**Apache Spark Fundamentals**

## Day 2 Lecture

*High-Performance Spark - Dataframe, Dataset, UDFs, and Caching*

**Transcript:**

0:00
[Music] in this Advanced spark course we're
0:07
going to be covering all the nitty-gritty details of spark like when should you use spark server versus spark notebooks when should you use pie Spark
0:14
versus Scala spark what happens when you use udfs is a py spark UDF worse than a scholar spark UDF we're going to go into
0:22
great detail covering all of that and then later on you know spark also has a ton of apis that we need to cover the
0:28
dataframe API the spark SE API the rdd API and the data set API and we're going to go into great detail about each one
0:35
and when you should pick which API depending on what job you're working on by the end of this 2hour course you will
0:42
feel a lot more empowered to know when to use which thing in spark because there's so many things to pick from here
0:49
and if you're more interested in learning how to use this stuff in the cloud make sure to check out the data expert.i Academy you can get 20% off in
0:56
the description below I'm really excited for you to check out this course so spark server versus spark notebooks I
1:02
just want to talk a little bit about this cuz uh this is a very different way that uh companies uh use spark so the
1:10
spark server is where you have to submit your spark job via like the CLI I don't
1:16
know if yall have used like spark-submit and then like you pass it like a a Java
1:22

class and then that Java class has like a run method or something like that and then it runs runs via that and usually
1:29
what happens there is you have to like upload a jar and then that jar gets ran and that's that's literally how Airbnb
1:36
does it it's a little bit slower than the notebook way of doing things sometimes it's sometimes slower and
1:42
sometimes better uh I I'll explain but like so with the notebook you just have
1:47
like one spark session that stays live and then it like you have to terminate it later uh whereas with the spark
1:54
server like the the session will run when you submit the job and it will just run until it completes and then it's
2:00
done and then once the job completes then the session is killed and it just
2:05
starts and ends kind of on its own whereas like with the notebook you have to kind of start and stop it yourself so
2:11
that's a big difference um I uh have found spark server stuff to be kind of
2:16
annoying in some regards because like when you upload that jar it uh can it can be a beefy jar especially if you
2:23
have a lot of dependencies and it can take you know a couple minutes to upload the jar so it takes a little while to
2:30
submit your job on the notebook side though I also don't like the notebook side because the notebook side doesn't
2:35
like a lot of times if you're running it a bunch of times and if you're using caching or anything like that then the
2:42
notebook is not going to mirror what production is going to do so that's why I generally speaking I like to use spark
2:49
Ser the spark server and Spark submit because then I know that when I submit my job via Dev however it's going to run
2:57

it's going to run like that in production as well so that's a kind of a big difference and it's one of the things I like about spark server but uh
3:05
they both have their pros and cons though okay so um in data bricks uh just
3:12
a thing to consider here um and uh This was um like going to be uh something I'm
3:18
going to be adding in when I actually break the deal with them but so data bricks and in our code today is we have
3:27
uh this notebook code that's going to be we're going to be playing around with that's going to fiddle with data and do all sorts of like combinations and all
3:33
sorts of stuff like that and generally speaking uh you don't want to deploy that code like with data bricks it's
3:39
wild because like you can have a notebook that's just sitting there and then you can edit the notebook and then that code is just running in production
3:45
immediately and it's crazy it's crazy that's like like when I see that I'm like that that that's dangerous because
3:51
the reason why that's dangerous is cu someone could just go in and Fiddle a fiddle with a column or a filter condition or whatever and now it's like
3:57
now you have bad data in production like can mediately it just immediately picked up and so one of the things that I
4:05
caution people about when using notebooks and it's one of the things like when I worked at Netflix because
4:11
Netflix actually pioneered this concept of being able to use notebooks and schedule notebooks in production and I
4:17
was always like dude that's like it's not it's not good it's not good because
4:23
the problem is is like notebooks can just be changed like it's not like they're checked in like they're not like
4:29
uh man by G right and if if you don't have like a cicd process for your notebook mutations
4:36
like no no no not for me not for me that's like a I would consider that bad

