Apache Spark Fundamentals

Day 1 Lecture

Spark + Iceberg - Memory Tuning, Joins, and Partitions

Transcript:

0:00
[Music] in this course you're going to learn all
0:07
the basics that you need to know about Apache spark like the architecture what does the driver do what does the
0:12
executive do how does it all come together why is it so powerful and when should you pick spark over something
0:19
else these Basics are going to be covered in the course and then later on we are going to go into a hands on lab
0:24
to just talk about partitioning and sorting and how to actually have spark read and write data you will get a lot
0:31
of Hands-On knowledge in this course make sure to download spark in the description below and if you're
0:36
interested in learning how to use spark in the cloud I highly recommend checking out the data expert academy uh you can
0:42
get 20% off in the discount below I'm excited to see you there okay so spark is a distributed compute framework that
0:49
allows you to process very large amounts of data efficiently uh some of youall probably knew that um it's kind of the
0:57
successor to other Technologies um if y'all followed any of like the
1:02
progression of Big Data Technologies uh really everything started back with like Hadoop and Java map reduce back in like
1:10
2009 and that's when uh those things were developed and then we learned and then they kind of transitioned to Hive
1:17

and then now spark and Spark is kind of winning and like it's going to it's
1:22
going to be wild to see how long spark actually sticks around because it looks like it has a longer kind of uh
1:27
technology curve than some of the other predecessors but that's essentially What spark is we're going to go way more deep
1:34
into the details of spark here but that's the idea so I just want to talk a little bit
1:39
about my history with spark so I learned Java map produce back in 2014 when I was
1:45
working at teradata and then 2015 to 2017 I used Hive a lot uh a lot at
1:51
Facebook a lot at teradata uh and then um I remember in 2017 I uh there was a big push at
1:59
Facebook too adopt Presto uh to switch Hive to presto but the problem with
2:04
Presto back then and still is kind of a problem with Presto now is Presto does
2:09
everything in memory and so if you have an operation that can't fit in memory
2:15
Presto just fails just fails you can't use it and so uh and I worked in
2:20
notifications at Facebook so I we couldn't use Presto but I also didn't want to use Hive cuz Hive was slow and
2:26
terrible so like I was one of the very first early adopters of spark at um
2:31
Facebook I think I was like the third or fourth person to like create a spark job at Facebook and um it was great it was a
2:38
slam dunk in so many different places uh in my in my usage both at Facebook and
2:44
further on in my career um and uh I've tried all the different apis for spark
2:51
right there's spark squl there's data Frame data set piie spark spark R like I
2:57
always joke with people about spark R cuz people for forget that you can actually write our code with spark and it's kind of crazy but anyways I
3:04
attribute learning this technology as one of the main reasons why I was wildly

3:09
successful in big Tech because I learned this framework very very thoroughly and
3:14
I know all the ins and outs of it and so it's been great I I've been using it since then like since 2017 to 2023 so
3:22
like it's many years like six six years of just using spark and it's just gotten better and better and better since then
3:27
and uh yeah let's uh let's let's keep on here so why is spark so good
3:34
um uh there's a couple things uh spark is really good because it leverages Ram
3:39
more efficiently and more effectively than previous iterations uh so for example like if you wanted to do a group
3:46
by computation in Hive or map produce um everything had to be written to dis and
3:52
then written to dis and read from disk and every single step had to be written and read from disk and it made it very
3:58
resilient like hi hien Java map reduce jobs were very resilient but they were
4:04
also incredibly painfully slow uh spark was able to kind of short circuit some
4:09
of that and minimize the amount of times that it needs to write to disk spark essentially only writes to disk when it
4:15
has an operation that it doesn't have enough memory for and then it will do this thing called spilling to dis
4:22
generally speaking though like uh you don't want spark to Spill the disc you want spark to use Ram as much as it
4:28
possibly can and that's going to be where you get very effective and fast computations with spark spilling to dis
4:34
makes spark it essentially has spark go back to being Hive and it's again being
4:40
kind of slow uh so another thing about spark is it's storage agnostic like you
4:45

can read whatever you want right whether it be a relational data relational database a data Lake a file uh mongod DB
4:54
whatever you want to read spark can read it easily um also spark has a massive massive massive community and you don't
5:01
need to use spark with data bricks you can use spark with whatever technology you want and it works great so um that's
5:08
pretty much what uh is going on with like why spark is so
5:13
good um when is spark not so good uh there's going to be um I think two cases
5:20
when I would say that you don't want to use spark uh the first one is um similar it's a very similar guidance to what I
5:27
said about Flink was if you're the only one on your team that knows spark then
5:34
uh like you probably shouldn't use spark you should or you should teach your team so that you can have better maintenance
5:39
and better uh reliability and you can distribute responsibility more effectively and also um your company
5:47
might already use something else like uh a lot of companies bet heavily on big query or they bet heavily on snowflake
5:53
or they bet heavily on another kind of technology that isn't sparked but also works and in those cases is like it's
6:00
better to have 20 big query pipelines than it is to have 19 big query pipelines and one spark pipeline like
6:07
it's just like a homogeneity of pipelines matters a lot I've used that word a couple times in some of these
6:13
presentations and that's when like you don't want to bring in spark in those cases you want to just use what the
6:19
company's betting on okay so how does spark work so I like
6:25

