Applied Accelerated Artificial Intelligence

Prof. Saurav Agarwal
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Performance Boosting ETL Workloads Using RAPIDS on Spark 3.0
Lecture - 52
Accelerated ETL Pipeline with Spark part 1

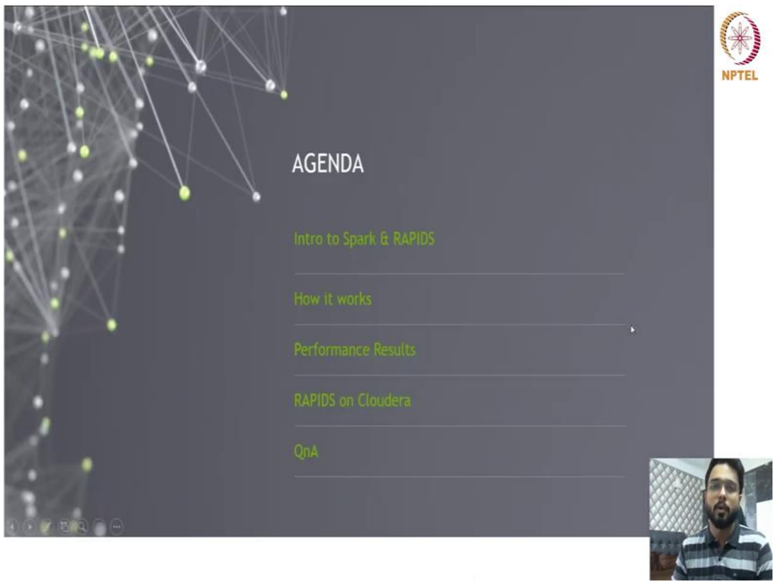
(Refer Slide Time: 00:15)

NVIDIA

PERFORMANCE BOOSTING ETL WORKLOADS USING RAPIDS ON SPARK 3.0

Hey hi everyone, this is Saurav Agarwal again. So, I am working as a senior solutions architect at NVIDIA. So, today I will be going through Performance Boosting ETL Workloads Using RAPIDS On Spark. So, I will be covering the following agenda today.

(Refer Slide Time: 00:33)


AGENDA

Intro to Spark & RAPIDS

How it works

Performance Results

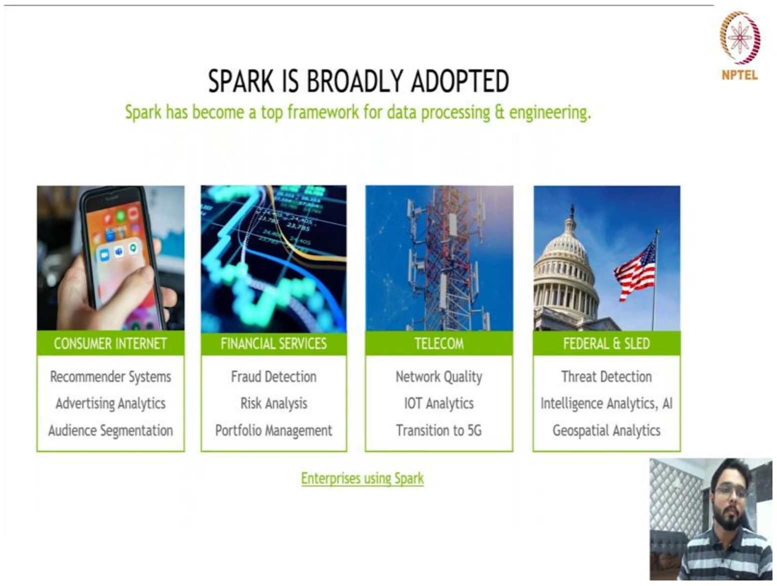
RAPIDS on Cloudera

QnA

So, I will start with what is Spark all about where it is applicable and what is RAPIDS platform all about on Spark, how it works in detail. So, I will get in depth of how the acceleration on GPU works on Spark using the NVIDIA GPUs and NVIDIA RAPIDS SDK. Then I will deep dive into performance results where I will compare the CPU versus GPU benchmarks of various types of queries or data processing workflows using Spark on CPU versus Spark on GPU.

And at the end I will cover RAPIDS on Cloudera data platform which is one of the most famous and used data platform on how we can use the rapids Spark on that. I will try to keep the session very basic, but there are some intricacies or Spark I will go through. So, in case you are not aware of spark. So, please bear with me it is very difficult to cover and two and Spark in one session though I will try to make it as simple as possible from the listener point of view.

(Refer Slide Time: 01:48)



SPARK IS BROADLY ADOPTED

Spark has become a top framework for data processing & engineering.

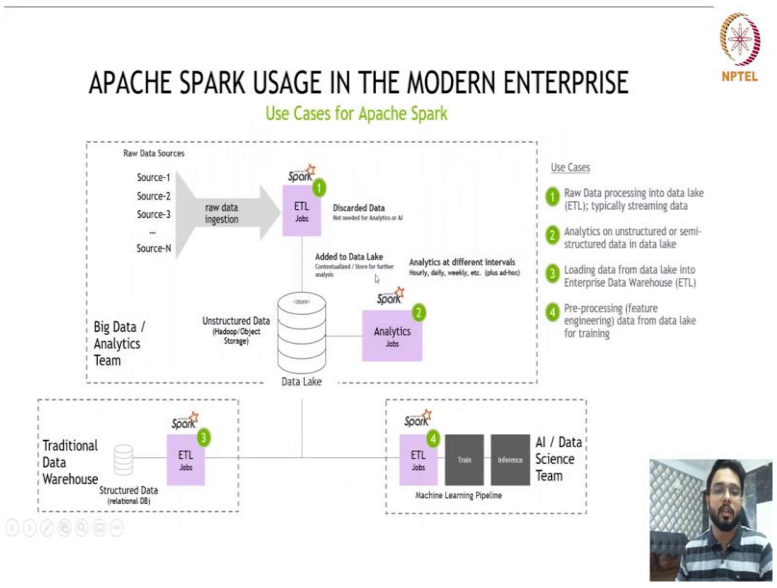
CONSUMER INTERNET	FINANCIAL SERVICES	TELECOM	FEDERAL & SLED
Recommender Systems	Fraud Detection	Network Quality	Threat Detection
Advertising Analytics	Risk Analysis	IOT Analytics	Intelligence Analytics, AI
Audience Segmentation	Portfolio Management	Transition to 5G	Geospatial Analytics

Enterprises using Spark

So, lets start with where Spark is adopted. So, we know that Spark has become a top framework for data processing and data engineering across enterprises. So, if there is big data then there is Spark, because Spark is not only scalable, but also is a Swiss army knife and very robust to do data processing any kind of data extraction, data transformation, data loading from A system to B system or X system to Y system.

So, be it consumer internet companies, be it financial services telecom, federal everywhere we have use cases using Spark as the ETL or the data processing framework. So, like recommender systems, advertising analytics, audience segmentation, fraud detection, IoT analytics, geospatial analytics. So, you name it and wherever we have data in volumes, velocity and variety we have big data processing, we have Spark there.

(Refer Slide Time: 02:52)



APACHE SPARK USAGE IN THE MODERN ENTERPRISE

Use Cases for Apache Spark

Use Cases

1. Raw Data processing into data lake (ETL); typically streaming data
2. Analytics on unstructured or semi-structured data in data lake
3. Loading data from data lake into Enterprise Data Warehouse (ETL)
4. Pre-processing (feature engineering) data from data lake for training

Raw Data Sources
Source-1
Source-2
Source-3
...
Source-N
raw data ingestion
Spark
ETL Jobs
Discarded Data
Not needed for Analytics or AI
Added to Data Lake
Contextualized / Store for further analysis
Analytics at different intervals
Hourly, daily, weekly, etc. (plus ad-hoc)
Big Data / Analytics Team
Unstructured Data (Hadoop/Object Storage)
<store>
Data Lake
Analytics Jobs
Traditional Data Warehouse
Structured Data (relational DB)
Train
Inference
AI / Data Science Team
Machine Learning Pipeline
NPTEL

So, how a traditional or a kind of a usual modern enterprise users Spark for the data processing and ETL workflows? So, if you see this slide here we may have n number of sources it can be say IoT data it can be CRM data, it can be data coming from sales source systems. So, a customers are directly putting some forms entering the data. So, it can be from a web app or a mobile app anything.

So, first step is to ingest the data do some ETL and discard all the bad quality data or data which is not needed for the analytics purpose. So, and then we store the data in something called a data repository or a data lake. So, once we store the data in the data lake then we can have multiple use cases on top of it. The first use case be it can be; it can be a analytics or on the unstructured or semi structured data, in the data lake. So, if we need to do some raw data analytics then we can do it on top of the data lake itself.