4:41
engineering and so uh that's something just to consider when you are kind of working in notebooks and working with
4:47
spark in general is how are these changes where are the checks and
4:53
balances in these changes and like how am I preventing bad data from entering production if I make a bad change
4:59
because is you're going to make a bad change I've made many I've made hundreds of bad changes throughout my career I've
5:05
broken pipelines in production I have made so many things fail like I'm not saying that it's never going to happen
5:11
but like you should have processes in place to minimize it so that like every time that you have a change that you are
5:18
confident in and then like cicd will put you in your place when it it's actually not a change that is uh ready to go so
5:26
just a big thing to think about when you are working in notebooks and kind of a kind of a gotcha that I would
5:33
recommend okay so
5:38
um uh this is uh remember this this this presentation is like really all over the place it's like the miscellaneous grab
5:44
bag odds and ends presentation so in spark when you have a data frame you can create this thing called a temporary
5:50
view which is a lot like a CT I almost consider a temporary View and a CTE to
5:56
be the same thing they're very very similar so you can create a temporary view but the problem with temporary
6:01
views is like if you use them multiple times Downstream they're going to recompute every time unless you cash
6:09
them so you have to cash them and then if you cash them then they won't get like if you query it twice it won't run

6:16
like cuz say you have a temporary view that's like an aggregation like you could imagine like okay if you have the
6:22
aggregation it should just run that aggregation once and then pass the result set around to the downstreams but
6:29
that's actually not how spark generally works the only way that spark will do that is if you call docash on uh the
6:37
temporary view then you can reuse it and it will not recompute it over and over and over again so that's a very
6:44
important thing to remember when you were working with temporary views in spark uh I noticed that when I was
6:50
working in unit economics at Airbnb where like it was crazy because I was using this temporary View and then it was being called Downstream like seven
6:57
times and I was like why is this job so slow and then I put cach in there and it went from the job went from taking like
7:04
3 hours to taking like 10 minutes and I was like wow like pers persistence and
7:11
like uh really actually caching result sets makes a very big difference in spark so you want to be careful about
7:17
stuff like that so caching right uh what
7:23
should what are the so caching can you can cash in a bunch of different ways right where imagine if you have
7:30
a very big data set then that's not going to fit in memory then you might
7:35
want to cach that to what's called dis dis I would consider dis uh this Dison
7:43
um option as I don't know if y'all have ever worked with a materialized view uh
7:48
materialized view and caching to disk are very very similar in terms of performance so you can write stuff and
7:55

then you have to but then you have to read it out of the materialized view so those are very very similar um generally speaking uh my rule of
8:02
thumb for uh caching is it should mostly you shouldn't use dis at all like it
8:09
should fit mostly in memory that's like the kind of the rule here obviously there are some small edge cases where
8:15
you have a big result set that you're going to reuse a bunch of times and but a lot of times in those cases like if
8:22
that's the case most of the time what you want is a staging table because
8:29
when you cash something to disk you're writing it out you're caching something
8:35
to dis and writing it out as a table are the same thing they're the same thing
8:42
and so like why are you doing it that way why are you caching it to dis so that that um all that data that you
8:49
wrote out to a table at the end when spark when the spark job terminates that data just disappears like why are you
8:56
doing that like you shouldn't do that like if you are using a lot of disk caching in spark you should be
9:03
questioning that and you should be questioning like is there actually a staging table here a table that I should have that is and then I can break up my
9:10
job into um a couple more steps and then that way you can have uh way faster uh
9:16
steps like each step will be faster and then like you won't have to do any dis caching and then back fills can be
9:22
better too because then you can just hold on to that staging data and then you only have to run the second step and
9:28
you don't have to keep Rec Computing and throwing away all this data that you wrote to disk I've seen that happen a
9:33
lot like that's actually like a like like almost every time like I feel like every time I've seen like dis only um uh

