>
</tr>
<tr>
<td>Classification</td>
<td>Naive Bayes</td>
<td>map</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td>K-Means</td>
<td>map</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mapPartitions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reduceByKey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>filter</td>
</tr>
</tbody>
</table>

3GB of Wikipedia raw datasets, Amazon Movies Reviews and numerical records have been used
Hyper Threading and Turbo-boost are disabled
## System Configuration

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Wc</th>
<th>Gp</th>
<th>So</th>
<th>Km</th>
<th>Nb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap Size (GB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>Young Generation Space (GB)</td>
<td>45</td>
<td>25</td>
<td>45</td>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>MaxPermSize (MB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>Old Generation Garbage Collector</td>
<td>ConcMarkSweepGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Young Generation Garbage Collector</td>
<td>ParNewGC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.storage.memoryFraction</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>spark.shuffle.consolidateFiles</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.shuffle.compress</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.shuffle.spill</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.shuffle.spill.compress</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.rdd.compress</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spark.broadcast.compress</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spark scales poorly in Scale-up configuration
Multicore Scalability of Spark Stage Level Performance

- Shuffle Map Stages don't scale beyond 12 threads across different workloads
- No of concurrent files open in Map-side shuffling is C*R where C is no of threads in executor pool and R is no of reduce tasks
Task Level Performance

<table>
<thead>
<tr>
<th>Stage</th>
<th>12-threads</th>
<th>24-threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wc_1</td>
<td>17.03</td>
<td>61.50</td>
</tr>
<tr>
<td>So_3</td>
<td>24.58</td>
<td>68.50</td>
</tr>
<tr>
<td>Km_0</td>
<td>38.02</td>
<td>83.20</td>
</tr>
</tbody>
</table>

Percentage increase in Area Under the Curve compared to 1-thread
Is there thread level load imbalance??

![Graph showing thread load imbalance](image)
CPU Utilization is not scaling with performance
Is there any Work Time Inflation??
How does Micro-architecture contribute to Work time inflation??
Cont...
Cont...
Is Memory Bandwidth a bottleneck??
Key Findings

- More than 12 threads in an executor pool does not yield significant performance.
- Spark runtime system need to be improved to provide better load balancing and avoid work-time inflation.
- Work time inflation and load imbalance on the threads are the scalability bottlenecks.
- Removing the bottlenecks in the front-end of the processor would not remove more than 20% of stalls.
- Effort should be focused on removing the memory bound stalls since they account for up to 72% of stalls in the pipeline slots.
- Memory bandwidth of current processors is sufficient for in-memory data analytics.
How Data Volume Affects Spark Based Data Analytics on a Scale-up Server

The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020

Do Spark based data analytics benefit from using larger scale-up servers

![Graph showing the speed-up of various Spark operations with increasing number of cores](image-url)
Is GC detrimental to scalability of Spark applications?
How does performance scale with data volume?
Does GC time scale linearly with Data Volume??
How does CPU utilization scale with data volume?
Is File I/O detrimental to performance?
Motivation

How does data size affects micro-architectural performance?
How Data Volume Affects Spark Based Data Analytics on a Scale-up Server
Cont..
Cont..
Key Findings

- Spark workloads do not benefit significantly from executors with more than 12 cores.
- The performance of Spark workloads degrades with large volumes of data due to substantial increase in garbage collection and file I/O time.
- Without any tuning, Parallel Scavenge garbage collection scheme outperforms Concurrent Mark Sweep and G1 garbage collectors for Spark workloads.
- Spark workloads exhibit improved instruction retirement due to lower L1 cache misses and better utilization of functional units inside cores at large volumes of data.
- Memory bandwidth utilization of Spark benchmarks decreases with large volumes of data and is 3x lower than the available off-chip bandwidth on our test machine.
What are the major bottlenecks??


Future Directions

- NUMA Aware Task Scheduling
- Cache Aware Transformations
- Exploiting Processing In Memory Architectures
- HW/SW Data Prefetching
- Rethinking Memory Architectures