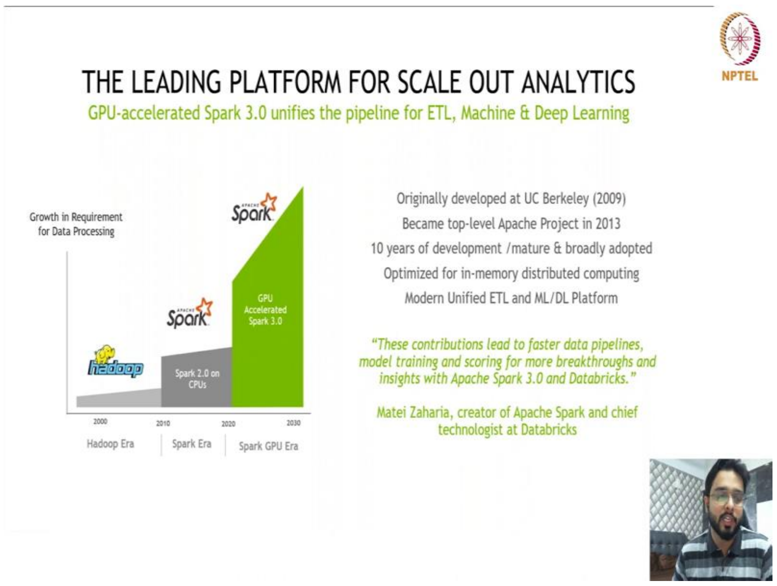
But apart from that we can do two more things, we can have a data warehousing system where what we do is we de normalize the data and we make it more and more usable for structured relational databases kind of stuff where we can run large analytics query which is more of a historical descriptive analytics queries which may run say suppose in the last 10 years of data, last 20 years of data and give us some analytical insights.

So, we call that kind of where data repository as a data warehouse and Spark can be used to build that data warehouse and those queries on top of it as well. And then finally, we have the data science or the ML AI team where they can use the same data for training, model training and inferencing. So, we need the first time historical data taking in and doing the model training suppose they want to build a channel analytics model and build a XG boost kind of model.

So, they can take all the data and run the XG boost model training on top of it to create that model and then after that for the inferencing also, the real time data is coming and we need to infer at real time we can leverage something like Spark streaming to stream the data and do the ETL in real time and do the inferencing as well. So, there are multi folds use cases in an organization be it raw data processing, be it analytics on the unstructured data or loading the data into the lake for data warehousing and finally, pre-processing the data for the data lake for training.

So, all these functions where we have need for a data processing and b data analytics we need apache Spark as a base framework; why? Because, traditional frameworks like pandas or SQLs or similar tools do not scale I mean if the data volume goes up in gigabytes or hundreds of gigabytes or terabytes or petabyte then a single system or a single server will not be enough. Hence, we need something which is distributed which is real time which can process both structured and unstructured data as well. So, Spark is one Swiss army knife which can do all of this together.

(Refer Slide Time: 06:24)



THE LEADING PLATFORM FOR SCALE OUT ANALYTICS

GPU-accelerated Spark 3.0 unifies the pipeline for ETL, Machine & Deep Learning

Originally developed at UC Berkeley (2009)
Became top-level Apache Project in 2013
10 years of development /mature & broadly adopted
Optimized for in-memory distributed computing
Modern Unified ETL and ML/DL Platform

"These contributions lead to faster data pipelines, model training and scoring for more breakthroughs and insights with Apache Spark 3.0 and Databricks."

Matei Zaharia, creator of Apache Spark and chief technologist at Databricks

So, if you might know about big data processing you might know about the word Hadoop. So, Hadoop era started in the early 2000s where Google and Yahoo released multiple papers on big table and map reduce. And similar frameworks, where they used to process terabytes and petabytes of data, but it was very slow. So, map reduce framework or the Hadoop framework as a whole was very good to process batch data, but the processing was so slow that it took sometimes even days together to process some new volume of data.

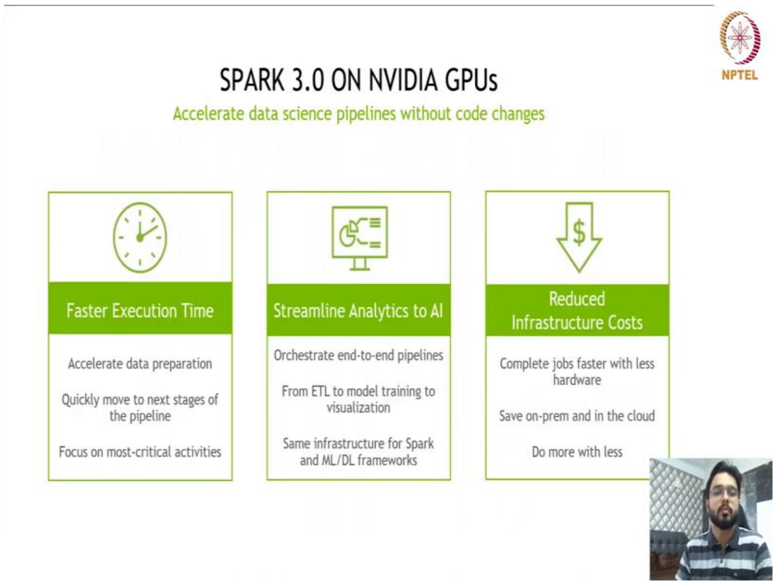
So, after that in the early 2000s 10 era Spark came into the picture and then many organizations felt that there is a need of faster data processing; hence, Spark leverage the memory that the RAM instead of that hard disk to do the intermediate data processing. So, the entire workflow instead of being run on CPU plus disk, it ran on CPU plus RAM and finally, if the there is a need to write back the final process data as the output to the disk.

So, there is a huge saving from the perspective of read and write to the disk because reading and writing to the disk is very slow, though our processing capabilities have increased multifold our RAM has the RAM capacity has increased multifold, but the speed of reading and writing data from a disk or to a disk is still not that great. Hence, the Hadoop framework was really slow to be used to do data processing and data analytics on top of it.

Hence, the Spark became very popular and it became. So, popular that almost 80 to 90 percent workloads got migrated from map produced to Spark. And then finally, after this Spark 2 eras, we at NVIDIA are envisioning that the next set of say the next set of accelerated analytics era on Spark will be on Spark itself. There will be no new tool which will be coming like Spark, but it will be Spark itself, but it will be highly GPU accelerated and it will be based on previous optimizations done on the open source code of Spark on the GPUs.

So, here since we first moved from disk to memory, now we are moving from memory to GPU plus GPU memory. Hence, we think that the next wave of accelerated computing or data processing will be on Apache Spark's deal on GPUs.

(Refer Slide Time: 09:27)



SPARK 3.0 ON NVIDIA GPUs

Accelerate data science pipelines without code changes

Faster Execution Time
- Accelerate data preparation
- Quickly move to next stages of the pipeline
- Focus on most-critical activities

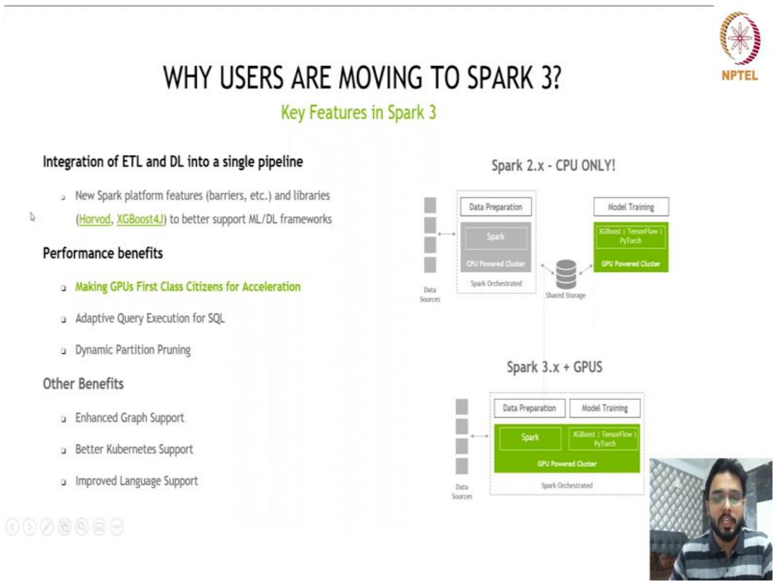
Streamline Analytics to AI
- Orchestrate end-to-end pipelines
- From ETL to model training to visualization
- Same infrastructure for Spark and ML/DL frameworks

Reduced Infrastructure Costs
- Complete jobs faster with less hardware
- Save on-prem and in the cloud
- Do more with less

So, there are three main advantages of running Spark 3 on NVIDIA GPUs first is yes execution time is faster. So, accelerated data preparation be it for machine learning, be it for data warehousing, be it for data analytics; quickly move to the next stages of the pipeline focus on the most critical activities. Second will be streamlining analytics to AI, so orchestrating end to end pipelines from ETL to model training to visualization same infrastructure for Spark and machine learning deep learning frameworks.