9:42
caching uh the right thing to do was actually to write it out to an actual like table with like an actual schema
9:49
not just caching the data frame to dis like caching the data frame to dis never the right
9:55
answer so uh back to what I was saying say caching uh is really only good if it
10:02
fits into memory I think that's important to remember that like uh caching uh if you were writing to
10:10
disk not the right way to go memory caching is really good because it can be really fast and you're not writing stuff
10:16
out but uh and then in notebooks this is another thing to remember is that like
10:21
if you call docash in notebooks make sure that you call UNP persist to uh
10:28
uncache them so you can test how the pipeline runs uh the first time because
10:34
if you cach it and then you rerun the code that data frame is already cached so a spark will already just like
10:40
essentially skip the computation so the second time you run the pipeline it will be a lot faster and but that's not how
10:47
it's going to run in production because you already have it cached in your spark server which is why I don't like it's a
10:53
great example of one of the reasons why I don't like using notebooks for development because you have these nice Edge Edge cases where uh you feel like
11:01
you made a great optimization to the pipeline but really you just forgot to call do UNP persist so it's a thing to
11:08
remember and we're going to go over that in quite a bit of detail in the lab today um just like on like how caching
11:14
Works um so that's kind of the kind of perspective with cashing good temporary
11:20
views um okay uh here's a good one
11:25

uh what's the difference between caching and broadcast Cas join I think that that's uh something that is very
11:33
interesting uh one of the things about caching is caching will if you cach a
11:39
data set it will stay partitioned and that is uh and you can save a result set
11:45
that way it's beautiful um so it will still pass only a fraction of the data
11:51
like when it when a join happens or whatever it's still going to be in the kind of partitioned up data set uh but
11:58
broadcast joins are not that way broadcast joins have to ship the whole data set it's not partitioned anymore
12:04
you have to you get one partition with with broadcast join so uh the entire
12:09
data set needs to generally be pretty small uh you know the rule here is a
12:15
couple gigabytes is probably going to be the most that you're going to work with like if you have more than that it's like it's going to be hard to use
12:21
broadcast join broadcast join has a lot of uh really powerful benefits though and we'll be talking about those in a
12:26
second but one of the things I wanted to really clarify here and this is something that I remember when I was
12:32
making this lab I actually didn't know the difference between these two things because they actually seem very similar
12:37
because a broadcast join ends up putting your entire data set into memory and shipping it that's what broadcast join
12:44
does and caching also puts stuff in memory so it's like how are they different and so this is essentially how
12:50
it works because caching will um if you have a say you have a 100 gigabytes that you're trying to trying to cach and then
12:57
it's uh but it's in 200 partition then you can cash it because then each
13:02

executor is only going to get like two gigs cuz they'll have the four tasks and each task gets 500 megabytes so that's
13:10
how you get like the two gigs for each exeutive and so you can actually cash it that way and it's not going to like
13:16
overwhelm your memory even and even though like when you think about the when you initially think about the 100 gigs you're like no way can I cash 100
13:23
gigs because that's not going to um that's going to be too much but that's kind of a perspective that you could think of about when you're kind of
13:30
building out uh when to leverage memory and when to use different types of joins
13:36
in spark and stuff like that highly recommend checking that stuff
13:41
out okay so um broadcast join this is one of the
13:50
most important optimizations in spark and for the most part it gets triggered automatically when it when it can uh how
13:58
it works is is if you have one side of the join that is um uh small uh usually
14:05
like the default here and and what like small is uh like kind of a rule of thumb
14:12
it's not like a a hard and fast rule but there's a a a setting here spark. SQL doob broadcast join threshold this
14:20
setting is the default value here is 10 megabytes so small the default
14:25
definition of small is 10 megabytes um but you can crank this up
14:34
I've cranked this up to you know way more than that you can even 200x this
14:39
threshold and Spark is still going to let you run as long as you have enough memory in your executors you can do like
14:45
uh like it can go up to it can go up to the order of gigabytes uh that you can broadcast uh definitely not terabytes
14:50
and definitely not tens of gigabytes tens of gigabytes is going to be too much and terabytes is going to be way too much but uh you can brast like sing

