HYPER ACCELERATION WITH BIGSTREAM TECHNOLOGY

Bridging the Gap

After holding true for 50 years, Moore's Law is coming to an end. The law that predicted a doubling in processor transistor count—and hence computing power—every 18 months has ceased to hold true largely because of fundamental physics.

At the same time, the demand for big data and machine learning continues to grow as enterprises look to data for a competitive advantage. Data demands have inspired a new generation of tools, such as Apache Spark™ and TensorFlow, that are pushing advanced analytics into the mainstream.

These tools generally utilize a cluster of servers to handle these large computations, but cluster scaling alone has limits in its ability to provide high performance. Scale-up and scale-out strategies can work effectively for smaller workloads, but they run into diminishing returns when cluster sizes (scale out), or server capability (scale up) grow larger.

Hardware acceleration such as graphics processing units (GPUs) or field-programmable gate arrays (FPGAs) provide a vehicle for high performance and, in fact, enhance the gains of scaling. Historically, however, acceleration has had limited deployment due to a key gap between data scientists and the performance engineers working with the computing infrastructure, illustrated by Figure 1.

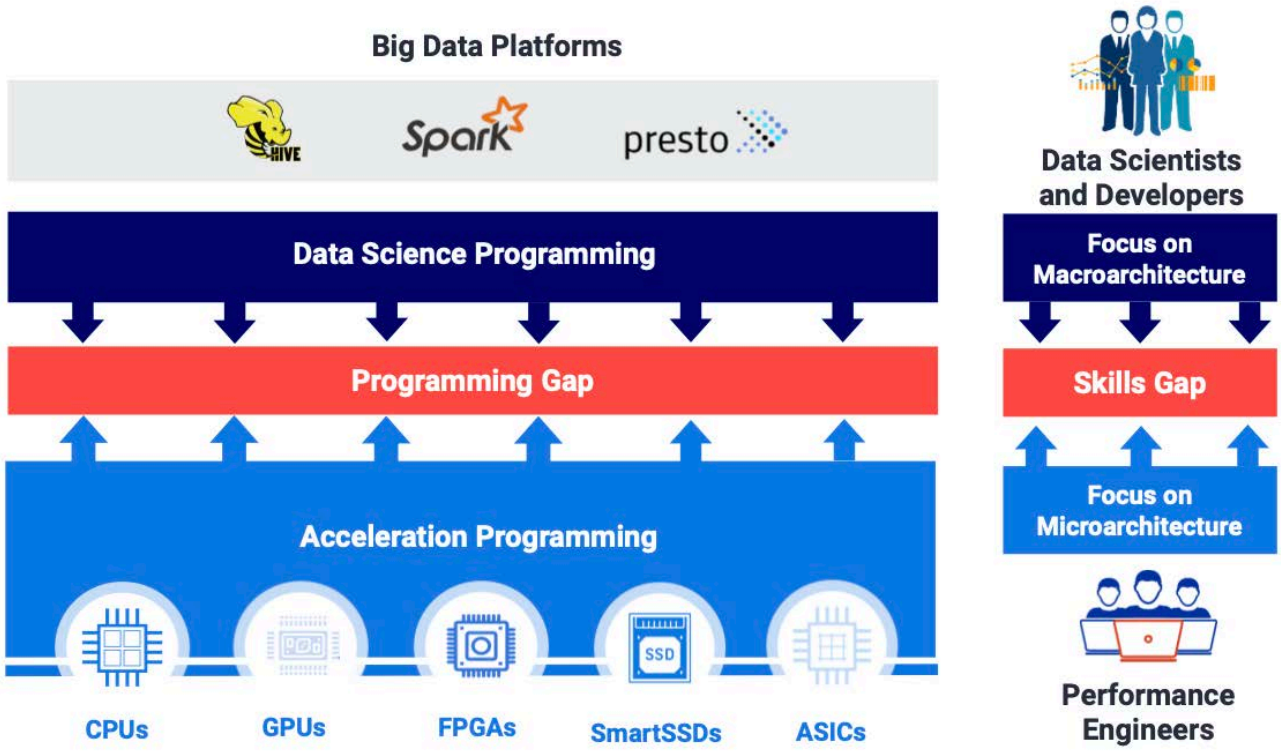

Figure 1: Programming Model Gap Inhibiting Hardware Acceleration

Until recently, there was no automated way for big data platforms such as Spark to leverage advanced compute hardware. Consequently, data scientists and analysts had to work with performance engineers to fill that programming model gap. Though feasible, this process was inefficient and time-consuming.