And finally, reducing infrastructure cost where we can complete jobs faster with less hardware. So, we will save a lot of cost there. So, suppose if you can replace 100 CPU nodes with say if 30 GPU nodes. So, imagine the amount of cost you are saving for the organization. So, even if the GPU node are 1.5 x costly, but since you are reducing the number of nodes by 3 x still you are at the positive total cost optimization state.

(Refer Slide Time: 10:38)



WHY USERS ARE MOVING TO SPARK 3?

Key Features in Spark 3

Integration of ETL and DL into a single pipeline

- New Spark platform features (barriers, etc.) and libraries (Horvod, XGBoost4J) to better support ML/DL frameworks

Performance benefits

- Making GPUs First Class Citizens for Acceleration
- Adaptive Query Execution for SQL
- Dynamic Partition Pruning

Other Benefits

- Enhanced Graph Support
- Better Kubernetes Support
- Improved Language Support

So, first of all a lot of organizations are still using Spark 2, I did some survey recently and I found that on a high level 50 percentage of the organizations are on still Spark 2, the reason being there is a; there is a tech depth that they need to work upon that. That means, that they do have the plan, but they do not have enough resources to migrate the jobs, because there are some breaking changes as well.

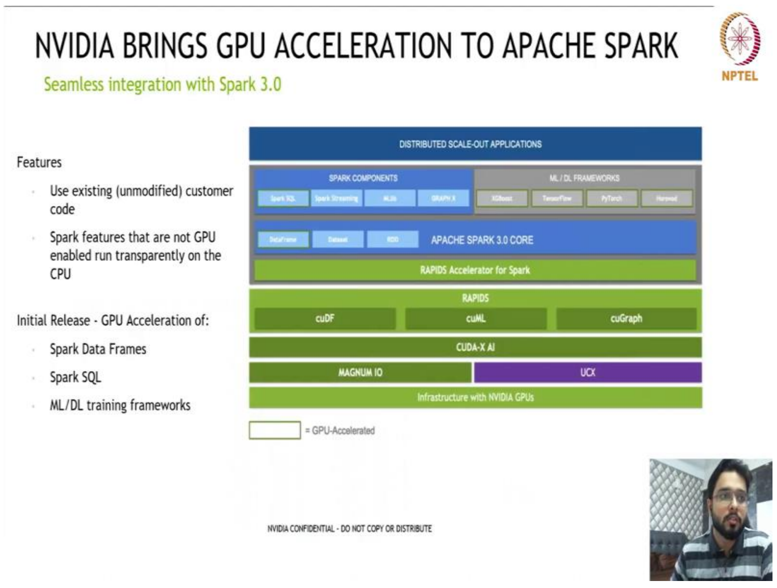
And this is the biggest reason, but the breaking changes are mainly 5 to 10 percent. It is not more than that. If the organizations see these benefits definitely they will prioritize the 50 percent which have not done yet to migrate their workflows to Spark 3. Now, if you see this Spark 3 advantages basically without GPUs at all is that you can have adaptive query execution; that means, that automatically it will optimize the SQL query execution or the processing if based on the amount of memory amount of CPU it needs.

Then we have dynamic partition pruning; that means, we will only prune or we will only scan the data which is needed for the analysis not go to the entire memory or entire disk you are to check where the data is lying and all that stuff because, we do have that partition pruning which is very dynamic in nature in the Spark 3. So, apart from that we do have graph support, Kubernetes support which will give you better orchestration and better language and coverage of all the operators we can think of in SQL and data processing world.

However, apart from that, if we just migrate from CPUs Spark 2 to CPUs Spark 3, we have seen based on various benchmarks that you will get at least 2 x acceleration. However, it is not it does not ended there if you add the GPU flavor to it you will further get at least on top of that 2 x you will get further 3 x to 4 x at least on an average acceleration, plus you will also get one single cluster for both your data preparation and model training.

Because, starting this Spark 3 and the GPU support you can train your models on Spark, basically if they are based on XG boost, TensorFlow and PyTorch based models you can use TensorFlow on Spark or frameworks like Horvod on Spark to do the machine learning or deep learning and you can do Spark on the same cluster for the data preparation. Hence, we can call it that ok we can use the same cluster for data preparation and ML DL as well.

(Refer Slide Time: 13:23)



NVIDIA BRINGS GPU ACCELERATION TO APACHE SPARK

Seamless integration with Spark 3.0

Features

- Use existing (unmodified) customer code
- Spark features that are not GPU enabled run transparently on the CPU

Initial Release - GPU Acceleration of:

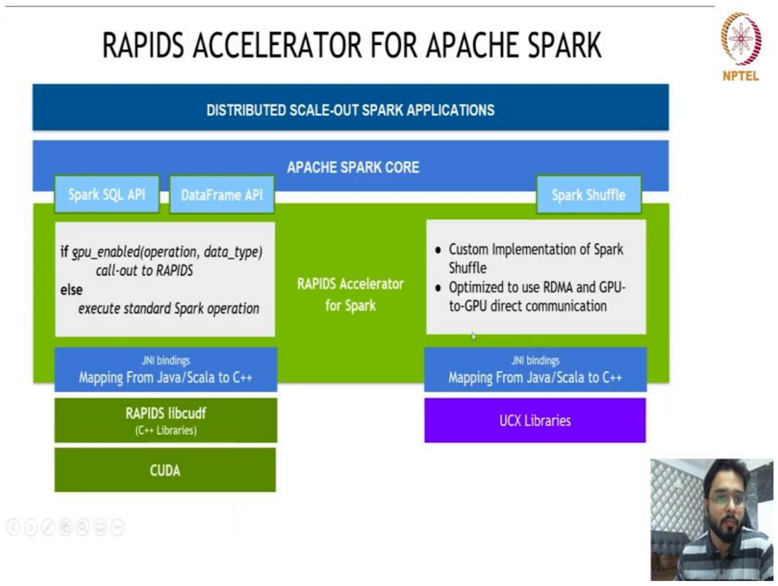
- Spark Data Frames
- Spark SQL
- ML/DL training frameworks

NVIDIA CONFIDENTIAL - DO NOT COPY OR DISTRIBUTE

So, NVIDIA brings GPU acceleration to Apache Spark. So, here we you can see that seamless integration with Spark 3 has been done, where we are using unmodified customer code or your code is completely unmodified do not need to change or make any configuration changes in the code itself. So, only change is the underlying cluster if we use a GPU cluster rather than a CPU cluster its more than enough to do the processing on GPUs.

So, however, from the coverage perspective it covers Spark data frame, it covers Spark SQL, it covers machine learning deep learning frameworks like PyTorch, XG boost, PyTorch for deep learning, TensorFlow or deep learning; Horvod for deep learning and XG boost for machine learning. I mean from the perspective of classification regression and classical machine learning.

(Refer Slide Time: 14:20)



RAPIDS ACCELERATOR FOR APACHE SPARK

```
if gpu_enabled(operation, data_type)
     call-out to RAPIDS
else
   execute standard Spark operation
```

- Custom Implementation of Spark Shuffle
- Optimized to use RDMA and GPU-to-GPU direct communication

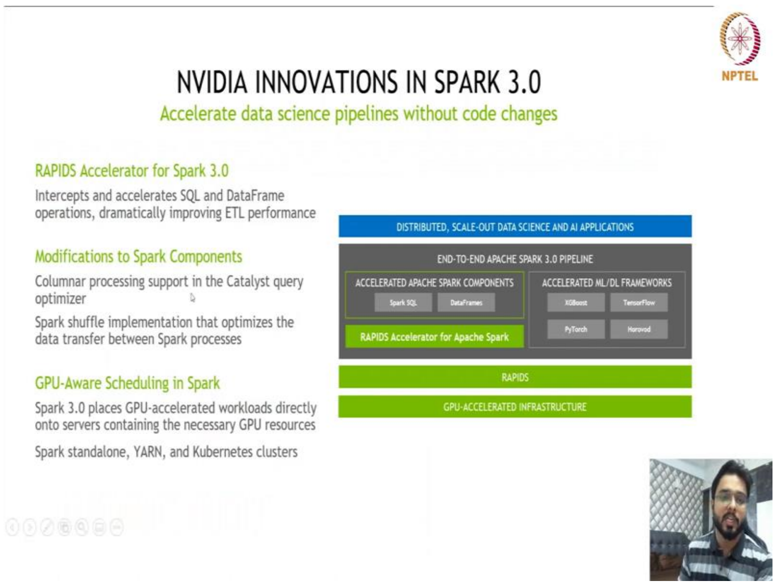
So, here if you see there are apart from the SQL API and data frame API we do also have custom implementation of Spark shuffle which is optimized to use something called RDMA and GPU to GPU direct communication. That means, if the data during the processing stage suppose there is a stage where you need to aggregate the data.