to use the analogy of spark is like a basketball team so the basketball team has uh three things in it um it has the
6:33
plan which is the play that the basketball team runs the driver the
6:38
driver is like the coach the coach of the basketball team and the executiv are the players of the basketball team so
6:44
we're going to be kind of using more analogies here I need to update these slides so they actually have like
6:49
basketball players in them but it make it'll make a lot more sense as we kind of go through this so remember we have
6:54
uh the play the coach and the players and those are going to be the three things that we have but they're called the plan the driver and the
7:02
executors okay so the plan you can do this as the play this is like uh the coach will tell the players what what
7:09
play to do and uh this is going to happen uh like where the play can be
7:16
described in whatever uh way you want to do it whether usually it's going to be
7:21
python Scala or SQL or R if you want to be a weirdo and then uh the plan is
7:27
evaluated lazily um what that means is the plan doesn't actually run
7:34
until uh I like to say like a player takes a shot which means like they
7:40
either um they either like try to write data out somewhere or they try to
7:45
collect data somewhere so that that's what write taking a shot means like writing output or collecting data and
7:53
the collect what that means is like essentially they are taking information from the play and bringing it back to
8:00
the coach to to have the coach tell them what to do next and that's going to be uh the main things around the plan of
8:07

how spark works so this is going to be like your data frame code this is going to be your spark SQL this is going to be
8:14
your data set API code and that's essentially what we're going to be working with here so that is kind of the
8:20
plan of how spark works okay so you have the driver uh which is like the coach
8:26
you can think of like the coach reads the plan so the coach like tries to figure out what plan to uh run and what
8:32
play to make and um spark only like so you uh the driver can have a couple
8:39
settings that it can have in it um but uh there and there's like probably 10
8:47
different driver settings for spark that you can set uh really there's only two
8:52
that you need to care about here um spark. driver. memory which is going to be the amount of memory that the driver
8:58
has to process the job um I think this defaults to like 2 GB it can go all the
9:04
way up to 16 um uh generally speaking the only times
9:09
that you need to bump it up is like if you have a crazy complicated job that has like so many steps and plans you
9:17
might need to bump it up in that case or if you do collect where like the data
9:22
that the job is processing changes the plan of the job which mostly is bad
9:28
practice you don't generally speaking want to do that there is some edge cases where you want to do that but like for the most part you don't want to do that
9:34
but those are going to be the only two cases when you really need to bump this setting up from two gigs to whatever
9:41
other number you want to bump it to um the only other one that you might want to also uh change is memory overhead so
9:50
this is where um the driver needs memory to process
9:56

the plan and if for some reason you have a very complicated plan you might need
10:04
more uh um overhead memory because the jvm might take up more memory in those
10:10
cases so you might want to crank this to a different number uh which is get so that your non Heap memory is the memory
10:17
that Java needs to run not like the memory for the plan or the memory for
10:22
the collect calls or the memory for anything like that you don't it's none of that it's just the memory that Java
10:27
needs to run and the the only time that that needs to be bumped up is if you have a very complex
10:33
job every other driver setting you shouldn't touch at least in my experience I've and I've spent many many
10:40
years you know I've been playing with playing Ral spark for seven years and I've played with all of these settings
10:45
just to see like played a lot of like what if and Science and experimentation games with a lot of these settings and
10:52
these are the only two that I've seen to have any sort of substantial impact on the performance of the job all right so
10:57
that's the driver so keeping in mind the driver determines a few things for you um so the driver
11:05
determines when the job should actually execute so that's like when it finds like an output or a collect call it will
11:14
find those um to tell the players to start running the play it will also
11:20
figure out how to join the data sets which can be very very important um because depending on what
11:26
type of join Sparks spark decides to use that can mean that your job is 10 times
11:32
um less performant or 10 times more performant and it also determine how much parallelism is needed in each step
11:39