Data scientists, developers, and quantitative analysts are accustomed to programming using big data platforms in a high-level language. Performance engineers, on the other hand, are focused on programming at a low level, including field-programmable hardware. Thus, the scarcity of resources, along with additional implementation time, would significantly lengthen time to value of analytics when accelerating. In addition, the resulting solutions would typically be difficult to update as analytics evolve.

Figure 2 illustrates how Bigstream architecture address this gap.

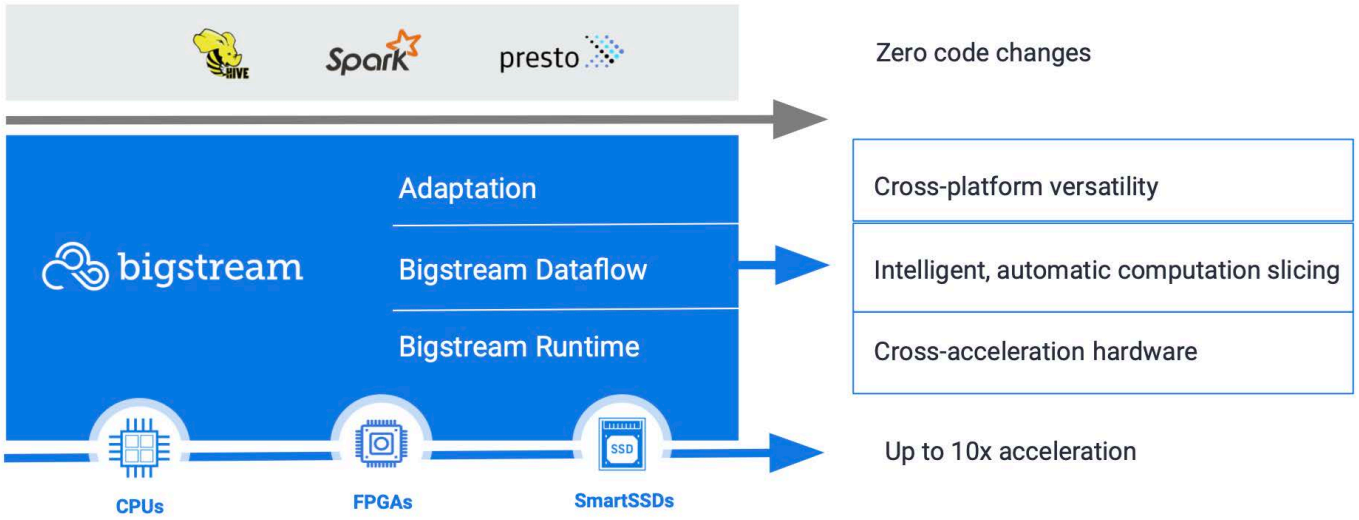

Figure 2: Bigstream Hyperacceleration Addresses the Gap

Bigstream Hyperacceleration automates the process of acceleration for users of big data platforms. As shown in the architecture in Figure 2, Bigstream interacts with both the data platforms and the underlying processors. It includes compiler technology for both software acceleration via native C++, and FPGA acceleration via bitfile templates. This technology yields up to 10x end-to-end performance gains for analytics, but with zero code change.

Performance Analysis

Cluster Scaling with Acceleration

When organizations need to increase their cluster performance, they typically scale by growing the cluster. “Scaling up” refers to increasing the capability of the existing cluster nodes.“Scaling out” refers to adding additional nodes to the existing cluster. It is also common to combine scale up and scale out.