So, in those aggregation steps, we need to move data from one node to another; because data is distributed across multiple nodes and we cannot use the data where it is lying. Suppose, if you need to calculate the average age. So, age of first 5 people is lying in node 1 and age of next 5 people is lying in node 2 and we need to calculate the average age. So, the intermediate average age of node 1 and node 2 needs to come somewhere to get again averaged out to get the final result right.

So, because of that the networking IO is also a factor; that if the networking is slow. That means, movement of data from one node to other is slow then it may become a challenge for processing in the GPUs because unless and until we have accelerated Spark shuffle the entire job can go slower. Hence, we have this a custom implementation where we directly move data from one GPU to another GPU data on the same node or a different node using something called UCX libraries.

So, that detail I will come through in the next set of slides. From the perspective of SQL we have the base RAPIDS library on top of on bottom of it called libcudf and then yes it uses the CUDA architecture at the end because that is what our main USB is that the GPUs provided by NVIDIA have the CUDA cores which are highly optimized for increased throughput and parallel processing. So, that is what CUDA is helping us to do.

(Refer Slide Time: 16:34)



NVIDIA INNOVATIONS IN SPARK 3.0

Accelerate data science pipelines without code changes

RAPIDS Accelerator for Spark 3.0

Intercepts and accelerates SQL and DataFrame operations, dramatically improving ETL performance

Modifications to Spark Components

Columnar processing support in the Catalyst query optimizer

Spark shuffle implementation that optimizes the data transfer between Spark processes

GPU-Aware Scheduling in Spark

Spark 3.0 places GPU-accelerated workloads directly onto servers containing the necessary GPU resources

Spark standalone, YARN, and Kubernetes clusters

NPTEL
DISTRIBUTED, SCALE-OUT DATA SCIENCE AND AI APPLICATIONS
END-TO-END APACHE SPARK 3.0 PIPELINE
ACCELERATED APACHE SPARK COMPONENTS
Spark SQL
DataFrames
RAPIDS Accelerator for Apache Spark
ACCELERATED ML/DL FRAMEWORKS
XGBoost
TensorFlow
PyTorch
Horovod
RAPIDS
GPU-ACCELERATED INFRASTRUCTURE

So, what are the NVIDIA innovations particularly on the Spark 3 what kind of open source contribution NVIDIA has done to make it faster to run on GPUs. So, these are the main three changes which has been done first is NVIDIA has created a jar or a java based I mean jar based plugin, which can be used to accelerate in by passing it as the additional jar to the Spark jobs.

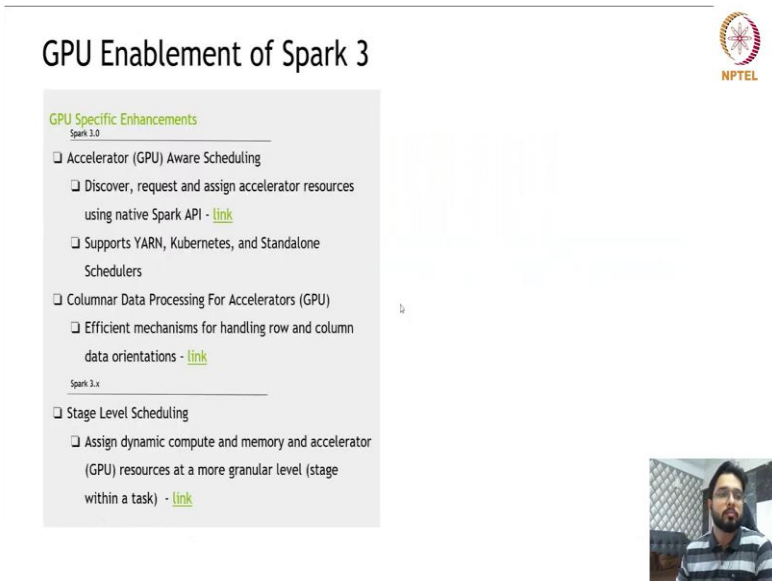
So, it intercepts and accelerates SQL and data frame operations, dramatically improving the ETL performance. Second is modification to the Spark components where we have columnar processing support in the catalyst query optimizer. Columnar processing that means that instead of processing row by row, suppose it has thousands and millions and billions of rows. So, it cannot process one row at a time or row by row right; it will be very slow and that will not even leverage the parallel processing of NVIDIA.

So, what we can do is instead of that passing the columnar kind of queries and columnar processing the entire column together in batches so that the entire chunk of the data goes together in the GPU system and gets processed together. So, that is how the modification has been done. And then as I said the Spark shuffle implementation which I already explained and finally the third is GPUs aware scheduling.

That means, if you have a cluster which has a combination of CPU nodes and GPU based nodes and that does not mean that only it will run on GPU nodes or it will fail on the CPU nodes. Wherever it has GPU where whatever possible it will run on GPU, whatever possible not possible on GPU it will fall back on the CPU. So, it is a combination it is the same job the same query can have multiple operations like select, filter, aggregate.

So, maybe filter sometimes may work may not work on GPU based on the operator support capability and if it supports on GPU will run on GPU if it is not supported, it will fall back on the CPU. Hence, your job is not killed or failed and at the same time you are also able to leverage your investments into the GPU.

(Refer Slide Time: 19:00)



GPU Enablement of Spark 3

GPU Specific Enhancements

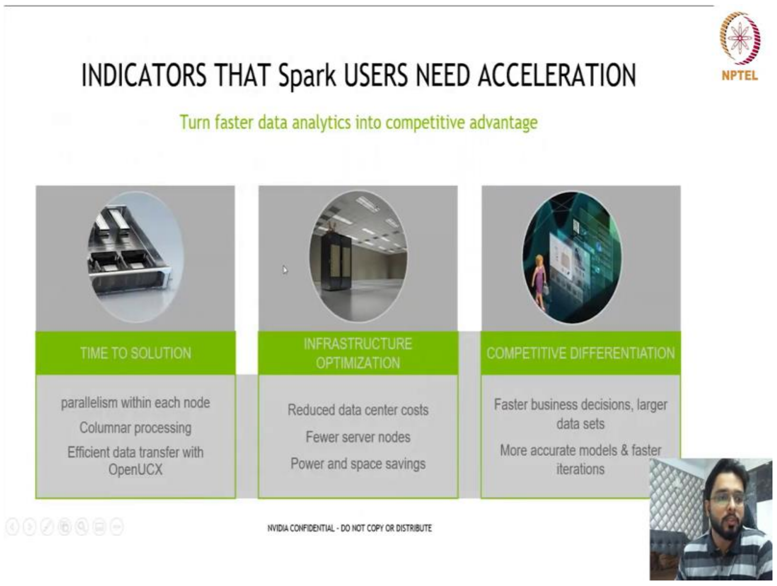
Spark 3.0

- Accelerator (GPU) Aware Scheduling
  - Discover, request and assign accelerator resources using native Spark API - link
  - Supports YARN, Kubernetes, and Standalone Schedulers
- Columnar Data Processing For Accelerators (GPU)
  - Efficient mechanisms for handling row and column data orientations - link

Spark 3.x

- Stage Level Scheduling
  - Assign dynamic compute and memory and accelerator (GPU) resources at a more granular level (stage within a task) - link

(Refer Slide Time: 19:04)



