ANACONDA
Powered by Continuum Analytics®

WHITEPAPER

HIGH PERFORMANCE PYTHON FOR OPEN DATA SCIENCE

Michele Chambers, VP Products &
Christine Doig, Data Scientist & Product Marketing Manager
Kristopher Overholt, Ph.D. & Product Manager
Stanley Seibert, Ph.D. & High Performance Python Team Leader

May 2016

In This Whitepaper

The rage is Big Data. But, the reality is that enterprises have a vast array of data sizes, some small and some very large, with most someplace in between. Hadoop disrupted the marketplace by making the price of data storage dramatically drop by many orders of magnitude. This economic change was made possible by connecting low cost servers into a cluster that made the cluster appear to be one massive machine with not only virtually unlimited storage, but also large amounts of processing power.

While accessing data efficiently in a network had been addressed for decades in databases and data warehouses, parallel compute processing remained a specialized field for scientific and high performance computing. The MapReduce paradigm was an early attempt to create a simplified framework to make it easier to divide and conquer workloads in order to take advantage of the inexpensive distributed processing power in a cluster. For all its progress, MapReduce remained relatively slow, as it was designed as a batch processing system, just as early mainframes relied mainly on JCL (job control language) and batch processing.

As Hadoop has matured, multiple processing frameworks have evolved to handle different workloads. Storm was created for real-time, stream processing. Spark was created for interactive processing. Many others were created, each specialized and requiring a different set of technologies and approaches. This often meant creating a solution with one set of technologies and then, as the data grows or service level agreements become more demanding, the solution was refactored into a new set of technologies. This is the modern day computing equivalent of treading water—working really hard just to find out you're in the same place.

Ideally, enterprises want a unified set of open source technologies that help them create cost effective solutions and give them the flexibility to scale their solutions up and down, with the right price for performance ratio, to meet their service level commitments.

In this paper, you'll discover why Anaconda, the leading Open Data Science platform, is the right solution to deliver that range of flexibility and performance. We will:

- Describe ways to deliver high performance on today's modern architectures
- Identify techniques for resolving the computing constraints
- Define the computing constraints
- Show the flexibility of the Anaconda platform to scale with any modern infrastructure

Computing Constraints 101

At the root of all software computing constraints there is one or more of three limitations:

- Compute
- Memory
- I/O

A compute bound or limited problem occurs when the software is throttled by the CPU. If the CPU was able to handle more computational processing, then the program would go faster. This typically occurs in numerically intensive calculations often present in scientific computing and advanced analytics algorithms.

A memory bound or limited problem occurs when the computer is throttled by the amount of memory and/or access rate of the memory. If there was more memory available, then more data could fit into memory, be accessed faster or be used for temporary working space as calculations are performed. This typically occurs when using data that is larger than fits into the memory space available or when a calculation processes a large amount of data from memory. This uses up the balance of memory for temporary working space to perform the calculation. A common memory bound problem is large matrix calculations that underlay many advanced analytics.

An I/O bound or limited problem occurs when the software is throttled by the disk access rate or the network speed. If the disk or network could access data faster, then the program would go faster. Bottlenecks often occur when scanning or reading large amounts of data from a database to perform analytics.

To address these limitations, there are three approaches:

- Scale Up
- Scale Out
- Scale Up and Out

Scale Up is an approach that maximizes the resources on a single machine to achieve the best throughput and performance, given the limitations of that machine. There are multiple techniques to achieving additional performance by scaling up that can be used in combination. Scale Out is an approach that distributes the software workload across many machines to achieve the same goals. Again, there are multiple techniques to achieving additional performance by scaling out that can be used in combination. The combined approach – Scale Up and Out – maximizes the individual resources on a machine for all machines in the cluster.

SCALE UP OPTIONS. Most Python programs are written using a sequential programming model where instructions in the program are executed in the same order that they appear. As an interpreted language, Python code is executed as each line of code is read into memory, and then the instruction is executed on the machine.