so for example like if say you have two data sets you're reading in and then you join those two data sets and uh neither
11:46
data set is um is small you have two big data sets you're reading in so generally speaking the initial parallelism for
11:53
those steps is how many files those data sets are written to and then when you
11:58
join then you uh you get into the parallelism step which we're going to talk about here in a second and that is how much
12:05
parallelism happens after a join or something like that and so those are the three things that the driver really is
12:12
all about and uh if you can get the driver to do these things the right way
12:18
then you can have way more efficient spark jobs so talk a little bit about the
12:24
executiv uh executiv here are going to be um what actually do the work these
12:30
are like the Michael Jordans and the Kobe Bryants and the you know the act the LeBron James of the um of the spark
12:38
framework and they actually do the work they're the ones who actually take in the data and then transform the data and
12:45
filter and Aggregate and whatever other operations you want to do on the data so
12:51
in this case uh there's really only about three um settings here that you want to touch in terms of executive um
12:59
settings in this case you have um spark executor memory uh this is again has the same um
13:07
constraints as driver memory so it defaults like one or two gigs and then
13:12
it can go all the way up to 16 gigs um I've seen bad practice with this
13:20
actually where there's like if a job like o right
13:26
out of memory exception um one of the things that can happen here is people will update this to 16 gigs and then
13:33

just forget about it and they're like whatever like it's it's fine even though like it makes it so the job it runs way
13:39
more expensively um generally speaking uh the way I like to think of like how to set your executive memory is you want
13:47
to try running it at a couple different memory levels like running it at like 2 4 6 8 something like that and then uh
13:55
one of them is going to run all the time and once you have the one that runs all the time for like a couple days that's
14:02
the the the smallest number that runs all the time for a couple days is the kind of way that I like to figure out
14:09
the right number here because one of the things that's tricky about this is you don't want to have too much padding
14:14
because if you have too much padding you're just wasting all this memory but you also don't want to have too little padding because then uh the chance of
14:21
like um out of memory is annoying and when things break like engineering time is one of the most expensive uh things
14:30
for um uh like a company data engineering time like your actual time
14:36
is more expensive then the cloud costs of a little bit extra memory usage like a little bit of wasted memory usage is
14:43
going to be worth the trade-off of your actual time on the job so that's the way
14:49
I like to think about it but I also am against just setting it to 16 gigs and then just call just setting all jobs to
14:54
16 gigs and calling it a day because I think that that's terrible um then you have executive cores so generally
15:01
speaking an executive can take four tasks at once um you can go up higher to
15:07
six I would say you can go all the way up to six if you want and then this makes it so that you can get more
15:13

parallelism per executor but if you have more than six the thing that happens is
15:18
you get a different bottleneck right the bottleneck in that case is the executor disc and the throughput between like
15:25
when tasks finish and so you don't want to crank this up higher than six because
15:31
you end up bottlenecking yourself uh by the throughput of too many tasks running at the same time and so generally
15:37
speaking you want it to be four I don't like I don't generally touch this setting I've seen people touch this
15:43
setting and uh see some results but generally speaking I don't touch this
15:48
but maybe crank it to five or six if you want to get a little bit more parallelism make your job a little bit
15:54
faster uh but more than that like it's a trade-off right you can't just crank this up to like a th000 or something
15:59
like that cuz obviously your executive doesn't have a thousand cores to begin with like but like maybe it can leverage
16:05
a couple more um uh you want to be careful there too because like if you crank it up to six then you have six
16:11
tasks on one executive and it also makes it more likely to O because um depending
16:18
on the memory footprint of each of those tasks if they're all running at the same time then uh it could it's more likely
16:25
to om because the the probability that you get a skewed you have six tasks that are skewed a
16:31
little bit more is higher than if you have four tasks that are skewed so um
16:36
that's just another thing to think about um and then you have the memory overhead just like you have with the driver uh in
16:42
this case uh you really want to maybe bump this number up if you have jobs that have a lot of udfs especially like
16:49

if you're using p spark py spark udfs are terrible um and uh just udfs a lot
16:55
of complexity a lot of joins and unions and all sorts of like if the plan is crazy and like you need your jvm might
17:02
need a little bit more um uh of the memory footprint the thing about this is
17:07
I like about the memory overhead is that you can increase the memory overhead without increasing the memory itself and
17:14
you can make the job more reliable at no additional cost so that's the thing I
17:19
like about these overhead settings is that they can actually make a difference so definitely try them out uh especially
17:25
if you have a job that's a little bit unreliable uh that's uh a good kind of setting that you might want to fiddle
17:31
with so remember we're going back to the analogy real quick so we have the plan
17:36
which is the play that the the players make you have the driver which is like
17:41
the coach and you have the executiv that are the players that actually run the play um all right this is a very very
17:47
important slide uh in spark here so generally speaking um you get
17:54
uh there's essentially three types of joins and Spark you have uh Shuffle sort
18:01
Shuffle sort merge join broadcast hash join and bucket join these are going to
18:06
be your three um uh joins that are important uh
18:14
Shuffle sort merge join is going to be the least performant uh broadcast has join and bucket join are going to be a
18:20
little bit more performant um but Shuffle sort merge joint is also useful
18:27
because it's the most versatile because it works pretty much no matter what whereas broadcast and bucket join work
18:33
under very specific circumstances so broadcast hash join works when one side