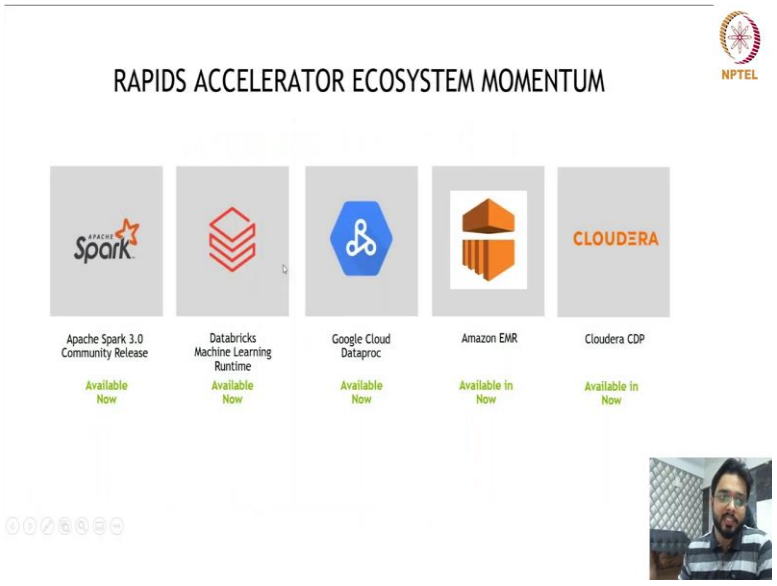
INDICATORS THAT Spark USERS NEED ACCELERATION

Turn faster data analytics into competitive advantage

TIME TO SOLUTION	INFRASTRUCTURE OPTIMIZATION	COMPETITIVE DIFFERENTIATION
parallelism within each node Columnar processing Efficient data transfer with OpenUCX	Reduced data center costs Fewer server nodes Power and space savings	Faster business decisions, larger data sets More accurate models & faster iterations

NVIDIA CONFIDENTIAL - DO NOT COPY OR DISTRIBUTE

(Refer Slide Time: 19:09)

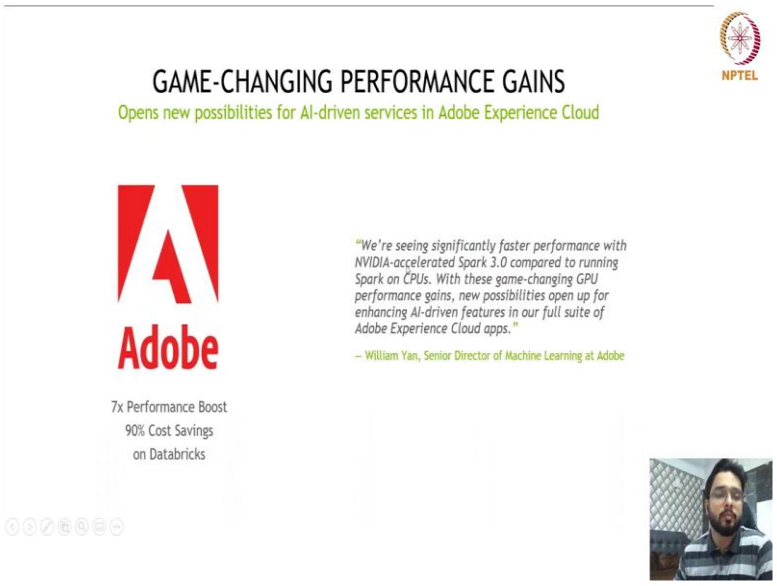


RAPIDS ACCELERATOR ECOSYSTEM MOMENTUM

Apache Spark 3.0 Community Release	Databricks Machine Learning Runtime	Google Cloud Dataproc	Amazon EMR	Cloudera CDP
Available Now	Available Now	Available Now	Available in Now	Available in Now

So, RAPIDS accelerator for Spark is available on all types of platforms now be it open source Spark, be it Databricks, be it Google Cloud, be it a Amazon EMR, be it Cloudera example of which I will show you today on the we can also do it in the Google colab environment.

(Refer Slide Time: 19:29)



GAME-CHANGING PERFORMANCE GAINS

Opens new possibilities for AI-driven services in Adobe Experience Cloud

Adobe

7x Performance Boost
90% Cost Savings
on Databricks

"We're seeing significantly faster performance with NVIDIA-accelerated Spark 3.0 compared to running Spark on CPUs. With these game-changing GPU performance gains, new possibilities open up for enhancing AI-driven features in our full suite of Adobe Experience Cloud apps."

— William Yan, Senior Director of Machine Learning at Adobe

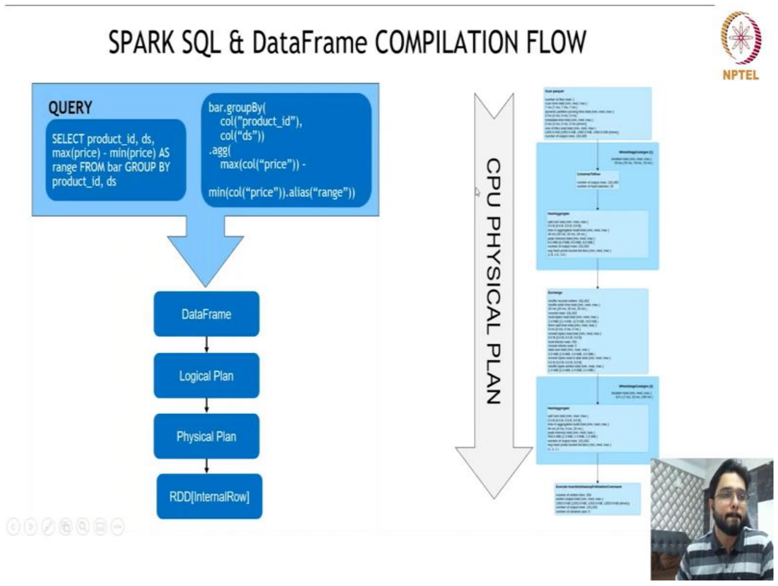
So, one of the customer story we have here is Adobe where they got 7 x performance boost and 90 percent cost savings on data bricks. So, they have given this testimonial to us that significantly faster performance with NVIDIA accelerate Spark 3 compared to the Spark 1 CPUs.

(Refer Slide Time: 19:55)

HOW IT WORKS

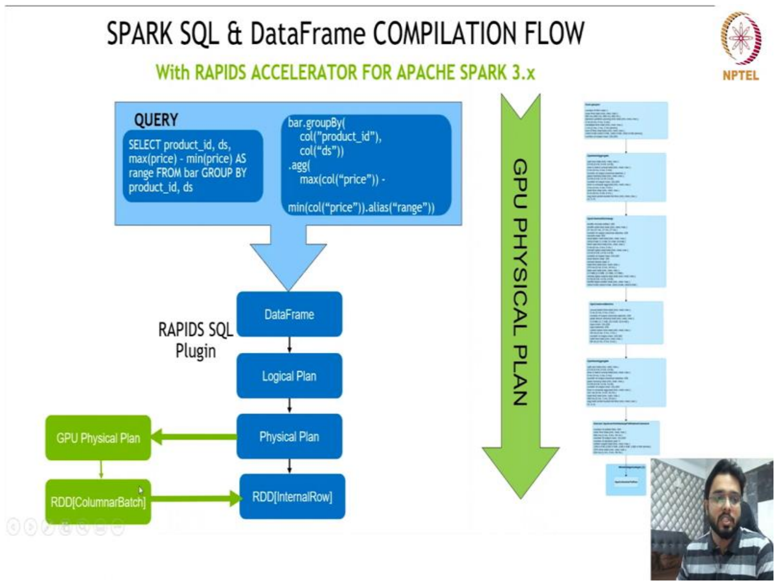
So, let us deep dive on exactly how it works how the CPU to GPU conversion works and why it is faster. So, the three things which I told, how exactly that will perform at the end.

(Refer Slide Time: 20:11)


SPARK SQL & DataFrame COMPILATION FLOW
NPTEL
QUERY
SELECT product_id, ds, max(price) - min(price) AS range FROM bar GROUP BY product_id, ds
bar.groupBy(
col("product_id"),
col("ds"))
.agg(
max(col("price")) -
min(col("price")).alias("range"))
DataFrame
Logical Plan
Physical Plan
RDD[InternalRow]
CPU PHYSICAL PLAN

So, here if you see Spark SQL how the CPU physical plan look like. So, any query even if you do not know about Spark just let me explain at a high level that any query whether you use traditional SQL platform or you use this modern Spark platform needs to go into a physical plan that how this query will get executed in the hardware ok. So, if you see the hardware execution, so first the query is data frame based then the logical plan is created then the physical plan is created then the RDD is created.