14:58
single digit digit gigabytes um but the default is 10 Megs so like if uh if you're trying to broadcast more than
15:04
that you need to update this threshold to be like hey my my uh my little data
15:11
set is not so little it's actually like 100 Megs instead of 10 Megs and but I still want I don't want to shuffle
15:17
because how it works right the broadcast join is the the little data set doesn't Shuffle around it just always the entire
15:25
little data set gets shipped to every executor because it's so it's small enough to do that and you don't have you
15:31
don't have to ship uh you don't have to split it up and the splitting it up is the part that is expensive that shuffle
15:37
Shuffle step so another thing you can do like if you don't want to set this
15:42
threshold is you can just use broadcast data frame you can broadcast the
15:47
whatever data frame that you're using you can wrap it explicitly with this broadcast function and then that will um
15:54
that will trigger the broadcast join regardless of the size of the data frame so so that is another way to do it I
16:00
kind of like this way to way of doing it better because then like you don't like if you don't want to have to like if
16:06
this if the if the little table gets a little bit bigger and then it goes above this threshold then you have to go and
16:12
update your code because you're like H such a pain right well there's two ways of go going about doing that right is
16:18
one you just set this threshold really really high and that could be the other way to do it so that um the that you
16:25
don't really ever have to worry about the small data set getting too big um I personally like the more explicit way of

16:33
doing these joins where you do wrap it in the broadcast because then uh data
16:38
Engineers down the line are going to be like they understand your intent a lot better than uh than if you just set this
16:47
to a high value they don't they they won't know what the thing there is even though if you set this to a high value
16:52
it's technically like less code but I I I like I like to be more explicit than
16:58
implicit it whenever I am kind of working on the functionality of different things in my uh in my code
17:03
base so definitely uh look into broadcast joins they save my life uh in
17:09
many many cases when I've been uh working in Big Data um okay so udfs udfs are an
17:17
interesting one uh we're going to talk quite a bit about udfs um so one of the things about udfs
17:25
is they so UDF stands for userdefined function so it's like essentially uh a
17:31
way to drop into code while you are um in data frame world like you can go and
17:36
drop into code and Define a column or whatever so um one of the things that's
17:43
nice about udfs is they uh allow you to do all sorts of complex logic and
17:49
processing and so that can be a very powerful thing that you're going to want to look at um but there is some gotas
17:56
here um they the the some of these gotas
18:01
are more historical than they are um kind of like now uh like like but I'm I
18:10
want to talk about them a little bit so one of the problems with python udfs right so say you're defining a UDF in
18:16
Python then uh What spark and Scala has to do is really crazy so what it does is
18:24
it has to um it has all its data it's running in Scala it's running in the jvm and then when it when it hits a python

18:30
UDF then what it does is it it it serializes that data and then passes it
18:36
to a python process python will run your 10 lines of code and then python will uh
18:42
have some sort of uh value that it's going to return back so then it will serialize it back and send it back to
18:49
Scala and then Scala will have your result whereas like so you see how it's like okay you're in Scala you serialize
18:55
to python python then deserializes the data python then runs its python code
19:01
and then it goes uh and then it takes that data and then it serializes it again and then it goes back to spark
19:07
where Spark can then deserialize it again so um uh I don't know if y'all know but
19:15
that's a lot of steps it's just a lot of steps so in some regards that can make Python and py Spark udfs a lot less
19:22
performant in spark um with that being said um Apache arrow is something that
19:28
has been helped make uh the python udfs and this nasty like many many um
19:35
serialization steps uh a lot better so it they have become mostly in line with
19:43
scholis spark udfs like they're mostly the same I've noticed that they still have a little bit of a per performance
19:49
hit when you aren't talking about udfs but you're talking about udfs like userdefined aggregating functions and
19:57
that's going to be you still have kind of you still take a performance hit in uh python when you're defining a
20:03
UDF uh versus like a UDF so you want to be a little bit careful there and that's
20:09
going to be one of like when people ask like should you use pie spark or should you use scalas spark um that's about it
20:17