18:39
of the join uh is small and it actually doesn't
18:44
have to be the left side of the join it just one side of the join needs to be small and uh then it works pretty well
18:51
because in that case instead of shuffling the join it just ships the whole data set to the executive and you
18:57
don't have to shuffle at all and it it works great uh for um smallish amounts
19:05
of data like I've been able to broadcast up to like 8 to 10 gigs if you if you
19:10
try to broadcast more than that you're going to bump into problems with executive memory at some point so that's
19:17
a thing to think about when you're um determining which joins to pick bucket joins are going to be where if you want
19:23
to do a join without Shuffle you can do that with um a buck bucket join we're
19:29
I'm I have some diagrams in the uh some future slides that will kind of explain
19:34
more about like how this works but so you have Shuffle short s merge join
19:40
broadcast hash join and bucket join and being able like like in spark you get you use do explain and it's going to
19:47
bring up these words so you need to know what each one of these words means if you want to be able to optimize your
19:52
spark jobs generally speaking you want to minimize the amount of times you use suffle Shuffle sort merge that is a very
19:59
hard word to say too many s's um but broadcast and bucket joins are going to be your best friends in a lot of cases
20:06
so um so let's just talk about Shuffle real quick because I think Shuffle is an important part of this puzzle so um uh
20:14
Shuffle is going to be like the least scalable part of spark that's the thing
20:20

like and like the more you Shuffle and the as scale goes up
20:26
the the the usage of Shuffle is just gets more and more painful and it gets
20:31
to a point when like once you're over like 20 or 30 terabytes if you're trying to process more than 20 or 30 terabytes
20:37
a day then um Shuffle just goes out the window and you can't do it you got to do you got to solve the problem in a
20:42
different way so anyways you can think of like uh imagine you had a job that
20:48
has you have a table that you're reading in and it has um four files and what
20:53
we're going to do is we're going to do a map operation first maybe we add a column or we filter it down so map
21:00
operations here are going to be where or um you add a column like you like add a
21:05
new column to it you like select star comma and then you add another column or like dowi column if you're using like
21:12
the data frame API and then you add columns and then uh and the thing about
21:18
that is it's infinitely scalable for the most part it's scalable to the number of files and you don't have to do anything
21:23
and you don't have to shuffle but then what happens is like imagine then you need a group bu so in this case we we
21:30
we're saying that there's three partitions in spark the default is actually 200 but I didn't want to build
21:37
a diagram of 200 squares so this is just kind of a simplified view of this but what ends up happening is imagine you
21:44
have um you're grouping on user ID in this example here and then everything in
21:50
file one all the user IDs if they um you divide the user ID by three and then if
21:56
and then you look at the remainder if the remainder is zero it goes to partition one if the remainder is one it

22:01
goes to partition two if the remainder is two it goes to partition three same thing happens for file two file three
22:07
and file four and so all of the data from so all the data from file four will be kind of in files will be in Partition
22:14
one partition two and partition three depending on where they're at
22:19
so this is how Shuffle works and this is how like if you imagine on
22:26
both sides so this is just like a Group by operation in this case but if imagine uh instead you were doing a join where
22:33
files one and two were part of one table and files three and four were part of another table and then you want to join
22:39
them together then you can do that the same way because then they they'll Shuffle and then all the data like all
22:46
the user IDs that um are divisible by three go here right and then you get all
22:52
and so then you get all the right data in each partition and then you can do the comparisons then you can actually do
22:58
the join on the data with the relevant data there so that's essentially how um
23:05
like this is this is the default case this is going to be that shuffle sort merge join that's going to be how that
23:11
works right so uh the difference here for um if you're working with uh like a
23:17
broadcast join is instead you don't even need to shuffle because what you can do
23:23
is the broadcast join can just files three and four are small enough that you can just ship all the data to each of
23:30
these executiv to file one and file two and then they can do all the comparison and um and and all that stuff in this
23:37
spot so they don't have to do any shuffling at all and that can be a very powerful uh way to go about things

23:43
anyways uh like we're going to go back to this slide in a second because I want to talk more about buckets but let's uh
23:48
let's go a little bit deeper into this presentation so Shuffle partitions and parallelism
23:56
are linked they are are actually the same thing it's kind of weird like spark is
24:02
so weird CU has these two settings as spark. sq. shuffle. partitions and spark. default. parallelism and they are
24:09
for the most part the exact same thing like like unless there is one one tiny
24:15
exception and that's like if you are using the rdd API directly then uh
24:21
spark. spark. default. parallelism is the is the setting to use there uh but
24:27
if you're using anything else data frame spark SQL data set API any higher level API which you should be doing don't use
24:33
the rdd API like that you should almost never ever ever be using the r rdd API there might be a small tiny Edge case
24:40
where you should but like 99% of the time you should not use the rdd API you should be using the higher level apis
24:46
that's guidance from data bricks and from spark as well so these Shuffle partitions are linked with parallelism
24:52
this is essentially if you go back to this slide here this is the number of partitions that you get and when after
24:58
you do a join or a group by so um let's let's talk a little bit
25:05
about Shuffle so is Shuffle good or bad uh I think so far in this presentation
25:10
you might be considering Shuffle is kind of bad Shuffle is probably not what we want and I uh I I don't agree with that
25:18
I think that shuffle is actually pretty decent that like you should probably uh use Shuffle most cases uh especially if
25:25