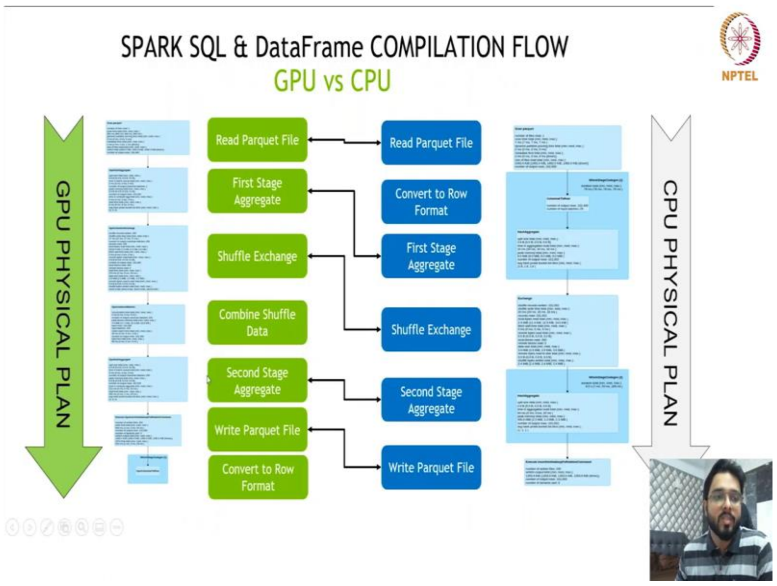
(Refer Slide Time: 20:47)


SPARK SQL & DataFrame COMPILATION FLOW
With RAPIDS ACCELERATOR FOR APACHE SPARK 3.x
NPTEL
QUERY
SELECT product_id, ds, max(price) - min(price) AS range FROM bar GROUP BY product_id, ds
bar.groupBy(
col("product_id"),
col("ds"))
.agg(
max(col("price")) -
min(col("price")).alias("range"))
RAPIDS SQL Plugin
DataFrame
Logical Plan
Physical Plan
GPU Physical Plan
RDD[ColumnarBatch]
RDD[InternalRow]
GPU PHYSICAL PLAN

So, that particular plan is now changed the CPU physical plan gets rewritten because of the jar which you add, because of the plug in which you add to the GPU physical plan. So, instead of running on say CPU 1, CPU 2, CPU 3 it will run on a GPU 1, GPU 2, GPU 3 and instead of creating row based RDD as I said, we will create RDD of columnar batches. That means, maybe one column at a time maybe that particular column 20000 values at a time.

So, it will entirely go together and will execute on one core of the GPU and similarly we since we have so many codes a lot of such columnar batches can go and execute in parallel.

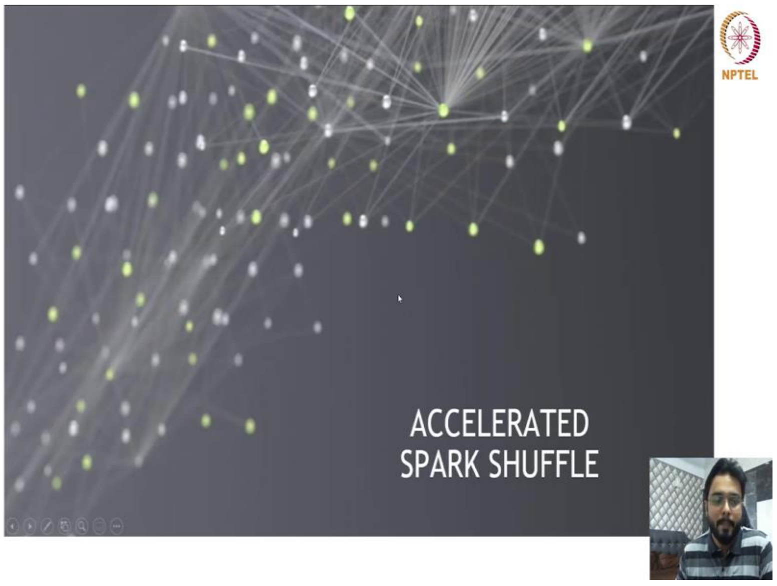
(Refer Slide Time: 21:34)



Deep diving into the physical plan now you may ask that ok, how the physical plan looks like. So, if you see the physical plan has a lot of steps if you see on the right hand side the CPU physical plan, it reads the parquet file converts it to row format and then does the stage aggregates, shuffle the data across the nodes because there is aggregate function here, then second aggregate and then finally, write it. But from the GPU perspective, it will read the parquet file it will directly do the stage aggregate.

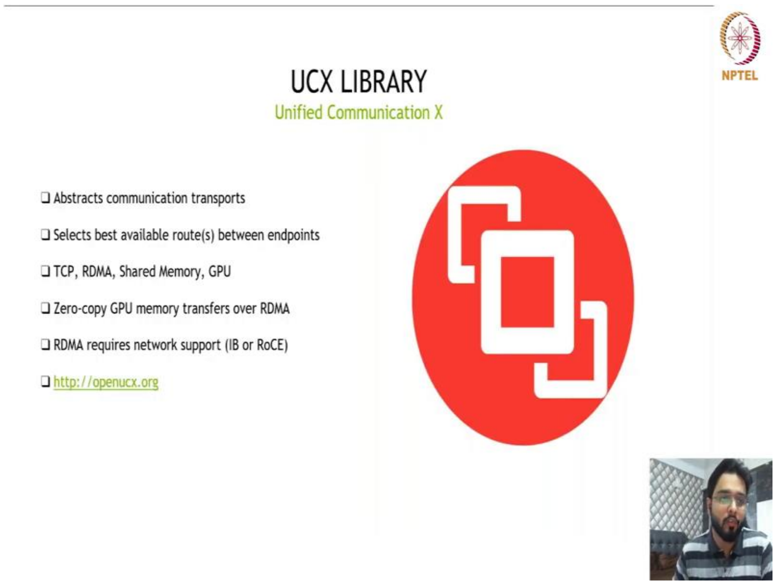
There is no need of converted conversion to the row format. And then we will have the shuffle exchange which will be again accelerated because of the RDMA capability which I was talking about and then it will directly go to the writing the parquet file without bothering about the row format again. Hence, we see that there are lot of steps which are not required like specifically working with the row format data and also come accelerating the processing and shuffle of the data across the nodes. Hence, it is more faster than the CPU plan.

(Refer Slide Time: 22:49)



ACCELERATED SPARK SHUFFLE

(Refer Slide Time: 22:55)



UCX LIBRARY

Unified Communication X

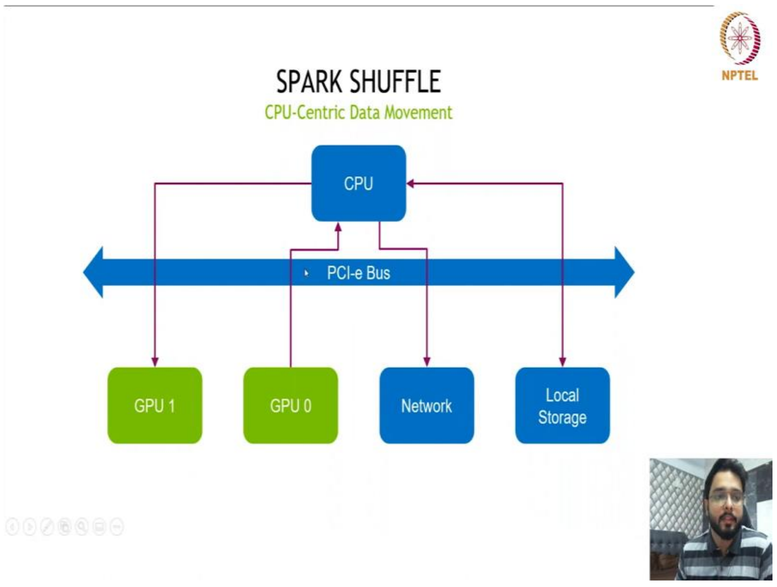
- ❑ Abstracts communication transports
- ❑ Selects best available route(s) between endpoints
- ❑ TCP, RDMA, Shared Memory, GPU
- ❑ Zero-copy GPU memory transfers over RDMA
- ❑ RDMA requires network support (IB or RoCE)
- ❑ http://openucx.org

Now, deep dive into this shuffle as I was explaining about the UCX library that what is UCX. So, the form of UCX is unified communication x library which is again like a kind of a plugin, if you deploy it in your nodes which has the GPU, it abstracts communication transports. So, what it does is suppose you have a specialized interconnects. So, the network fabric, the by default the network fabric used to connect from one node to another is Ethernet which will give you a maximum of 10 GBPS speed or something like that.

But there are some specialized interconnects nowadays being used which is something like InfiniBand or RoCE that is RDMA over conversed Ethernet which can provide up to 202 to 300 Gbps speed. So, what you unified communication next library does is, if it finds that this particular cluster has a hardware which can support higher bandwidth memory data transfer like 200, 300 Gbps.