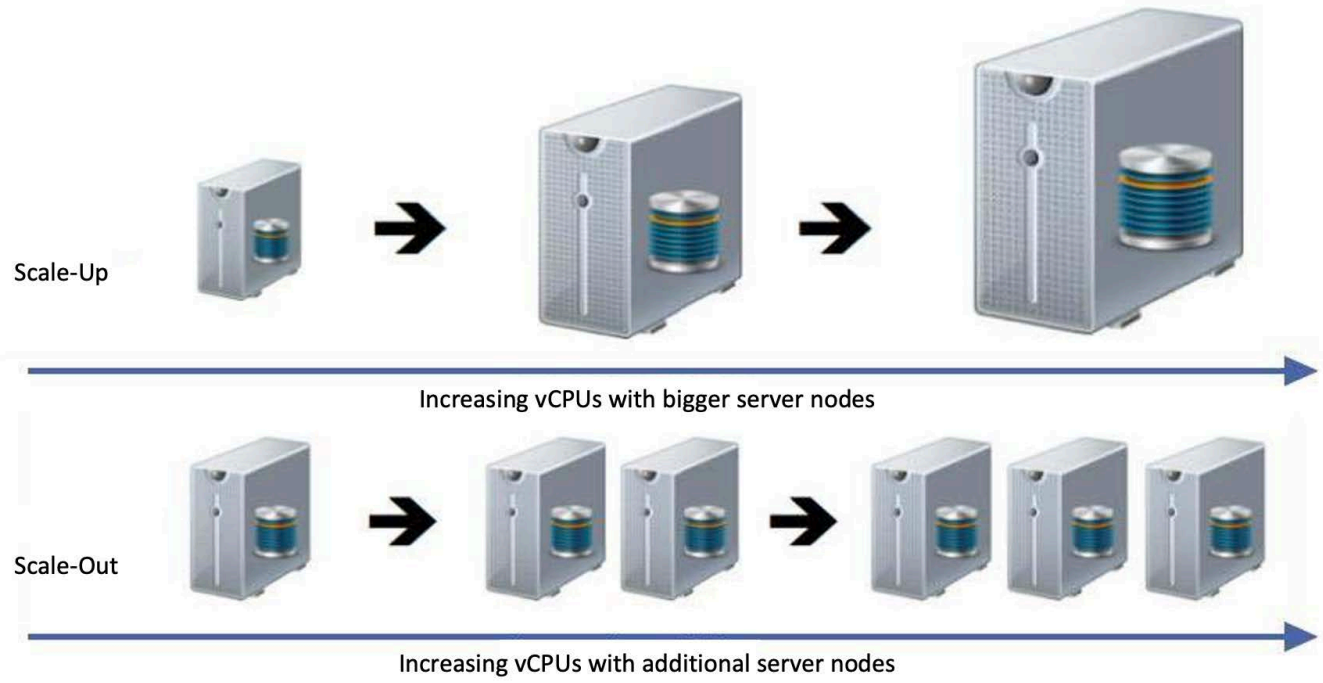

Figure 3: Figure 3: Scaling Up vs. Scaling Out

Figure 3 illustrates these two approaches to scaling. In this example, both approaches increase the number of virtual CPUs (vCPUs), increasing the computing power. It is also possible to scale in other ways, such as adjusting network connections, memory, disk, or other resources.

In almost all cases, scaling yields a sub-linear performance increase as resources are added. That is, as the cluster is scaled by a factor of N, performance increase is less than N.

This diminishing return is severe at large scale for the following reasons:

Scale Up

- Increased I/O overhead and throttling
- Shared resource contention (memory and L2 cache)
- Increased scheduling complexity

Scale Out

- Increased network overhead
- Exacerbated straggler effect
- Increased failure rate

Bigstream Hyperacceleration addresses these in two primary ways:

1. Reduces the cluster size required to yield a given performance level
2. Reduces network and I/O overhead and the resulting impact