As with most programming languages, Python is single-threaded, which means that the Python code is executed linearly from the program start to end. As the data being processed by the Python program grows and/or the computation becomes increasingly more complex, the memory and computational resources are utilized even more and can become fully consumed, which limits the Python program.

"Scale Up is an approach that maximizes the resources on a single machine to achieve the best throughput and performance, given the limitations of that machine."

One approach to initially scaling up the Python program that has bumped into memory and computational limits is to use Python as a "steering language" to guide calls to libraries (i.e.: libraries in the NumPy stack, including SciPy, pandas, scikit-learn and many others) that are not single threaded.

Another approach to scaling up and eliminating the memory and computational constraints is to use a compiler to turn the interpreted code into machine code, which is executed immediately without the overhead of being interpreted. This increases performance more efficiently by using both the memory and CPU.

As memory becomes consumed for either holding additional data or for working space to perform computations, the Python program will once again become constrained by the physical memory. Depending on the machine, additional memory can be added to overcome the constraint. A programmatic technique to address the same memory limitation is "chunking," where a problem is divided into work that will fit into memory and processes the chunks sequentially until the work is complete. This approach is a tradeoff between memory and I/O, since the chunking requires additional reads and writes to process the data.

The next option requires parallel programming, where the program is written so that multiple instructions are performed at the same time. There are three ways to perform parallel programming:

- Multiprocessing: numerous, typically independent processes, each with its own memory, that are executing concurrently or at the same time
- Multithreading: threads are a subset of a process that use the same memory
- Both multithreading and multiprocessing

In this option, multi-core CPUs are used to boost the computational processing power. Once multi-core CPUs are added, the Python program has to be parallelized to take advantage of the additional processing power. While multiprocessing provides greater flexibility to divide and conquer workloads, it also creates additional I/O work since each process has its own memory that must read/write its own data. Multithreading minimizes memory since the threads use the same memory, but overall throughput can be limited by thread locking in the Python interpreter. While it seems obvious to use both multithreading and multiprocessing, the programming difficulty increases dramatically and they are not often used in combination.

Scale Up Approaches

- Use multithreaded analytic libraries
- Compile Python into machine code
- Parallelize analytics
  - Use multiprocessing
  - Use multithreading
- Employ multi-core CPUs
- Leverage GPUs

Visualizing Parallel Computing

To illustrate parallel programming, imagine a convoy of 100 train cars traveling from Austin, Texas to Dallas, Texas on a single track, where cars must travel sequentially until all have arrived at their destination. This is analogous to single-threaded computing. However, if there were four tracks, the 100 cars could be distributed into four tracks of 25 cars each. This is analogous to multiprocessing of independent processes. By distributing the workload across tracks, the 100 cars arrive at their destination sooner than they would have if traveling in a single track. That approach is akin to basic parallel computing or what is known as "embarrassingly parallel."

> "Scale Out is an approach that distributes the software workload across many machines to achieve the same goals."

But, what if we extended this analogy for a train that runs from Boston to New York, through a busier network of rail lines — one with many crisscrossing tracks? In that case, communication between each set of trains is necessary to prevent possible collisions. In the same way, multiprocessing of dependent processes in parallel requires coordination through inter-process communication. This is typically accomplished through an interface that allows the processes to communicate or by a special construct to share memory among multiple processes.

The final Scale Up option is to change or supplement with GPUs (graphical processing units), which are specialized multi-core CPUs, each with their own memory that processes numerically intensive calculations in parallel that address compute bound problems. GPUs easily distribute data across many tiny cores to realize huge performance gains. GPUs require parallel programming via another API, such as CUDA for NVIDIA-manufactured GPUs, which increases program complexity and potentially involves multiple code paths. GPUs do not scale down for smaller data and can actually be slower in throughput for smaller data sets.

SCALE OUT OPTIONS. Scaling out involves connecting multiple machines into a cluster. This may add sufficient processing capability to address CPU constraints or memory limitations.

Alternatively, another approach to initially scaling out a Python program is to distribute data across multiple disks/machines to address I/O problems. This increases performance by eliminating I/O contention for data.