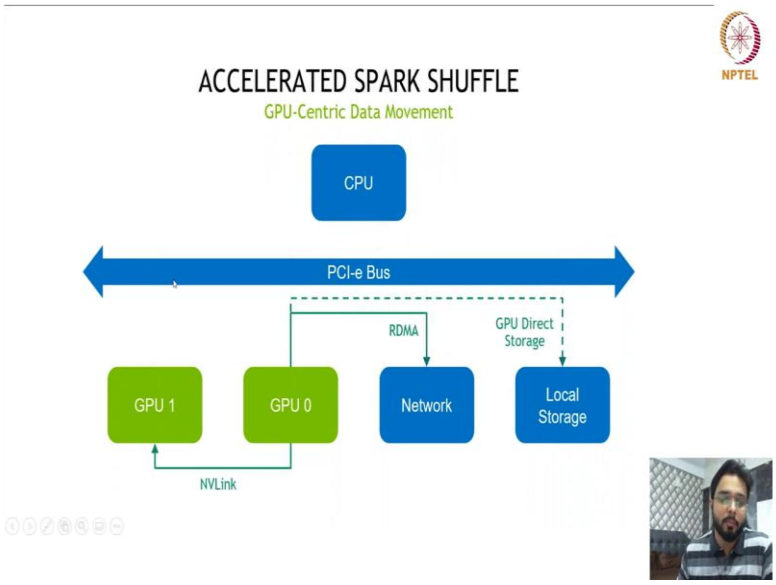
So, first it will identify that kind of hardware second is it will redirect the data transfer from a normal TCP based CPU to CPU Ethernet based 10 Gbps movement to a accelerated movement of data up to 200 to 300 Gbps.

(Refer Slide Time: 24:16)



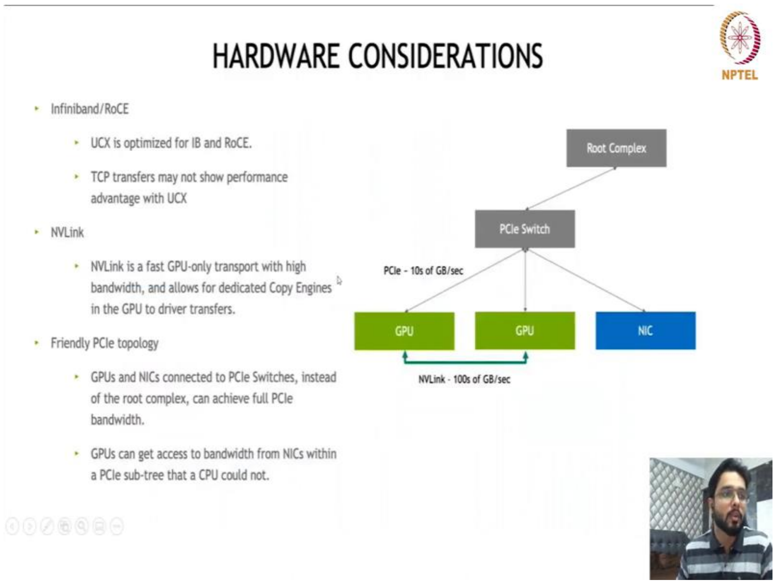
So, here if you see from the example from the hardware perspective the CPU centric data movement even if it has GPUs, if the actually UCX library is not added, then you will see that GPU 0, GPU 1 will have to transfer the data to the CPU first to go to the GPU from 1 GPU to other. Hence, there is the PCI-e bus is a bit slow if the data transfer is involving the PCI-e bus and the CPU in between as a hop then it may become again a bottleneck for the performance.

(Refer Slide Time: 24:59)



Hence, the GPU center data movement completely removes the CPU from the equation and it says that, whether you have a RDMA or Infiniband or a specialized interconnects like NVLink I will first prefer to do that instead of preferring to use CPU. Hence, it is very fast and dynamic in nature.

(Refer Slide Time: 25:25)


HARDWARE CONSIDERATIONS

- Infiniband/RoCE
  - UCX is optimized for IB and RoCE.
  - TCP transfers may not show performance advantage with UCX
- NVLink
  - NVLink is a fast GPU-only transport with high bandwidth, and allows for dedicated Copy Engines in the GPU to driver transfers.
- Friendly PCIe topology
  - GPUs and NICs connected to PCIe Switches, instead of the root complex, can achieve full PCIe bandwidth.
  - GPUs can get access to bandwidth from NICs within a PCIe sub-tree that a CPU could not.

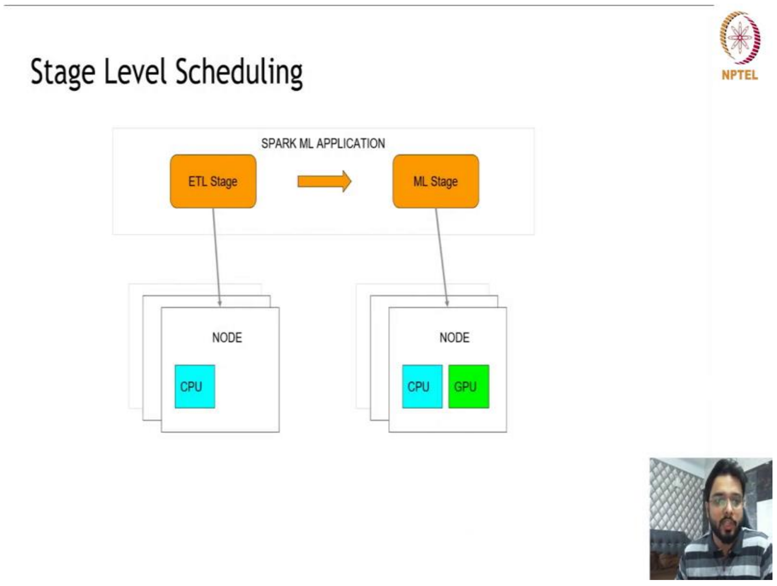
So, if you see like a brief description about hardware InfiniBand is UCX optimized TCP transfers may not show performance advantage, hence you can use that. NVLink is fastest GPU to GPU which is only present in our specialized systems like DGX and also we can do some friendly PCI topologies where GPUs and the network the interface cards connected to the PCI switches instead of the root complex can achieve a full PCI bandwidth.

So, that will also make sure that the even if the PCI is involved it is utilizing as fast as possible; because, at the end of the day you can have a scenario where the GPUs are not on the same node it is on the multiple nodes.

(Refer Slide Time: 26:16)

STAGE LEVEL SCHEDULING
& COLUMNAR PROCESSING

(Refer Slide Time: 26:20)



Stage Level Scheduling

So, the other feature we have added is stage level scheduling. So, you may know that ok when we run distributed computing jobs in Spark. So, it divides the data into chunks ok. We call those data we can call it as partitions, we can call it as blocks or whatever you can call it. So, those partitions of data pick up the parallel task. So, each data partition goes to a task.

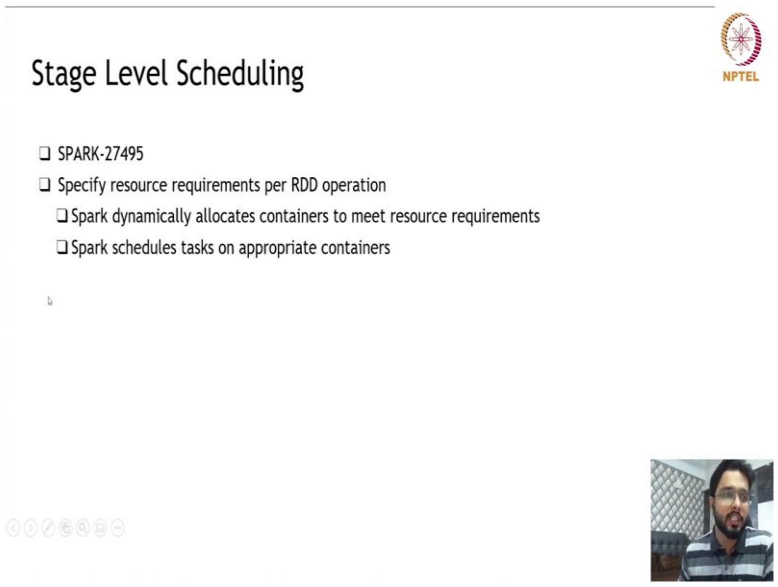
And if the task level execution completes for all the task level execution completes then only we can say that ok that particular job is complete; because, a job is a combination of many tasks and a data is like a data set is a combination of multiple partitions. So, we can run the each partition on each task and multiple tasks combined to make a job. Now, you may say that ok what will happen if some of the operators in the same job are supported and some are not supported.