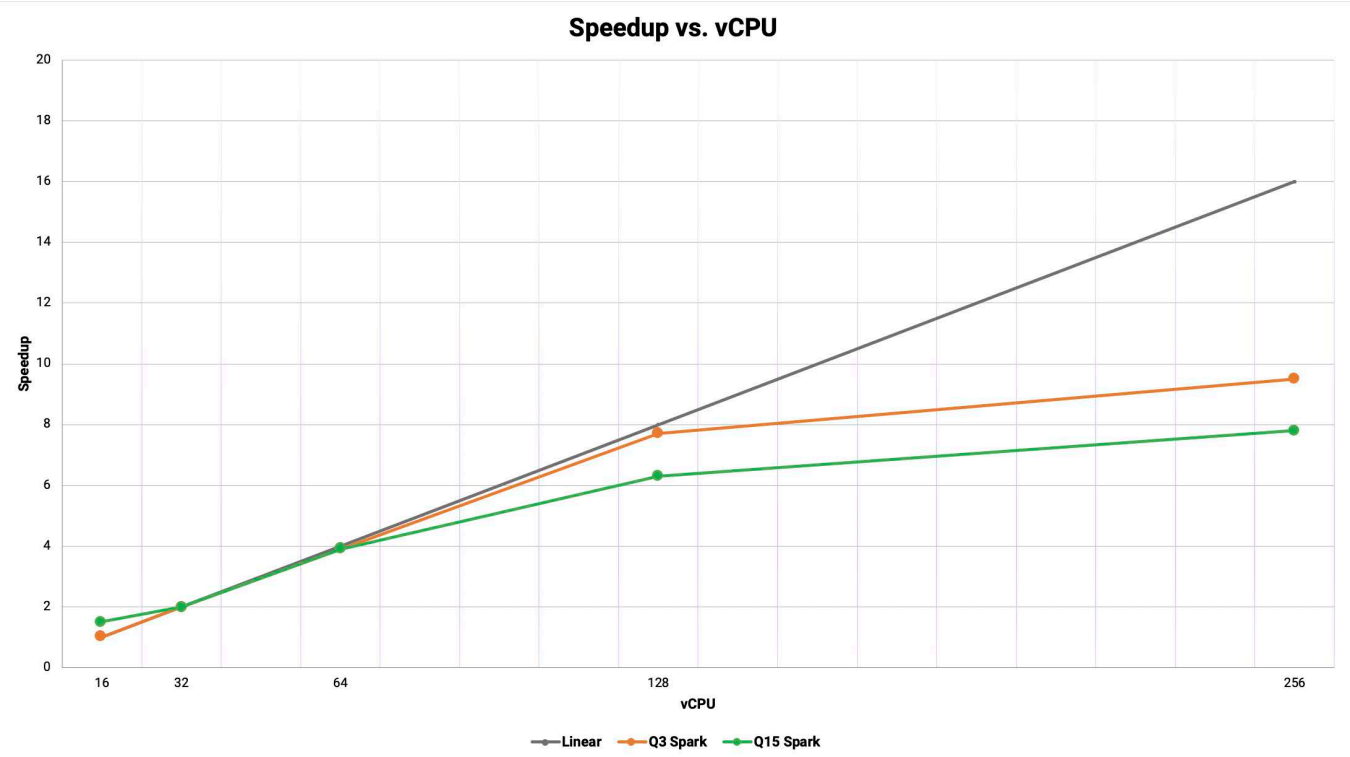

Figure 4: The Scaling Issue—Experiment Results

Figure 4 illustrates the scaling issue. This example includes scale-up experiments. We ran two TPC-DS benchmark queries on Amazon EMR using Spark with several different cluster sizes. Moving from left to right, each point represents the performance for a given

number of vCPUs. The cluster size doubles with each step to the right (16; 32; 64; 128; 256 vCPUs). The vertical axis, “Speedup,” is calculated relative to the “Base.” The speedup performance of both benchmarks fall off from the linear path of the gray line as the cluster scales, as predicted above.

