COMPSCI 590S Systems for Data Science **Fall 2016**

Lecture 15: Making Sense of Performance in Data Analytics Frameworks

Lecturer: Emery Berger *Scribe(s): Matt McNally, Yasaman Ahrabi*

1.1 Response to Paper

Previously, people thought that the main problems with data analytics frameworks were I/O and stragglers, but that wasnt the case in this paper. The paper asserts that those bottlenecks were not the real problem.

Spark was designed to avoid I/O, so when the authors of the paper tested how much faster I/O would help, it naturally wasnt impactful, since that I/O had already been avoided.

1.2 Threats to validity of paper

Are the papers results generalizable? In order for the result of the paper to generalizable, there needs to be a large, diverse sample. The first piece of that is to have a large sample size. This paper only had one, Spark, which isnt enough. The second piece is to avoid sample bias, which requires a representative sample. The paper should have used several popular data analytics frameworks, but it instead used only one. The third piece is to have workloads that are representative of real usage. The paper had one real workload, the trace, which was not sufficient. The paper should have also characterized the workloads that follow the hypothesis.

1.3 Question about Warmup

1.3.1 Bytecode

When Java is compiled, it is really just converted into java bytecode. Since Spark uses Scala and Scala uses the JVM, it has a similar process. Java bytecode is 20X - 100X slower than machine code. When Java runs bytecode, it runs one instruction at a time by going through a massive switch statement.

Python is a completely interpreted language. Python can compile to something like Python bytecode, which produces .pyc files, which is still interpreted. This is why Python is so slow.

1.3.2 JIT Compiler

Just in Time compiler compiled hot methods, which are frequently used. There is a warmup period where methods need to get hot so that they get compiled.

1.4 Side note about references in Java

A Strong reference is the normal reference in Java.

A Soft reference is reference to an object that the garbage collector can collect at any time. This could be used as a cache, but it can be completely wiped at any time, so really isnt a good option. Can be fixed with prioritized soft references that are introduced in a new paper. The prioritized soft references can be used to implement LRU caching.

Weak references are soft references that survive the first garbage collection cycle.

Phantom references are references that dont count as a reference when garbage collection counts references pointing to objects. It is used when dealing with finalization, which is essentially Javas version of C++ destructors. Objects with phantom references arent deleted until the very end of a round a garbage collection, which means that finalizers that run during garbage collection can use phantom references to point to objects that are also be garbage collected, and it will work.