like there's two things there's two things that schol spark gives you and that's going to be one is it gives you
20:22
more performant udfs and the thing is is like most people when they write their spark code
20:29
they don't use udfs it's a very like Niche very like I don't know like one in a 100 pipelines use udfs like it's very
20:37
rare um and so um if that's the case like what is the um other option here
20:45
right and that's going to be the other big thing is the data set API the data set API is the other big thing that you
20:51
get in Scala that you don't get in Python and those are going to be the two big things that make Scola spark a
20:58
little a little bit better but obviously scholar spark isn't free because it comes with the cost of like learning
21:05
schola which I think a lot of y'all like when I'm talking with y'all about that you're like I don't know about that I
21:12
don't know if I want to freaking learn Scala that sounds like a lot of work and uh I feel lucky for my own sake because
21:19
I uh um my very first programming language was Java and Java and Scala are both jvm
21:28
languages so they actually both compile to the same um bite code Scala compiles
21:33
to Java B code and Java compiles to Java B code so like if you already know Java
21:39
picking up Scala is very very easy um but python to Scala is not quite as easy
21:45
because like python is what's called an an interpreted language whereas Scala is a compiled language and just that one
21:53
difference is if you've never worked in compiled languages Scala is going to be a annoying to learn it's going to be
22:00
super annoying to learn so in those cases like like I've got I get this question all the time like should I
22:06

learn Scala should I learn python or py Spark versus Scala and like generally speaking I would say you want to learn
22:12
pypar py spark is going to be uh is going to serve you better for the most part it's going to be a lot a a much
22:18
larger percent of the jobs are going to be using py spark um because the performance considerations that I'm
22:24
talking about in terms of like udfs and native functionality and the
22:30
data set API a lot of that stuff you um don't even really need uh like if you're
22:37
not working at massive massive massive scale then like you don't even really even need to consider it because it's
22:44
like it's it's a negligible benefit if you're at medium to high scale you you
22:49
only start to feel the pain a little bit when you're at like extreme scale so uh
22:56
that's kind of my perspective on udfs and P Spark versus scholas spark uh so the data set I just want to talk a
23:02
little bit about the data set API because the data set API actually allows you to do completely just functional
23:09
programming uh with your python or with your scalas spark uh code where you
23:14
don't have to do any data frame or any declarative programming you can just do pure functional uh data engineering uh
23:21
we're going to go over some of that today uh there's a bug in the lab that I like was troubleshooting today and I thought I was going to figure it out but
23:27
there was like a it's the code runs in data bricks but it's not working for some reason um in the example Docker
23:34
image I have and uh but we we'll figure it out okay so this is I think the other
23:39
big debate in spark uh is around do you use data Frame data set or spark SQL
23:47

those are essentially the three ways that you can kind of transform data um so let's kind of go over each one here
23:56
uh I kind of think of these things on a continuum where you have on one side of the Continuum you have spark SQL and on
24:02
the other side you have data set needed to update the slide here but okay so spark SQL is going to be um useful for
24:11
when you have many cooks in the kitchen if you have a data scientists who want to collaborate because spark SQL is the
24:17
lowest barrier to entry because you don't need it you only need to know SQL and then you can like write spark code
24:24
and it's really good in that way and uh and so you get a lot of benefits from that um and it's nice and if you're
24:31
looking to change stuff quickly spark SQL is definitely the way to go um then
24:37
kind of in the middle you have data Frame data frame is nice because you can
24:42
modularize the code and you can kind of uh put put stuff into separate functions
24:48
and make things more testable and all that kind of stuff I like the data frame API for that um and then the data set
24:56
API is uh the furthest towards like almost like software engineering because
25:01
the data set API gives you all of that and schema and then it allows you to create allows you to create fake mock
25:08
data a lot more easily and that's another thing that you can do like I know if y'all remember like um uh in
25:15
week three when we used like those named tupal and you had to like create a new named tupal but like if you're using the
25:23
data set API you don't have to you don't have to create any named tupal because those
25:28
tups already exist because those tup are an integral part of the pipeline so your
25:34