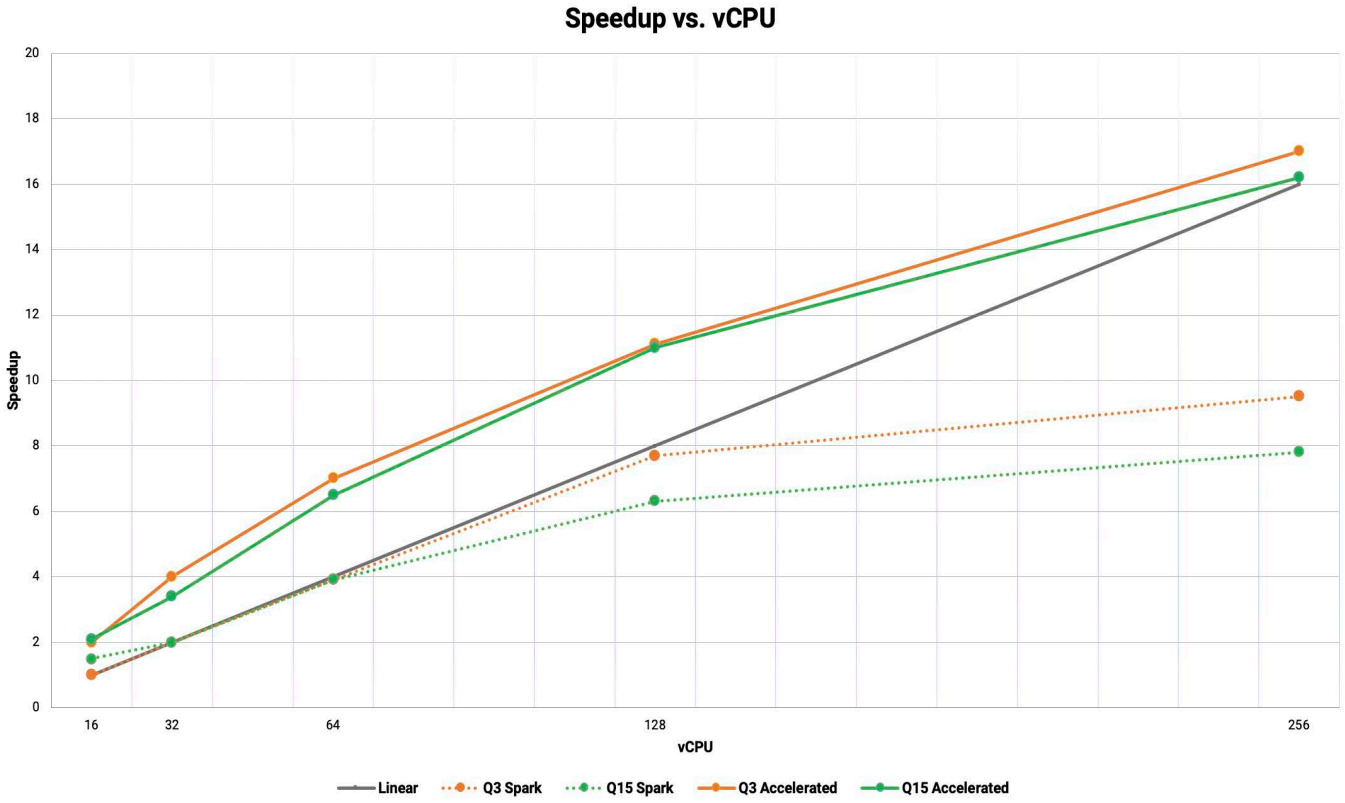

Figure 5. Results of Scale Up Experiment with Spark and Accelerated Spark

Next we ran the same Spark jobs on clusters equipped with Bigstream software-based acceleration. Figure 5 shows the combined results—the dashed lines are the unaccelerated runs from Figure 4, and the solid lines are Bigstream accelerated.

The accelerated curve displays a much more gentle falloff with scaling than the Spark curve. So for a given cluster size, Bigstream generates significantly more speedup than Spark alone (the vertical difference between the two lines). Another way of analyzing this, though, is to note the horizontal distance and the fact that Bigstream provides nearly the same speedup at 64 vCPUs as Spark does alone with 256 vCPUs.

We see similar results with scale-out experiments. These results indicate that acceleration can work synergistically with scaling, to provide maximum performance and a wide variety of performant configuration choices for the user.

This, in turn, can result in total cost of ownership (TCO) savings. Cloud users can use smaller clusters or use the same cluster for a faster, less expensive processing time. Users with on-premises clusters will be able to run faster applications and accomplish more in a given time period.

FPGA Acceleration

As discussed, hardware-based acceleration has the highest performance potential. Adding an FPGA to a server can be a cost-effective way to speed up big data platforms, if it doesn't come with additional complexity. These chips are typically a fraction of the cost of a full CPU-based server. Bigstream eliminates the complexity, delivering a zero-code-change solution.