the volume is not really really high uh in this case I'm saying volume being greater than 10 terab even one terab
25:32
shuffling is pretty decent um and so uh I I have an example here I want to go
25:37
over real quick and I I'm going to I'll send you guys the link to this there's some writing online about it as well but
25:43
um so one of the pipelines I worked on in spark this is the biggest pipeline I
25:48
ever worked on in my entire career was uh it processed about 100 terabytes an hour in spark and we had to do a join
25:57
and so the what we needed to do was every Row in this data set was a network request it was every Network request
26:03
that Netflix received and we wanted to join that data set with a data set that
26:08
mapped that Network request's IP to a micros Service app like because Netflix
26:14
has thousands of microservices inside of its architecture and we wanted to figure out which IP like where was this request
26:21
coming and where was it going just so we could see like which app is talking to which app that was the whole idea behind
26:27
this pipeline and so uh one of the things that was nice about that was we were actually
26:33
able to ship all that lookup table that IP lookup table we were able to get it down to like you know um several
26:40
gigabytes and we were able to broadcast join it and it worked um then uh a new
26:47
requirement showed up for that Pipeline and that was because Netflix wanted to
26:52
upgrade all of their services to IPv6 instead of ipv4 uh then at that point uh there we
27:02
we we were trying to ship things over an IPv6 and I don't know if y'all know but IPv6 is I think 100 or 200 or or it's
27:09
like 100 or a thousand or 10,000 times more um IP addresses than ipv4 so it
27:16

goes from like billions to like quintilian or something like that like it's a very very much bigger um search
27:22
space so when we tried to move over then uh it no longer was a broadcast join and
27:27
then it had to be Shuffle join and it didn't work didn't work couldn't do it like that was just literally not
27:33
possible like uh because it was too much data and there was no way that this join was going to work and so that was a very
27:39
painful thing to realize and we realized that okay Shuffle actually can bring some pipelines to its knees I found that
27:46
uh there was a lot of some other cases for me at Facebook in when I was doing some like feature Generation stuff for
27:54
notifications where uh we had to join two tables at the notific level so you had like notification join notification
28:01
and then it was like 10 terab on one side and 50 terab on the other side and we tried to join him together and like
28:08
they did that for a while and like that was what they did but it took up like 30% of all of the compute that we had
28:14
for all of notifications this like one whale Behemoth of a pipeline and so you want to be careful in some of those
28:21
cases like so what do you do like in those cases so I think uh depending on
28:26
the the use case you can kind of do things in uh separate ways I want to talk about how I solve both of those
28:32
problems so um let's talk about the notification one first so and for the notification data sets that we're
28:38
joining at that very um at that at that cardinality down there like instead of
28:45
Jo like uh doing a shuffle sort merge join we bucketed the tables and we
28:50
bucketed them on user ID so what we did was we put each side of the like both

28:56
the both the left and right side of the join into um 1,24 buckets and then what
29:02
we did with that was then when you do the join you can do the join without Shuffle because if you go back to oh
29:10
forward I mean because in that case when you when everything's bucketed you don't
29:15
have like the files already uh they already have the guarantee that the the
29:21
data all the data that you're looking for based on that ID already exists in
29:27
that file you already have that guarantee because it's bucketed and that's what the files essentially how
29:34
you write the files is based on the user ID it's you modulo the user ID and then you write the buckets out and then when
29:40
you do the join you just match up the buckets so in this case imagine you had two tables and they both had two buckets
29:47
and then and they were both bucketed on user ID then instead of having to map like file one to partition 1 two and 3
29:54
and file two to one two and 3 what you could do is you just line up file one and file three cuz file three is the
30:00
first bucket of the second table and file one is the first bucket of the first table and you essentially just line up the buckets and then uh
30:07
everything works and then you don't have to shuffle at all and you get a massive massive massive performance gain if you
30:13
do that with buckets and uh interestingly enough um you can actually
30:21
even if the two tables don't have the same number of buckets you can still do
30:26
a bucket join assuming that they are multiples of each other so imagine in
30:32
this case instead we had um one table had uh one bucket and the other table
30:38