Another approach to scaling out is to minimize data movement on the multiple machines by moving the computations to the data, eliminating I/O bottlenecks. This is accomplished by either using data locality and sending the processing to the data location or by dividing and processing the workload, without marshalling or moving the data to a centralized area for computation

The final technique is to parallelize both the data and workload across multiple machines to eliminate compute, memory and I/O constraints simultaneously.

Scale Out Approaches

- Use cluster
- Distribute workload across cluster

Scaling Up and Out with Anaconda

Big Data, advanced analytics and scientific computing bring exciting opportunities for businesses to leverage more and richer types of data to create meaningful business value and impact. However, this also creates serious computational challenges. To effectively manage this increasingly complex landscape, Data Science teams need technologies that will easily scale and take advantage of the available processing power from their desktop to high performance clusters with the latest advances in chipsets.

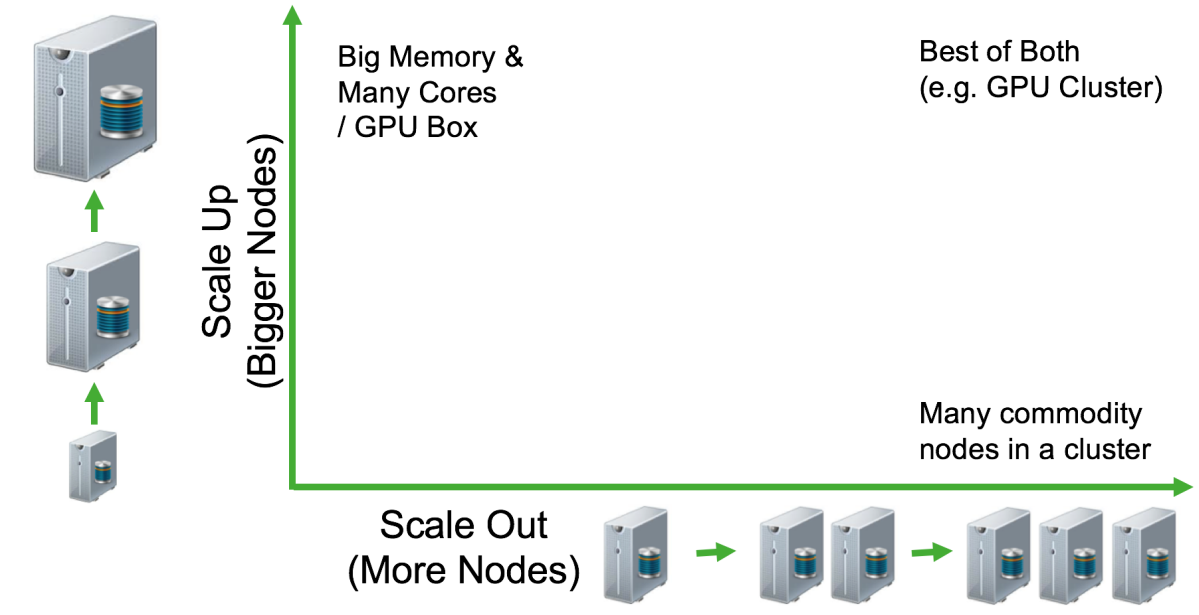
Python is the fastest growing Open Data Science programming language with an incredibly rich open source ecosystem. It is the go-to language for Data Science teams because of the easy learning curve and simplicity in programming and readability, which results in faster development time compared with compiled languages. Because Python is an interpreted language, it is not typically associated with high performance. With today's increasingly large data sets and complex analytics, scaling analytics has become an issue for many Data Science teams.

The Anaconda platform easily delivers high performance analysis to Data Science teams by leveraging investments in all types of infrastructure to both Scale Up and Scale Out workloads. The Anaconda Python distribution is a high performance distribution of Python that has been compiled with the Intel® Math Kernel Library (MKL) that delivers faster throughput on numerically intensive calculations. Additionally, there are a multitude of Scale Up options available in Anaconda that allow Data Science workloads to maximize the investment in single-machine infrastructure.