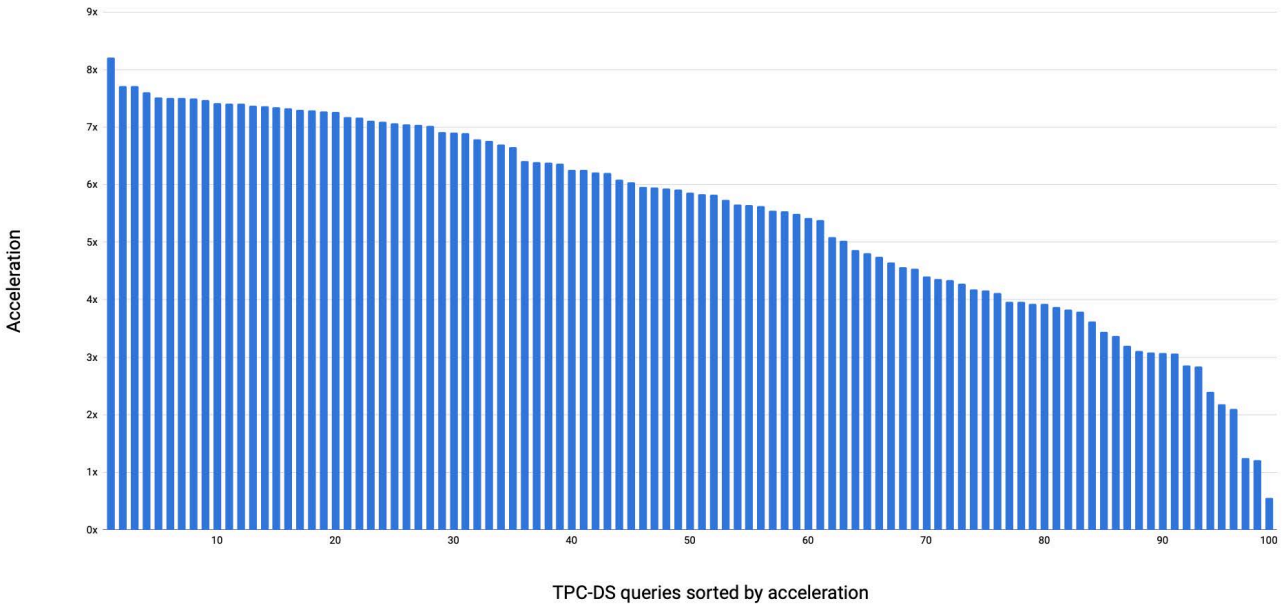

Figure 6: Bigstream Acceleration Benchmark Tests

Figure 6 illustrates the results of a separate set of benchmark tests comparing Spark alone versus Spark with Bigstream-enabled FPGA acceleration. 100 TPC-DS benchmark queries were run with Spark in a CPU-only platform as the baseline, and then again using Spark along with Bigstream and a commodity FPGA platform.

The average speedup across the 100 queries was 5.5x, with some queries as much as 7-8x faster with Bigstream and FPGAs. As with all Bigstream Hyperacceleration, no Spark code needed to be changed in the accelerated runs.

Technical Overview

This section presents a technical overview of Bigstream Hyperacceleration applied to Spark and the role of its components. We focus on its relationship to the standard Spark architecture and how it transparently enables acceleration.

Baseline Spark Architecture

Figure 7 shows the basic components of standard Spark and its resource management across a cluster of CPU nodes.