your uh I'm try to say here your uh unit and integration tests can just use the
25:40
the schema that is part of the pipeline to generate the fake data so that and
25:46
then you don't have to create a separate tupal to generate the fake data you can do it with the the one in the data set
25:52
API so it's a you have a tighter integration there and data set API also has a nicer way of handling null and um
26:00
it handles null better and nullability and all that stuff uh because SQL and
26:06
data frame uh when you encounter null it just returns null and that's all it does
26:13
whereas in the data set API you have to model it right you have to say whether or not a column is nullable or not and
26:21
if you don't um if you don't explicitly say whether the column is nullable or not then you're going to um
26:29
uh it's not going to work right it's like uh because if you say everything is not null then like if there is one of
26:35
the columns is null the pipeline will throw and you'll get a null pointer ex exception and the pipeline will just
26:41
fail and uh so and it fails very loudly so that could be uh it could be a great way to enforce uh quality control inside
26:50
the pipeline as well so that you uh like you know for sure if the pipeline ran
26:55
that means that there's no nulles in this column because the pipeline wouldn't run if there was and so that's
27:01
something that spark SQL and data frame don't give you and I like that in the data set API because then uh you know
27:08
for sure obviously that can be kind of annoying uh if like there are a lot of nles but then you can model that in the
27:14
data set API as well that things are nullable and then in those cases then you can just manage null more

27:20
effectively so that's the way I think about it uh I want to just go over this a little bit more here so if you are
27:27
building a pipeline and you're iterating on it quickly with data scientists use spark SQL if you are building a pipeline
27:33
for the Long Haul and you're using p spark use data frame if you're building a pipeline for the Long Haul and you're
27:39
using scalas spark use the data set API and that's going to be uh kind of my way
27:45
of thinking about which API to pick when I'm building my data pipelines so um
27:51
that decision is an interesting one it's uh like obviously you can go any way you want with that but that's kind of my
27:57
perspective Ive so we talked a little bit about this um in the last class I wanted to just
28:03
cover it one more time because it's something that is super important that I've found that is just so good for
28:09
efficiencies so uh Iceberg uses Park as the default file format and then um run
28:17
lengthen coding we talked about that a lot we talked about it in week one and in week six I like to talk about it at
28:23
the beginning and at the end so that y'all can really remember it both from a recency and a Primacy bias um one of the
28:30
things that runlength en coding really leverages a lot is if you sort things effectively so if you remember in the
28:37
lab from uh from Tuesday uh the Sorting showed that uh you have um less data the
28:45
the volume of the data is less because of that and so that is a great thing to recognize when you are uh building your
28:52
data sets is that you made a data set smaller and it's taking up less Cloud space and it's more efficient to query
28:58
because you compressed it better and so that is a big big thing um I want to