"With Anaconda, Data Science teams can pushdown processing of analytic pipelines into heterogeneous data stores"

Anaconda allows Data Science teams to turn high level Python scripts into high performance machine code that achieves comparable performance to C/C++ code. Scripts can be pre-compiled so that the machine executable code executes as a process that can be run standalone or embedded into an existing operational process. Alternatively, Python scripts can be compiled on-the-fly into machine code to achieve maximum throughput, even for ad hoc analysis. In either approach, Python scripts can also take advantage of precompiled libraries that have been optimized for the fastest throughput. These boost linear algebra and Fast Fourier Transformation (FFT) operations underpin the most popular Python numerical analysis libraries, including: NumPy, SciPy, scikit-learn and numexpr, by linking with the Intel® MKL. Additionally, powerful and flexible NumPy universal functions created with Anaconda can be automatically parallelized to take advantage of multi-core CPUs and GPUs.

Scale Up v.s. Scale Out Approaches



"Anaconda also includes a robust high performance parallel computing framework that operates with low overhead, latency and minimal serialization for fast numerical analysis."

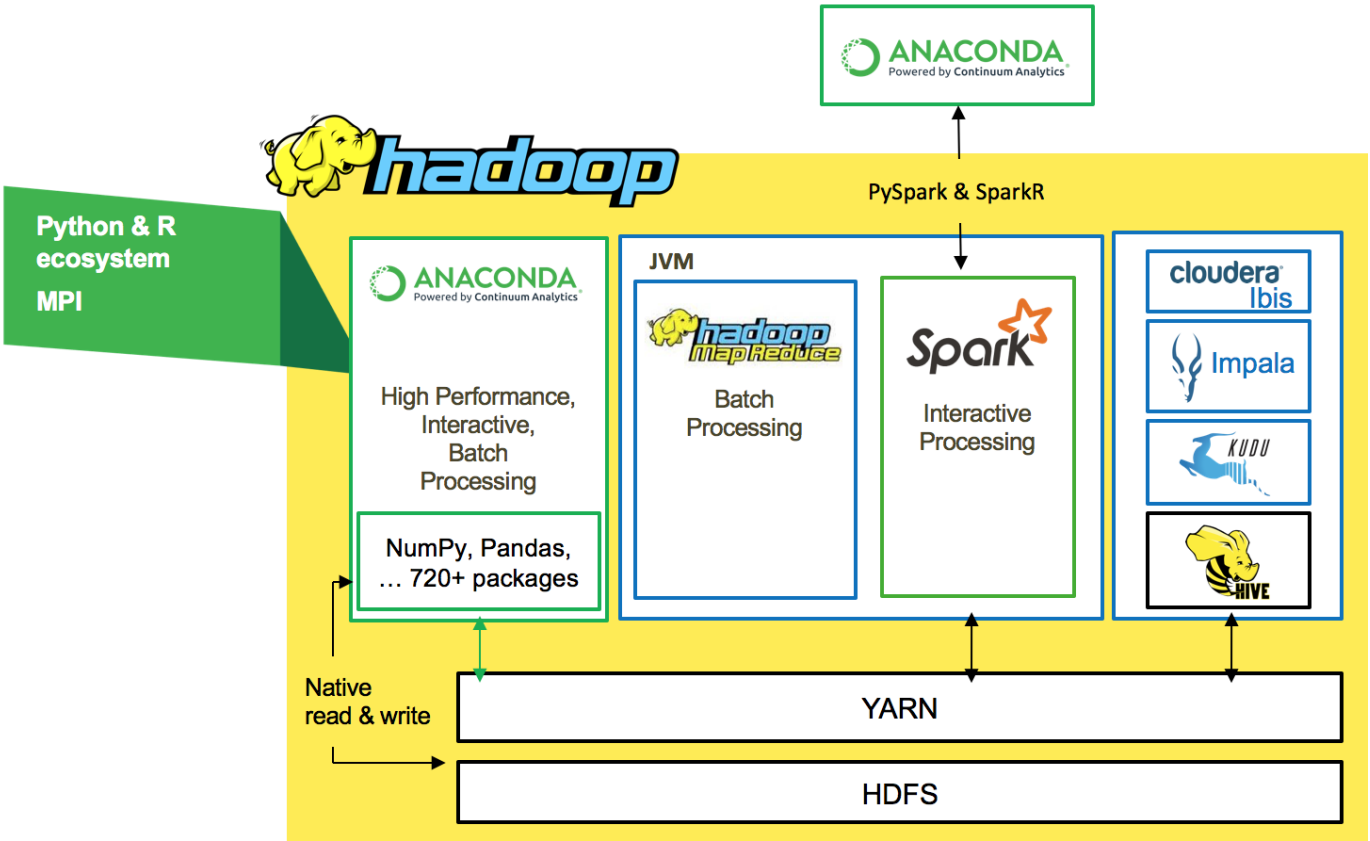
Using Anaconda, Python scripts that leverage mathematically intensive calculations, such as in deep learning algorithms or image processing analytics, can be compiled for fast execution on GPUs, which perform large scale calculations quickly. Plus, Anaconda includes bindings that use the CUDA optimized version for basic linear algebra, FFT, sparse matrices, random number generator and sorting algorithms.