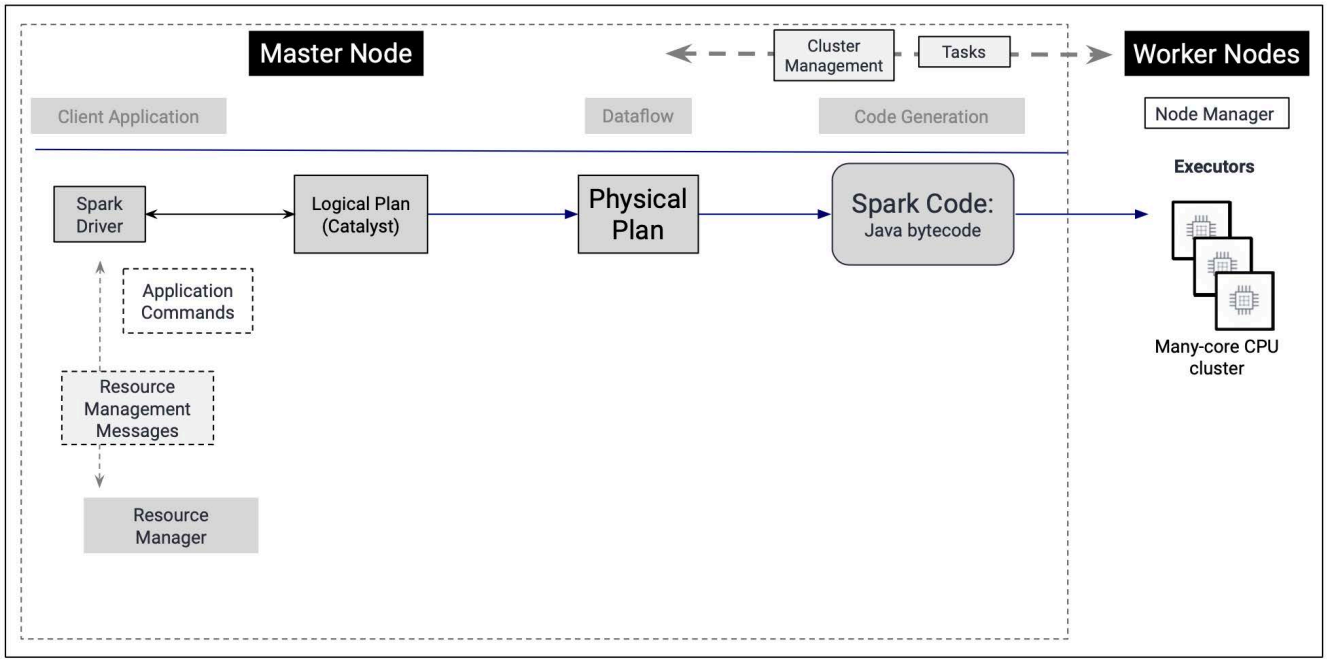

Figure 7: Baseline Spark Architecture

The Spark components and associated roles are as follows:

- **Spark Driver** - Clients submit Spark applications, including user code and configuration files, via the Driver. The configurations define the number of Master and Core nodes and associated memory allocations.
- The **Spark Master** process requests cluster resources to make them available to the Driver. The Resource Manager, in turn, allocates resources for Executor creation. The Master generates code for each stage of the application and distributes it to the Executors. Tasks are created as Java bytecode at runtime and downloaded to the Executors for execution.

- **Spark Executors** are hosted on Worker Nodes and run the individual Spark tasks. The computation proceeds in stages, generating parallelism among the Executors. The faster that the Executors can execute their individual task sets, the faster stages can finish, and therefore the faster the application finishes.
- The **Physical Plan** is a data structure produced by the Master Node that fully describes the analysis in a Directed Acyclic Graph (DAG). The Physical Plan dictates how the application will be executed on the cluster. Unlike the Logical Plan, it optimizes the detailed operations and their order, such as the type of a Join and when to introduce a Filter or GroupBy.
- **Catalyst Optimizer** runs to further optimize the Physical Plan performing edits to it such as redundancy elimination, reordering etc. Spark first creates an unresolved Logical Plan, then further logical Plans, and ultimately a Physical Plan. Catalyst is part of that process. It actually generates multiple Physical Plans and determines the optimal plan.