had two buckets or you can think of like four two and four or four and eight well in that case like files one and two
30:45
would have all of the data in files three because like it would be all of it would line up because they're multiples
30:51
of each other so the big lesson here is that when you're bucketing tables always
30:59
bucket your tables on Powers of two always use powers of two don't be weird
31:05
and be like I want seven buckets or like 13 buckets or something like that because then like the only way that you
31:11
can get a bucket join is if the other side also has 13 buckets and that might not be the case like um and so one of
31:19
the I think bigger questions here is like how do you pick the number of buckets well um in in some of these
31:24
cases it's based on the volume of the data right and then you want to kind of divide it where it's like okay we uh I
31:30
know for um like Facebook like we had 10 terabytes and so like okay a, 24 buckets
31:38
we were like shooting for um like in that case I think like 10 gigs a bucket or something like that that was kind of
31:44
the idea but like uh there's kind of a rule of thumb there where like cuz if the if you bucket if you have too many
31:50
buckets on data that's too small then you have that same problem where um
31:57
there might not any data in one of the buckets because of the modulo problem where like if say say you have like a
32:04
thousand rows and you're like okay I going to bucket this into 1,24 buckets
32:09
then you're guaranteed in that case to have 24 empty buckets by the I don't know if y'all know that the pigeon hole
32:15
principle in combinatorics but you're guaranteed to not have enough data in one of the buckets and if you have an

32:21
empty file that can mess up with a lot of these systems and they it doesn't trigger the bucket join the right way so
32:28
like that's where uh you want to make sure that like you you you aren't putting too many buckets on tables that
32:33
are small that's just uh kind of an aside so that's how I solved the problem
32:38
at um at Facebook but uh what about the problem at Netflix that that problem was different so um one of the things about
32:45
that problem was um bucketing the um 100 terabytes an hour was going to be very expensive anyways so because you have to
32:52
shuffle that's the only way that you can do it is you got to shuffle that thousand or the 100 terabytes an hour so
32:58
in that case we actually solved the problem Upstream where what we did was we got everyone to log their app when
33:06
they received a network request so we went and talked to all the microservice owners and we're like just log your app
33:11
please and that's how you solve the problem so you lo you just put the you put the app in the data so then you
33:16
don't do it just the join just goes away and that was how we solved that problem obviously that problem that that
33:22
solution was also a massive massive massive pain because you have to talk to like hundreds and hundreds of people but
33:28
that's how we solve that problem and so there's a bunch of different ways to solve these problems but um and know
33:34
that like what I just said is that like you might not want to use a spark
33:40
optimization technique to solve your problem just like what I said right there right where it's like maybe it's a logging problem maybe you're trying to
33:46

solve this problem with a hammer CU you're good at spark and so every problem seems like every nail seems like
33:52
you need a hammer to hammer it down and that's not always the case like sometimes you need to solve the problem upstream and you know work and have
33:58
people like other people give you give you the data the right way as opposed to trying to bring it all in especially as
34:06
you get to bigger and bigger scales that's usually how you solve the problem we're going to talk a little bit
34:11
about minimizing downstreams I talked a bit about this already but um bucket the data if multiple joins or aggregations
34:18
are happening Downstream bucketing is great but like if you only do if you're bucketing and you're only doing one join
34:25
then usually that's a waste because like you you because you have to pay the shuffle cost already to bucket that's
34:32
like that's the cost cuz you have to in order to write till a bucketed table you
34:37
have to you have to put that in there you have to already have that in there so um and then if if but and then if
34:44
you're only doing one join with it then like why are you doing that like you only get you really only get benefits
34:50
from bucketing is if you are doing multiple joins uh with these tables because then that's when you're going to
34:56
really see the big benefit so um that's a good thing remember the one thing about bucket joins that I want
35:02
to talk about is uh Presto can be weird with bucketed tables because especially
35:08
if you have tables that have a small number of buckets maybe it's like eight or 16 or something like that then when
35:14
you query stuff with Presto like uh the query might be slower because the initial parallelism will always be 16

35:20
and like maybe Presto wanted more and like when it's a non- buuck table it can split the files up and it can like
35:26
handle it better but uh generally speaking I like buckets buckets are good and the last thing I wanted to say is
35:33
that if you're playing around buckets always use powers of two 2 4 8 16 you know 32 64 128 256 512 1,24 you know
35:42
just powers of two uh always use powers of two uh so minimizing Shuffle is I
35:48
would say one of the most important things that I did in my career that has helped me uh move up the ladder cuz you can save a lot of money especially at
35:55
Big scales with this um okay let talk about uh Shuffle and skew
36:01
skew is another one that I noticed that was also I had to deal like sk's gotten
36:06
so much better though oh man like y'all are lucky like like I I I feel like this is one of those like back in my day like
36:12
things were difficult right so um so skew happens when you partition your
36:18
data right where you have that modulus thing right where you divide by 200 and then you split up all the data and the
36:24
problem happens right is where one of those 200 partitions has way more data
36:32
than the other partitions like what you can imagine in this case is like imagine if like Beyonce and Justin Bieber and
36:40
like I don't know like the rock they all somehow miraculously all of their user
36:46
IDs are divisible by 200 and then what happens is every single time all of
36:52
their notifications go to the same executive and then it's like them and then like a bunch of random people and
36:57
then like but the just those three uh users are going to have dramatically more data than everybody else because
37:04

