Scaling data processing from CPU to distributed GPUs – William Malpica Speaker Q&A

Question: Can you go back to the slide on this performance on CPUs flatlining? And why is this, why was the performance flatlining?
Answer: From a Spark perspective, the more workers you had, there's only so much that you can distribute. If you add more workers, then each worker is doing a little bit smaller part of the problem, but they still all have to intercommunicate to kind of get to the final answer. Especially in SQL and distributed computing, there are algorithms that have effectively what we call a pipeline breaker, where after all the workers have computed a certain amount of thing, all the workers have to intercommunicate and merge their results before they can continue to the next stage of processing. So that's kind of one of the reasons why it gets to the point that more workers isn't really going to help.

Question: How does the user know if their workload is bound by latency, I/O, or compute?
Answer: You have to profile your software to figure out your performance characteristics and bottlenecks. The tools you use will depend on the library, language, etc. For GPU programs, you can use tools like Nsight or profilers to see how much time is spent moving data from host to GPU, doing disk I/O, CPU compute vs. GPU compute, memory consumption, etc. For large serving systems, you can hook up telemetry and monitor different parts of execution using tools like Datadog.

Question: When you say about like the column format is like better for GPUs. Can you explain why and what is special about the color format that make it so easy as you work on GPU?
Answer: Whenever you're moving data between host and GPU, you want to move data that is in contiguous buffers, which will be faster. In many algorithms, you want to have data in nice vectors so that you can easily index across the data and parallelize how the different cores of the GPU are processing different elements of the data and vectors. A column of data or columnar format effectively translates to a vector in one contiguous buffer of memory that you can easily index and parallelize across all the different cores.

Question: What are some of the typical use cases of Theseus?
Answer: Theseus particularly excels at very large data sets, hundreds of terabytes or more. It's not necessarily a specific industry, but more about the scale of data. Companies that produce a lot of data and need to analyze it quickly can benefit, like large retailers, banks/fintech, data security, networking/telecoms. For example, telecoms can produce hundreds of terabytes of network data every day, and powerful, fast analytics enables threat detection, anomaly detection, system failure detection, with fast turnaround times.

Question: While parquet files more friendly than CSV for example, i still don't think it's as straight forward on the GPU. Could you speak to benefits? From what i've seen(albeit limited) work effiency is also lower on the GPU side than it is CPU
Answer: Work efficiency can be lower, but you have so much more work that you can do that it makes up for it. CSV files can still be read and parsed very fast on GPUs, because you can still parallelize the parsing. You can send different blocks of bytes to different threads, where each will do its own parsing, and then merge the results into one coherent output

Question: I've contributed to libcudf before, and one thing I'm curious to see is benchmark comparisons of CPU parallelized to GPU parallelized. Pandas is not parallelized for example.
Answer: I agree, that would be a good comparison. There are some frameworks out there which help with python parallelization

Question: Pinning used to be slower... only reason I ask.
Answer: When you move data from the GPU to host (or anywhere), usually it will need to get copied to a pinned bounce buffer. Because pinning memory is expensive, it does make things slower
There are some techniques for doing that faster, such as having and managing a pool of pre-allocated and pre-pinned bounce buffers, which you can re-use. But that also does add a fair amount of complexity.

Question: Can you go back to the part where you showed the graph of the CPU performance flatlining? Had the caption(paraphrasing) you can't throw more CPUs at the problem
Answer: Ahh, ok, so a brief explanation: Two main reasons:

1. In a typical SQL execution flow you will have some phases where you need to merge results from all workers before you can continue. Which means that at some point all workers can only go as fast as the slowest worker (there are some ways that engines can mitigate this, but its still affects performance). This means that sometimes having more workers means that you are more likely to have a slower slowest worker.
2. Workers need to often coordinate things between themselves, and having more workers can sometimes make this overhead worse.

These sort of issues can counteract the benefit you have from spreading the workload across more workers