Fact Data Modeling

Fact Data Modeling Day 3 Lecture
How Meta models Big Volume Event Data

Reducing shuffle with reduced facts

Transcript:

why should Shuffle
2:48:40
be minimized well uh Shuffle the thing that sucks
2:48:46
about Shuffle is that it's it's the bottleneck for
2:48:52
parallelism so if you have um a pipeline that doesn't do use
2:48:59
Shuffle it can be as pretty much as parallel as you want it to be uh because
2:49:04
of the fact that you don't need um a specific chunk of data to all be on one
2:49:10
machine you can have it where like all of it split out and we're going to talk more about like what that means and the
2:49:16
parallelism of stuff in another slide but keeping in mind that like some steps
2:49:22
in your big data pipeline are going to have more parallelism than other steps in your big data Pipeline and because of
2:49:29
that like the usually the more paral parallelism that we can have the more
2:49:35
effectively we can crunch Big Data because that is going to be what makes it possible for us to use more machines
2:49:44
and to have less bottlenecks because Shuffle you can think of Shuffle as the bottleneck it almost always is the
2:49:50
bottleneck for um uh if you were working with very high volume data so there's a
2:49:56
couple different ways that we can you know think about minimizing um Shuffle but uh we we'll be going into that
2:50:02
there's like a there's like a modeling way to do it there's like also like more um kind of tactical ways to do it that

2:50:09
we'll talk about uh so yeah let's let's let's talk more about Shuffle shuffling is so
2:50:14
fun when you are working with big data uh there is you have in SQL right you
2:50:21
have a couple different keywords select from where Group by join having order by
2:50:27
um these qu uh these keywords uh affect Shuffle in different ways um for example
2:50:34
uh if you have a query that is just SE that just uses select from and where and
2:50:40
importantly your select doesn't use a window function but like you so you have select from and where without a window
2:50:47
function that is what is considered a query that is infinitely scalable you could
2:50:53
essentially have it where like if you think about it this way like say you had a billion like let's think about it in
2:51:00
like kind of the most extreme sense here right so say you had a billion rows and
2:51:05
you had a billion machines and each machine had one row um the thing is is
2:51:11
like with select from and where that's fine right because one machine doesn't
2:51:18
need a have more data or less data because it all it needs to know is like okay here's the data I have does this
2:51:25
data fit this condition or not and if it does okay bring it out otherwise don't
2:51:31
bring it out and that's that's it that's all like you don't have to have like a specific chunk of data on a specific
2:51:37
machine that's the big thing to remember about select from and where is that like if you're using those three like you can
2:51:45
go as parallel as you want and that is um very very powerful if you can
2:51:51
understand that fact because you don't you can use if you have queries with those three keywords and only those
2:51:57
three keywords like uh the job is going to essentially come back instantly the
2:52:03

job is going to be no Shuffle the job's going to be cheap the job's going to be all those things right so now let's talk
2:52:11
about the ones in the middle uh here we have group by join and having so why are
2:52:17
these not extremely parallel well the
2:52:22
main reason for that is because so say you have group by right and let's go
2:52:27
back to that example where we have a billion rows and a billion machines right and there's one row on each
2:52:34
machine well with Group by the problem is is like in order to actually do the
2:52:39
right aggregation all of the rows for that key so say there's me and I'm
2:52:45
someone who has 30 rows in this billion row data set right and um all of all 30
2:52:52
of my rows need to be on one machine they all need to be moved over into onto
2:52:58
one machine in order to correctly count them right because if they're all separated and my 30 rows are all on
2:53:03
different machines then we how do we know that there's 30 we don't know like that's like and that whole process of
2:53:11
collecting like so the data is in a bunch of chunks and it's all uh separated out into like these this
2:53:16
comically chunked out like one a row um per machine kind of
2:53:22
situation uh and in that case we'd have to move all that all those into one machine where then it has all 30 rows of
2:53:29
data so then it it can count at that point and that's why Group by triggers Shuffle and that's what all Shuffle is
2:53:36
right is Shuffle happens when you need to have all the data from a specific key
2:53:43
on a specific machine that's all it is and so um and that's the big thing to
2:53:49
remember when you're doing shuffling and so that's how Group by works in this case right is then you would reshuffle
2:53:55

