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

Lecture 16: Scalability and COST Metrics

Lecturer: Emery Berger *Scribe(s): Anna Deng,*

16.1 What is COST?

COST is a measurement of how much performance overhead that should be paid compare to a single threaded implementation on one node. It turns out that a single threaded implementation could run the workloads discussed in paper in a hundredth of the time. That means, 100 cores can match up with a single node.

16.2 Good or Bad Metric?

16.2.1 Bad Metric

COST Metrics are bad because most of the times the systems are designed to have other features. For example, COST Metrics does not take the following into account:

- Fault tolerance
- Size
- Availability
- Network bandwidth

Gustafson's Law also play a part here. A reminder of the law is that scalability can be unlimited, and it is proportional to the problem sizes. Problem size is much greater than the number of cores.

$$\Delta \textit{ Problem size} >> \Delta \textit{ Number of cores}$$

A question of does the system scale to really large workloads? And how big can the size of the workloads be? The size of the workload can be determined by RAM, Disk, CPU and cache. Network does no not fit here, but it has a role in terms of serving people.

For Pregel, disk does not matter except during graph loading. RAM and CPU power does matter a lot.

16.2.2 Cache Consistency

Cache consistency for large workloads are never going to be a problem. Processor chips are set up to have many cores, and each core has its own on-core cache. The cores will share some level of cache and they have to make sure that their copies of cache lines won't go out of sync. This problem is known as Cache Coherence.

In general, if the workloads are embarrassingly parallel, than the number of threads will be proportional to the speedup.

16.2.3 Overhead and Speedup

The observation about overhead and speed up is what's important. It turns out that things get scaled really well if the overhead is really high.

Suppose we have an operation that adds two numbers:

$$ADD\ X\ Y,\ Z$$

It is difficult to optimize the single instruction. However, if we have a lot of overheads in addition to the single ADD operation. Like in the following:

$$For\ i\ in\ range(1, 100...00)$$

$$x = x$$

This can speed up a lot with multiple threads. Each thread can operate on a different range such that the work are spread out evenly across these threads. As more threads are added, speedup is scaled up much towards linear. But the overhead is not the core of the problem, because it is garbage and doing nothing valuable.

16.3 Hilbert Curve

In the case of a one-dimensional array, things that are likely to be accessed in the future are located next to each other in order to maximize locality. However, it is not as easy when moving to two dimensions such as a Cartesian Graph in the notion of Matrices.

Hilbert Curve is one of the optimization discussed in the paper. It is also one of the Space Filling Curves with natural locality. The basic idea of a Space Filling Curve is that if the things are divided up into quadrant and When one of the quadrant is visited (fits into RAM/cache), then it will never get out of RAM/Cache until it's done.