they're famous and so they're going to skew that one executor is just going to be given like I don't know 10 or 100 or
37:12
a thousand times more data than the other ones and that can be a showstopper actually I mean like I
37:19
mean that's taken so many pipelines down and another symptom that can happen here is your jobs will get to 99% and then
37:26
fail that is probably the most infuriating thing about data Engineering in my mind
37:32
like at least from a technical perspective is like skew is like such an
37:37
because it's like okay we're going to like we're going to almost run we're going to we're going to let your
37:42
job run for two hours and then fail and then you're like why did it fail like come on right and so that is why I I
37:49
don't like skew but things have gotten a lot better with skew and we're going to talk a little bit more about how to solve these problems so um how to tell
37:57
your data is skewed right I was talking about this um the most common symptom is your job is going to get to 99% and it's
38:04
going to take a million years way longer than you expect the job to take and then it will just
38:10
fail and it's sad because you're like wow why did it fail um another way
38:16
another more scientific way is you can actually like do a box and whiskers plot and actually like map it out and do it
38:22
like a data sciency way um I've literally never done that but if you want to do that can do that and like
38:28
have more uh confidence that your data is skewed I almost always just like build my Pipeline and then get mad when
38:34
it fails at 99% and just like kind of do it the do it the hard way and not the scientific way but that's how you want
38:41
to know tell if your data is skewed uh what's talk about how you can actually solve this

38:46
problem okay so the most effective way to solve this problem is but it's only
38:52
available in spark 3 and uh we're going to talk about it real quick is you need
38:57
to uh set your um set this spark. SQL adaptive do enabled and just set it to
39:04
true and that will solve your problem done like you don't have to do anything
39:10
else it's amazing like I I love this setting because then you don't have to do all like well I'm going to talk about
39:16
one of the ways that you can do it um if you don't have spark three and you want
39:22
and you have a skewed Group by this doesn't work with a skewed join but it works with a skewed group
39:28
where what you do is you essentially uh you add a column which is just like a
39:34
random number because then what that does is it will break up the the skew
39:39
because then uh some of the skew will go to one executive some some will go to another executor and then you essentially do two aggregations where
39:46
you aggregate once by where you Group by the random number and then uh and that
39:51
gives you like a partial aggregation and then all the skewed data will be small at that point because it will all be
39:56
back down to like whatever your group by grain is and then you aggregate again
40:02
and you just remove the random number and sum everything up again so you do two sums and that will give you the same
40:09
uh thing as you would get um otherwise so you want to be careful using this uh
40:15
uh I don't know if y'all remember from u a previous presentation I talked about additive and nonadditive dimensions and
40:23
this is a great example of where additive and non-additive dimensions can mess you up cuz just because uh if you
40:30

sum something twice like this they might not actually add up to the same number like and but it depends on like or add

40:36

up to the correct number and it depends on the dimensions so but generally speaking most dimensions are additive I

40:42

think like about 80 90% of dimensions are additive and so you can use this and it works great but make sure that you

40:48

are aware of that before you use something like this um but generally just use spark. sq. adaptive do enabled

40:55

um obviously don't just set this to true all the time like just in case because

41:01

it makes the job more expensive to run cuz spark has to compute all these like extra statistics about the job while

41:07

it's running and you don't get those for free and it makes the job slower but it makes the job more resilient to skew and

41:13

so that's why it's great and hopefully y'all are just on spark 3 so you don't have to do this like random salting hack

41:19

um I want to talk about one more way to deal with skew uh in Joins and this is

41:24

what I did at in at Facebook was um if you have a join the problem is is like

41:31

you can't have this like random number thing it doesn't work the same way and so uh what I did in that case was I

41:37

identified the outliers and like just filtered them out and then just was like

41:42

okay we don't need these IDs and then I was like Beyonce can just wait they can wait and have data tomorrow and

41:49

sometimes that is an an acceptable way to deal with skew is you filter out the outliers or what you do is you partition

41:56

your down stream table where you have one side of the pipeline that processes everybody else and then you have the