and then uh and and essentially what it does is you get a pick you get a pick the parallelism there right because uh
2:54:02
in spark there's a thing called spark. SQL do shuffle. partitions which is a configuration that you can pick uh which
2:54:10
is something we'll talk about in week five as well but uh the default is 200 so we would have our billion machines
2:54:17
they would pass all of their data to 200 machines based on the kind of the key
2:54:23
right so say you have user ID and the user ID is an integer and then what you would do is you would take the like the
2:54:29
modulus so you would uh take the you would divide the user ID by 200 and whatever the remainder is that's the
2:54:36
machine it goes to right so say it's like uh user ID is 207 so that would go
2:54:42
to the seventh machine right and user ID 4002 would go to the second machine
2:54:47
right Etc Etc and that's like how they kind of like split it out and uh and you can
2:54:54
change that number that 200 you can change it to whatever you want 200's just the default but anyways group bu is
2:55:00
going to do that it's going to kind of bring everything back together and it will do things where you have those 200
2:55:07
uh kind of machines that will process and then aggregate the data once it's all collected um along whatever whatever
2:55:14
you're grouping on then it will then do it that way uh what about join uh join is actually even trickier uh in some
2:55:21
regards because join you have to do this not once but twice so what you do in the
2:55:27
join case is you have all of the keys on the left side and all of the keys on the
2:55:33
right side and then the keys on the left and the keys on the right all have to be pushed to one machine and that's going
2:55:39
to be one partition in your Shuffle partitions which again defaults the 200 but then after that then the comparison

2:55:45
happens to see like oh do these match or not and then if they do match it keeps that record and if they don't match it Chucks
2:55:52
it away or maybe not if it's a left join or a right join or whatever like depends on the join I'm talking about inner join
2:55:58
in this case um then having so having I I I I was having difficulty having
2:56:06
difficulty putting it into a a bucket here I put it in kind of parallel because of the fact that you have to
2:56:12
have having having and group by kind of go hand inand so technically having is
2:56:18
like as parallel as select from and where because it's just again a filter condition but it's a step after group bu
2:56:25
so you can only really apply having after Shuffle so um that can be a big
2:56:31
thing right uh and then let's talk about the last one here uh in distributed
2:56:36
compute uh which you should almost never use right uh is order by order by is the
2:56:44
most painful and least parallelizable uh keyword in SQL and here's why so if
2:56:53
you think about order bu like um especially like I'm talking about order by at the end of the query not order by
2:56:59
in a window function order by in a window function is a little different and we'll talk about that in a second but if you have order by at the end of a
2:57:07
query um what ends up happening so say you order by user ID and you need to do a global sort of all the data and it's
2:57:14
like on and say it's on 200 machines or it's on a billion we'll go back to the comical example where we have one one
2:57:20
row of data on a billion machines and then we need to make sure that all the data in all those systems is sorted the
2:57:26
only way that we can make sure that that is the case is if all of the data goes on one machine and it all gets passed
2:57:33
through one machine and so uh which is I don't know if you all know but that's
2:57:39