For Intel-based CPUs, there is a special version of Anaconda available that has been compiled with the Intel® C++ Compiler (ICC) that boosts performance further.

With Anaconda, Data Science teams can pushdown processing of analytic pipelines into heterogeneous data stores—CSV, SQL, NoSQL, Hadoop and KDB—moving analytic computations to the data, eliminating data movement to prevent I/O bottlenecks. Pushdown processing can be executed on both single machines and clusters.

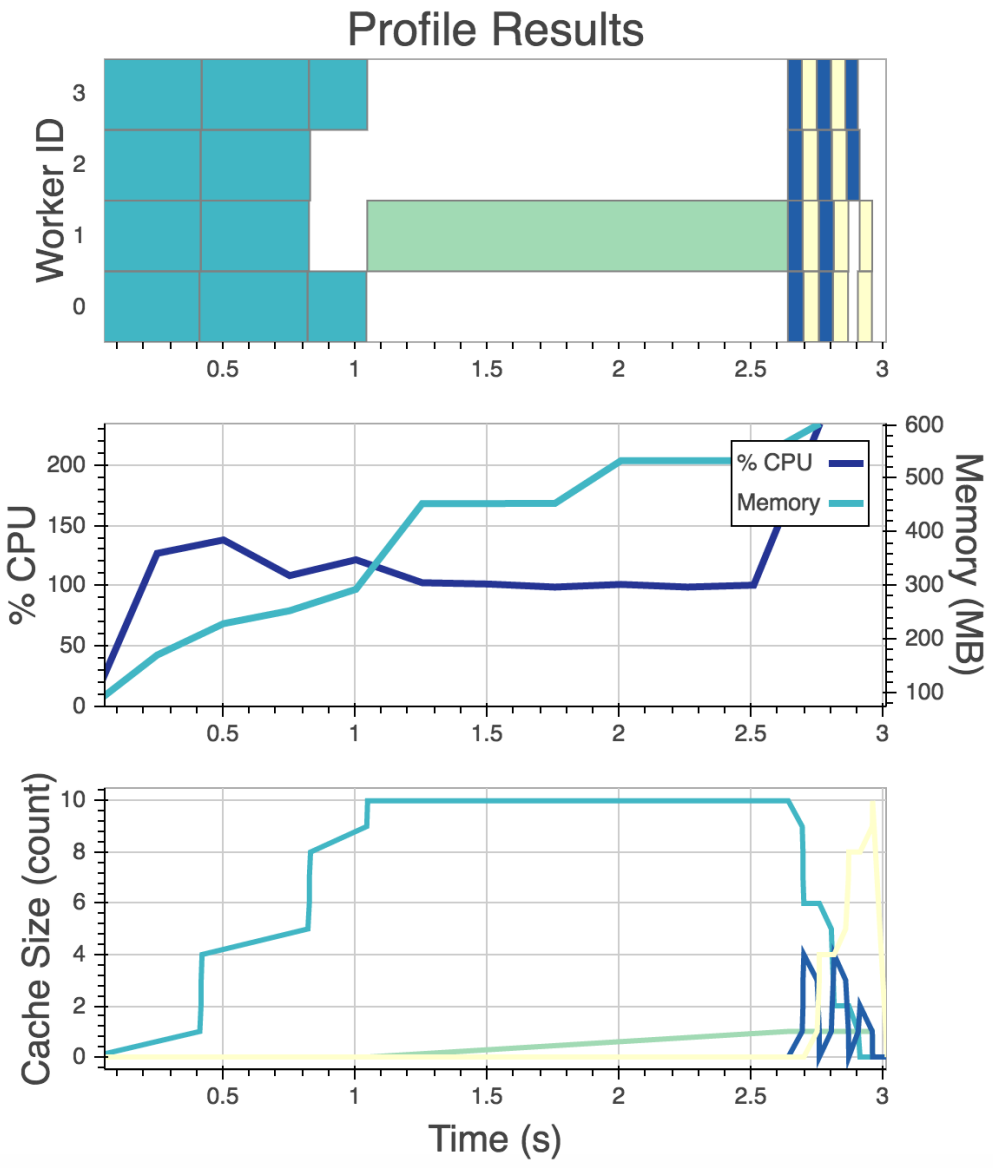
For Hadoop environments, Anaconda can leverage the Spark computing framework to perform either Python or R advanced analytics on a Hadoop cluster. Anaconda also includes a robust high performance parallel computing framework that operates with low overhead, latency and minimal serialization for fast numerical analysis.