42:02

other side of the pipeline that processes the outliers and in that case you can usually uh that can usually work
42:08
pretty well so that's another way that you can kind of manage stuff but those are the essentially the main ways to deal with
42:15
skew okay let's talk a little bit about the ways spark is kind of ran um so uh
42:24
y'all know we were going to use spark on data bricks and then it kind of fell through but uh spark on data bricks and
42:31
Spark unmanaged spark kind of have uh different ways that you want to run and
42:36
test your job so um generally speaking uh should you use like a pie spark
42:43
notebook when you're running your job uh this is something uh I remember back in 2018
42:50
when I was working at Netflix and then they got this working because Netflix kind of pioneered the idea of running
42:55
notebooks in production and I was like I still don't like it I still don't like
43:01
it at all I don't like the idea of running notebooks in production because it's like unchecked mutations you don't
43:06
you're not using git like there's a lot of problems with it but data bricks like
43:12
they all are about it like that's what the whole UI is all about they really want you to use notebooks right data
43:17
bricks are all about it so um but in big Tech you should probably only use notebooks for proof of concept and then
43:23
check your spark code into like a git REO and then then you're going to use this thing called spark submit um from
43:30
the CLI in big Tech whereas like you can just run the notebook in um data bricks
43:35
so that's kind of like the difference between spark on data bricks and regular spark uh like thinking about like kind
43:42

of best practices I personally like unmanaged spark using git and uh spark submit for my jobs I'm I'm kind of
43:49
anti-n notebook because I feel like notebooks they don't invite uh unit tests and they don't invite integration
43:57
tests and they don't invite good engineering best practices but they are they invite less technical people like
44:04
they invite people who don't have all of the technical things and allow them to
44:09
actually leverage spark so that's where they're useful is they actually allow
44:15
people to use spark and they don't have to be so technical so that's why they have adoption but as data Engineers I
44:21
generally think that it's like not the right way to go so um anyways that's spark on data bricks versus regular
44:27
spark I just like to call that out um so remember I was talking about
44:32
all those different types of joins uh one of the big things that you want to do and we we're going to check this out
44:37
a little bit today in class is we're going to uh look at this do explain so
44:43
you can look at uh spark will actually tell you everything that you want to know about it if you use the explain um
44:50
kind of uh in your data frames and that's going to give you a lot more insight into actually how spark is
44:56
running underneath the hood definitely check that out it's very similar to like explain in Sequel you can actually I'm
45:03
pretty sure you can use spark in spark sequel like the same way where you put like explain at the
45:08
front Okay uh another thing just talk about is like where can spark read data from um I'm sure y'all know just like
45:16
the answer here is everywhere for the most part like from the lake Delta Lake Apache Iceberg hype metastore postgress

45:22
Oracle rest API calls um you want to be careful I think that's one of the things here I want to talk about real quick is
45:28
like when you make a rest API call it's almost always done in the driver and um
45:34
remember your driver memory settings so like if you are making a rest API call and the response or you're making many
45:41
API calls and the responses are big then you want to be careful how you're
45:46
processing that data in spark because it is a lot more prone to running out of memory so and in those cases like a lot
45:54
of times you might want to like uh parallelize them so like what you can do is like you get a list of URLs and then
46:00
you do spark. parallelize and then you essentially make the API calls in the
46:07
executive uh it's a trade-off though because uh the the nastiness of that is then uh you essentially Hammer the API
46:14
in parallel and that can be a problem and that's like a separate problem versus like managing all the memory in
46:22
uh the driver and those are two separate problems that I I I find so for me me I
46:27
generally find apis to like not be very like like rest apis I don't find them to
46:32
be very spark friendly unless it's like if it's like one call then usually it's like whatever but if you're making a lot
46:39
of calls spark might not be the right way to do it so um that's a thing to think about like a lot of times like the
46:46
way that spark likes to do it is instead of calling and hitting the a the rest API they like to read directly from like
46:53
the postgress database or the like the inner guts they to essentially bypass the API and read directly from the data
47:00

tables and Spark's going to have a way nicer time doing that than reading from an API so definitely know that whenever
47:07
you're doing your Integrations that that's probably going to be the smoother integration is like where you go up a
47:12
layer and you read from the data tables as opposed to trying to read from the API and obviously flat files we're going
47:18
to be using flat files today in the lab so um that's uh pretty much it for house
47:25
Mark and read data um so spark output data sets um almost all
47:31
data sets in spark when you output them should be partitioned on date uh uh this
47:37
date should be the execution date of the pipeline which is like if y'all use airflow there's like uh it's like curly
47:44
brace curly brace DS curly brace curly brace like it's like that is like the peram it's like the date that you want
47:51
to backfill or the date that you want to run the pipeline for that date should be passed to spark and then Spark can you
47:57
know write the right data accordingly uh DS partitioning is very powerful definitely look that up if you're not
48:03
familiar with that cuz if you're not partitioning on date like your your tables are going to get way too big and
48:08
way too bulky so good thing to remember congrats on making it to the end of the spark Basics lecture if you're taking
48:15
this class for credit make sure to switch over to the other page so that you can do the lab and get credit there as well welcome to the basics and Spark
48:22
Advanced setup example so what we're going to be wanting to be doing here first is make sure you have Docker desktop running and installed uh that's