like the opposite of parallel right because that's that's one machine that's sequential at that point and that is
2:57:46
where uh a lot of times you'll see uh this can be a very big symptom in big data jobs if you have order by like you
2:57:55
can use orderby but use it at the end after you've like aggregated and the number of rows that is left is like in
2:58:02
the thousands or tens of thousands if you're using order bu and there's millions of Records like your job's
2:58:10
going to have a bad time it's GNA have a really bad time so how is this different
2:58:15
in a window function and like why is a window function not going to be at like if you use order in a window function uh
2:58:21
the main reason for that is a window function doesn't do a global sort it can
2:58:26
it can right if you don't put any Partition by in there and if you do like a if you're doing like a global like a
2:58:33
global rank function so if you use like Rank and you don't Partition by anywhere in the um if you don't use the Partition
2:58:39
by keyword then you're going to have the same nasty terrible parallelism problem
2:58:44
exact same problem like uh so like but if you're using Partition by which you should then uh then it's no longer a
2:58:52
global problem and you don't have then then it goes back to the shuffle problem where you end up sorting based on the
2:59:00
number of Shuffle partitions which is usually 200 but you can change that if you want and because that partition bu
2:59:06
is going to create that same sort of thing where like we need to cut up the data and pass all of that same data to
2:59:13
one machine and that's the uh so partition Buys in that way it's really
2:59:18
weird cuz Partition by in a window function and group by in uh like a regular query like in the distributed
2:59:26

compute World those things are those things behave very very similarly so especially in terms of how it affects
2:59:32
Shuffle so that's where order by is an interesting one like you can use order by in window functions only if you also
2:59:39
have Partition by like if you do it by itself like you're going to have a bad time so why am I talking about this like
2:59:47
some of this your probably like Zach like I thought we were talking about fact data modeling like why what this doesn't seem like anything that's
2:59:54
related to fact data modeling at all I'm just like I'm going off on this rant about spark and distributed compute and
3:00:01
one of the things though that I think is very important to remember about this stuff is that how the data is structured
3:00:10
determines which of these keywords you have to use and if you have uh if you
3:00:15
have data that's structured in a certain format you can skip out on using some of
3:00:20
these keywords so that's a big thing that you want to think about when you're like kind of going through and building
3:00:26
out uh your fact data modeling is that so remember this uh at the very least
3:00:33
just remember this uh this slide is very powerful it has more of an impact
3:00:39
besides just on data modeling exercises it can help you troubleshoot uh spark problems and other problems as well so
3:00:47
um it's it's great so let's go to the next slide okay so group bu right we talked
3:00:52
about how group buy causes Shuffle so um how can you fix that problem so there's
3:01:00
two two ways that you can fix this problem uh and here's here's essentially
3:01:05
how it works like you can in spark or in uh not not just spark but like with S3
3:01:12
or any um kind of like or Iceberg or huie or Delta lake or whatever they they
3:01:18
all support this is you can bucket your data so instead what you can say is like
3:01:26

you can essentially do this you can pre-show the data uh for spark so like
3:01:31
say like you're like okay I want to put my data into eight buckets and then um
3:01:36
when you and then you bucket based on a key probably it's going to be like a user ID or a device ID usually it's some
3:01:44
sort of like high cardinality integer field and then you bucket on that field and then you and then it does that whole
3:01:50
modulus grouping when you write the data out so if we have the modules grouping
3:01:57
when it writes the data out then when we do group by we don't have to shuffle uh
3:02:04
because it's already it's already in the buckets are already set and we already
3:02:09
have the guarantees that this data is here and this data is here and uh so
3:02:15
then the group bu can just leverage the buckets and it can just crunch it down and that can be very powerful that can
3:02:21
be a very powerful way to do group by without um uh without Shuffle and so
3:02:27
definitely we're going to talk more about that in the spark week uh week five and that's uh one way to do it um
3:02:34
but today uh we're going to be talking more about the other way which is all in
3:02:39
caps here which is reduce the data volume as much as you can and like CU if you can reduce the data volume and you
3:02:46
can uh uh also so reduce that that also reduces shuffle a lot and you can really
3:02:52
create uh way more efficient queries how can we represent fact data in a way that
3:02:59
really is efficient and wonderful and can do really magical things so let's
3:03:05
let's get into a little bit reduced fact data so we talked about
3:03:11
fact Data before back data often has a schema where you have like a user and then you have a time an action and a
3:03:19
date right this data is very very high volume incredibly high volume then you
3:03:27