29:03
talk a little bit more about these two differences uh I don't think I covered it very well in the lab but um do sort
29:10
is very slow and it it it incurs an extra Shuffle step and that shuffle step
29:17
is actually really really painful because it's a shuffle step of with Partition one and then you have to like
29:23
have you have to pass all the data to one executor so that uh it can guarant that the data is in the right order and
29:30
so don't ever use sort like freaking ever like don't do it it's bad um s sort
29:37
within partitions is another call that you can do and sort within partitions is great because then it looks at the data
29:44
within the partition that you're writing and it will sort locally and it's very fast to sort there and then you don't
29:51
have to pay like a very expensive cost to get the benefits of run length en
29:57
coding you get it um uh uh for pretty much free or for very
30:03
cheap okay uh let's talk a little bit about spark tuning um I again this is something I I I I I want to H rehash a
30:10
lot of this stuff because these are just bad practices I've seen uh executor memory uh can go up to 16 gigs and uh
30:17
don't just set it to 16 gigs and be like that's enough we're good to go um because it's just waste it's wasteful
30:22
I've seen that happen so many times in big Tech because like it depends right CU I think a lot of people in big Tech
30:28
like a lot of the data engineers in big Tech are like they think because they're like well I I get paid so much money
30:34
that like if I ever have to troubleshoot this it's not worth my time to troubleshoot it so might as well make it a little bit more wasteful so that like
30:40

I guarantee that it's a little bit more resilient because then I don't have to troubleshoot it ever because my time is
30:45
very valuable um but like I don't know I I I don't like that mentality for the
30:51
most part I like to at least like think about it a little bit so that like you can uh like don't just waste a bunch of
30:58
memory in the cloud you know uh and then driver memory same idea don't like only if you're calling DF do collect or you
31:05
have a very complex job are you ever going to need to move driver memory so uh those are kind of uh the two memory
31:11
tunings uh here's another kind of these are going to be some other very important sets of tuning the uh we're
31:17
going to talk about Shuffle partitions the default here is 200 I always wondered why it's 200 200 is such a
31:23
random number but um uh how do you pick the right number of partitions I think
31:30
that's an interesting question right so uh generally speaking uh you want to
31:35
like um if your data is uniform which it won't be but if it's roughly uniform you
31:41
want between like about 100 to 200 Megs per partition that's roughly what you want and that's how many like files you
31:48
should write and that's going to give you like the optimum kind of throughput and so uh what number is that like and
31:55
then also uh you have the whole idea around like split files so this this rule kind of changes a little bit but if
32:02
so but you can do math to figure out what the optimum partitioning should be right because if it's like if your output data set is say your output data
32:09
set is 20 gigs then uh 20 gigs divided by 100 Megs right is going to be like
32:16
2,000 or something like that so that's roughly a way to think about it um
32:22

that's how you can uh that's a you can do some quick kind of back of the envelope math that way that's a good
32:27
place to start uh I'm not saying that's necessarily the right for each job because jobs can depend like some jobs
32:34
are going to be a little bit more IO heavy or network heavy or memory heavy and so in that case like you might need
32:41
more or less partitioning um and in those cases you can like maybe try like
32:46
so in that case if you have a 20 gig output data set and uh the rule here says okay use 2,000 maybe try like 1,000
32:54
to 3 we try 1,000 2,000 and 3,000 and then look at the performance of the job and then see like which one is going to
33:01
be better cuz uh like that will give you a good range of uh performance metrics
33:08
and then one of them will be usually a clear winner given you pick such big differences right so it's like 50% lower
33:15
and 50% higher is going to be uh kind of the range I'd pick um um aqe I love aqe
33:23
aqe saves saves so many lives and so much like suffering and pain with the
33:28
data Engineers that we have now um so aqe is going to be so good um so that is
33:37
adaptive query execution and uh it's really good for skewed data sets but you
33:42
don't just want to enable it on on the off chance that your data set is skewed you really only want to enable it once
33:48
you are sure that the DAT the data set is skewed and then spark will make good work of the skew for you and you don't
33:54
have to do anything else it's so good I love that I love so much like it's like cuz that was like such a painful process
34:01
uh for so many years as a data engineer was dealing with skewed data sets congrats on making it to the end of the
34:07

advanced spark lecture that's a lot right uh get ready for the lab and if you're taking this class for credit make

34:13

sure to switch to the other Tab and to the other link so you can get credit there I'm excited to check out this lab and get some more hands-on experience