Anaconda Scales Out on Hadoop Clusters



Using this framework, new or existing Python scripts executing on single machines and Scale Out architectures, including Hadoop and HPC clusters, can easily be executed using multithreading and/or multiprocessing to quickly process tens or hundreds of gigabytes of data using familiar array or dataframe operations on multi-core CPUs. The workloads are divided and represented as directed acyclic graphs (DAGs) that are task graphs, which characterize how to parallelize computations. The DAGs fully represent the workload and can be executed on the cluster to distribute the computations. Additionally, Anaconda in-notebook, data and code profilers can also be used to identify bottlenecks and points of contention as data and workloads change over time. The DAGs can then be modified to remediate bottlenecks. In this manner, the initial DAG serves as a fast starting point and is flexible enough to be modified and enhanced over time to evolve with the ever-changing data and cluster. The parallel computing framework is resilient and includes multiple resource schedulers—threaded, multiprocessing and distributed—plus custom or third-party schedulers, including Torque, Slurm and YARN. With this flexibility and performance, Anaconda fully maximizes infrastructure investments to accelerate time to value at the best price for performance ratio.

Anaconda DAG Profiler



Summary

The Big Data movement has created opportunities to monetize data from a growing number of internal and external sources. The modern computing era, led by the Big Data movement, is a collision of data warehousing, HPC and advanced analytics that sparked a new technology of fast-moving innovation for high performance and low-cost architectures. This brought open source technology into the forefront to align software costs with lower cost infrastructures.

We've illustrated the complexities and interplay between growing data, physical limitations and programming models, and how that interaction leads to ever-shifting bottlenecks that limit our ability to maximize data, infrastructure and software technology. In this paper, we have:

- Defined the computing constraints
- Identified techniques for resolving the computing constraints
- Described ways to deliver high performance on today's modern architectures
- Showcased the flexibility of the Anaconda platform to scale with any modern infrastructure

Python is an ideal language for Data Science, and the Anaconda platform delivers a robust Open Data Science platform that embraces and maximizes performance for Python, R, Hadoop and a growing list of open source technologies. Anaconda is proven to both Scale Up and Scale Out on numerous architectures—from Raspberry Pi's to high performance clusters—and provides the flexibility to meet an enterprise's changing performance needs with these techniques.

Anaconda, the leading Open Data Science platform, has opened the doors to an exhilarating new era, where Data Science teams can focus on creating high value and impactful Data Science solutions without worrying about being boxed into a corner. Anaconda will exceed Data Science teams' current and evolving scaling requirements easily.