So, their stage level scheduling will happen; that means, the stages where we will see inside the job that ok if we say so, there are multiple types of stages. So, first is narrow transformation and the other type of transformation is wide transformation. If there is a change in transformation from a narrow to wide; suppose you are filtering a data and then aggregating; that means, it will have two stages.

One is filter stage other is the aggregate stage. So, because of the stages being multiple, so if the stage one is not being supported on GPU then it will run on the CPU if the stage two is being supported on the GPU that will run on the GPU. Similarly, you can think from this diagram that suppose ETL stage is there and the ML stage is there. So, ETL stage I want to just run on CPU or it is not supported on the GPU, it will run on the CPU itself and the ml stage will run on the GPU.

So, thereby giving a full flexibility that ok you can have your GPUs to start with and you do not have to convert your entire 100 nodes or 200 nodes of cluster to a GPU based cluster to start with.

(Refer Slide Time: 28:36)

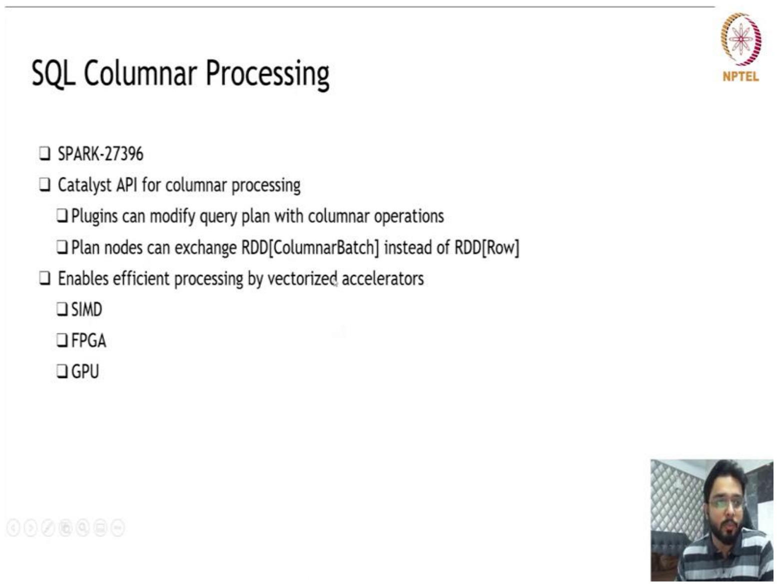


Stage Level Scheduling

- ❑ SPARK-27495
- ❑ Specify resource requirements per RDD operation
  - ❑ Spark dynamically allocates containers to meet resource requirements
  - ❑ Spark schedules tasks on appropriate containers

So, the Jira ticket is 27495 where we have contributed and we have added that ok specifies path resource requirements for operation, it dynamically allocates container to meet the resource requirements and it schedules task based on appropriate continuance which may run on GPU or run on CPU in parallel.

(Refer Slide Time: 29:05)

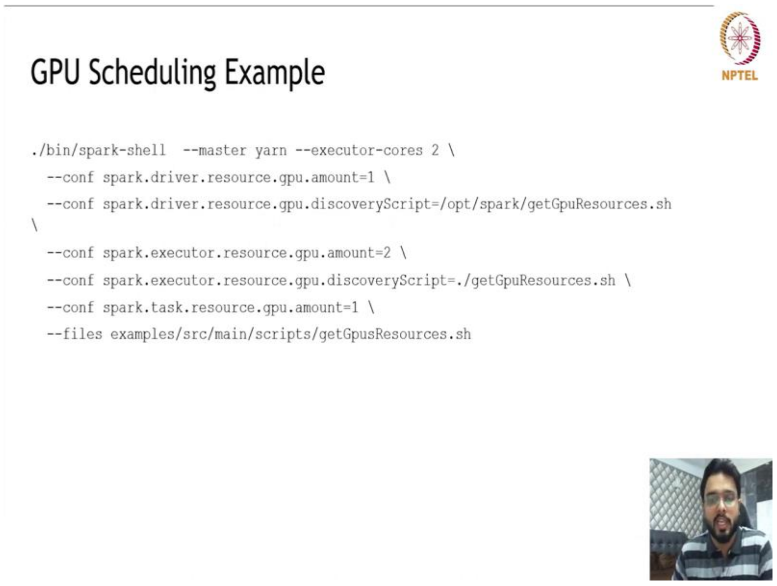


SQL Columnar Processing

- ❑ SPARK-27396
- ❑ Catalyst API for columnar processing
  - ❑ Plugins can modify query plan with columnar operations
  - ❑ Plan nodes can exchange RDD[ColumnarBatch] instead of RDD[Row]
- ❑ Enables efficient processing by vectorized accelerators
  - ❑ SIMD
  - ❑ FPGA
  - ❑ GPU

The columnar processing support is also added on the Jira ticket 27396 where we have added accelerators like GPU, FGPA, SIMD and so on and so forth.

(Refer Slide Time: 29:20)



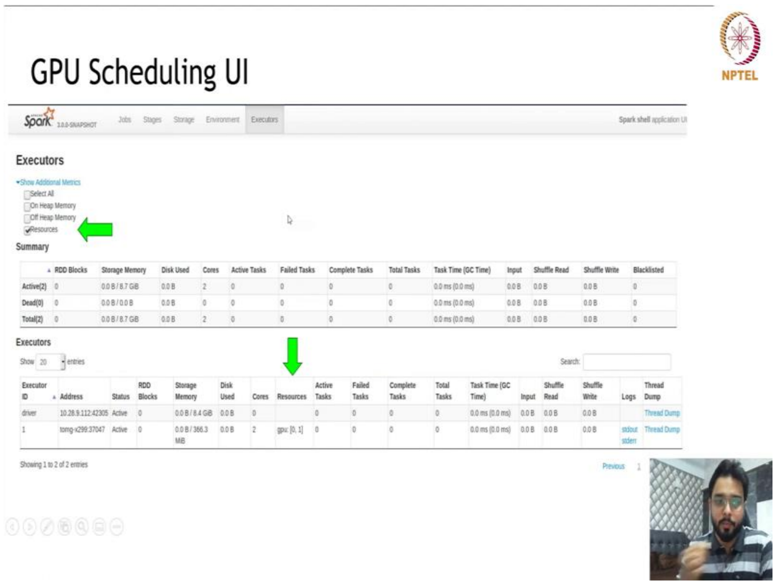
GPU Scheduling Example

```
./bin/spark-shell  --master yarn --executor-cores 2 \
  --conf spark.driver.resource.gpu.amount=1 \
  --conf spark.driver.resource.gpu.discoveryScript=/opt/spark/getGpuResources.sh
\
  --conf spark.executor.resource.gpu.amount=2 \
  --conf spark.executor.resource.gpu.discoveryScript=./getGpuResources.sh \
  --conf spark.task.resource.gpu.amount=1 \
  --files examples/src/main/scripts/getGpusResources.sh
```

So, how you do the scheduling? So, how it will identify that which nodes has GPU which node does not have GPU? So, where you submit; so, this is a shell command to submit a Spark job to do the data processing.

So, here if you see there is a additional parameter that ok the number of GPUs 1 the number of resources for GPU 2. So, this is the discovery script. So, this discovery script will make sure that to find out that in that particular node we do have GPU or not if it has GPU, it will use the GPU, otherwise it will not use the GPUs at all it will run on the CPUs.

(Refer Slide Time: 30:01)



GPU Scheduling UI

Spark 3.0.0-SNAPSHOT | Jobs | Stages | Storage | Environment | Executors | Spark shell application UI

Executors

Show Additional Metrics
- [ ] Select All
- [ ] On Heap Memory
- [ ] Off Heap Memory
- [✓] Resources

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(2)	0	0.0 B / 8.7 GiB	0.0 B	2	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(2)	0	0.0 B / 8.7 GiB	0.0 B	2	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Resources	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	10.28.9.112:42305	Active	0	0.0 B / 8.4 GiB	0.0 B	0		0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	tomg-x299:37047	Active	0	0.0 B / 366.3 MiB	0.0 B	2	gpu: [0, 1]	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump

Showing 1 to 2 of 2 entries

Previous 1

If you want to see on the UI from the GPU perspective, so that ok, if the GPUs are being used or not, so you can click on the executor tab of the Spark environment and click on the resources check box and there you will find that ok, GPU 0 1; that means, GPU is being utilized as of now.