Bigstream Hyperacceleration Architecture

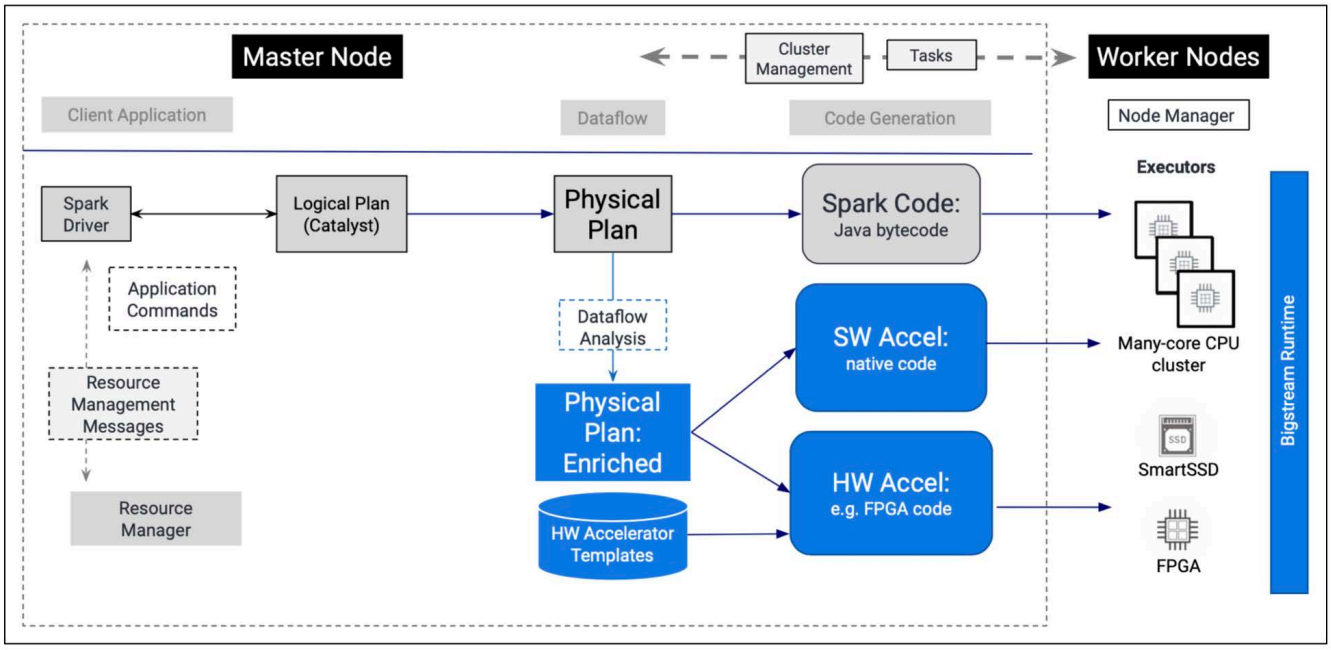

Figure 8: Bigstream Hyperacceleration Architecture

Figure 8 shows the Spark architecture with Bigstream acceleration integrated. Note that this illustration applies equally to software and hardware (many-core, GPU, and FPGA) acceleration. The blue boxes indicate Bigstream Hyperacceleration components that are added at bootstrap time and provide acceleration throughout multiple application executions.

The Client Application, Spark Driver, Resource Manager components, and the structure of the Master and Executors all remain unchanged.

The Physical Plan including any optimizations introduced by Catalyst remain unchanged. The standard bytecode for Spark tasks are generated by the Master as normal.

At stage execution time, a hook is added to all **Executors** that enables a pre-execution check of whether a stage can be accelerated. If so, the associated compiled module is called. If not, the standard Java bytecode version is executed.

Note that this illustration applies equally to software and hardware (many-core, GPU and FPGA) acceleration. Bigstream Hyperacceleration does not require changes to anything in the system related to fault tolerance, storage management or resource management. It has been carefully designed only to provide an alternative execution plan at a node level that is transparent to the rest of Spark. The key additions are:

- The **Bigstream Compiler** evaluates each individual stage of the Physical Plan for potential execution acceleration. It generates accelerated versions of the stages where possible, which call into the Bigstream Runtime API.
- **Bigstream Runtime** is a set of natively compiled C++ modules (software acceleration) and bitfile templates (FPGA acceleration) and their associated APIs that implement accelerated versions of Spark operations.

Other than improved performance, the programmer doesn't know whether a stage is running accelerated. Basically, Bigstream determines which stages can be accelerated and runs standard Spark for them if not. This approach ensures that Bigstream users continue to see an identical interface to standard Spark.

Delivering Solutions to Real-World Challenges

Bigstream continues to develop its product, focusing on the biggest opportunities for acceleration of the Spark platform. The current product has accelerated a broad range of functions and technical and business use cases. ETL and ELT workloads have seen some of the most dramatic acceleration, as have SQL analytics with Hive and Spark SQL.

Bigstream developed Hyperacceleration to solve real-world big data problems. With this state-of-the-art technology comes a significant return on investment (ROI) and staggering performance gains.