have the next layer down is you can aggregate this up right so instead you could have like a user ID and then you
3:03:33
can have like an action count so in this case you could say like I to I made Seven likes on January 3D right and that
3:03:42
could be that daily aggregate um that is great the the the
3:03:47
volume is dramatically reduced that way right it's like it it gets cut by the
3:03:53
number of actions per user so a lot of times that might be 100x smaller um
3:03:59
that's good but even that data can be too big and I'll talk more about like kind of the problematic things that can
3:04:07
happen there and that's where we we're going to go one step further here where we talk about this reduced fact take
3:04:13
this one more step right so in this case what you have
3:04:18
is you have a user ID and then you have an array of action counts so you can
3:04:24
think about it this way it essentially cuts the daily data to uh it divides it
3:04:30
by 30 if you use month but it can also divide it by 365 if you use year so it's
3:04:36
like you can pick which way you want to do it but this data ends up being like you can either have one row per month
3:04:42
per user or one row per uh year per user
3:04:47
and those two thing those are uh a very powerful way to get the data essentially
3:04:54
as small as you possibly can get the data and that is what what can really
3:05:01
minimize the amount of shuffling that can occur and really minimize the the
3:05:06
whole thing like it's it's really powerful and we're going to talk more about like how this unlocks a lot of new capabilities but the key thing to
3:05:13
remember here is you got three flavors here you have like just normal fact data very high volume daily Aggregates medium
3:05:21

volume and then you have reduced facts which are the lowest volume um they come with come with some trade-offs though
3:05:28
especially between the the second two like between daily Aggregate and reduced fact they there's some interesting
3:05:33
tradeoffs that we're going to kind of dive into as well because as the data gets smaller you lose some of the
3:05:40
flexibility around what type of analytics you can do on it but that's like worth the tradeoff of being able to
3:05:47
do analytics more quickly so that's one of the things that you'll find in data engineering a lot of the time is that
3:05:55
when you are working with data and you aggregate it up the data gets smaller easier to work with in terms of like
3:06:01
speed of queries but then it's like you're like oh what if I want to do this or what if I want to do that and then like those some of the options are taken
3:06:08
away from you because the data is too aggregated up so let's go a little bit deeper into each one of these um okay so
3:06:15
this is what a fact scheme might look like where you have like a user idid you have a time stamp you have the action
3:06:22
that they took you have the date the date partition and then there might be
3:06:28
like other um properties like for example here it's like I made a like on this
3:06:34
post on my on an Android phone at this time and then here's like I commented on this post on an iPhone at this time and
3:06:42
so like the thing about fact data like when you're here it's very very granular and you can ask ask very very specific
3:06:50
questions of like whatever question Under the Sun that you want to ask but the tradeoff is like you can
3:06:58
only ask questions usually for like what has happened recently or like within a
3:07:04
couple days you can you can look at very small time Horizons and the reason for
3:07:09

that is because you end up um missing uh like the like the volume ends up being
3:07:16
unwieldy if you have too if if your time frame gets too big so you can't like the
3:07:22
fact data like especially if you're looking at more than a month of fact data like if it's high volume enough
3:07:29
then like a month of fact data is going to cause like all sorts of Mayhem and all sorts of problems and it's going to
3:07:34
cause like it's just going to cause your queries to be really slow and like they might not even run you might get some
3:07:40
weird out of memory problems like there's all sorts of interesting things that can happen when you do uh your fact
3:07:46
data modeling and or your analytics just right on top of fact data right but at
3:07:52
the same time like this is great like for other questions right if you're answering very specific questions like
3:07:58
what happened today or what happened in a week what happened in two weeks right um one of the things that I noticed at
3:08:04
Facebook is uh like so they had this thing where like after 90 days everything had to be anonymized so like
3:08:12
that policy also really impacted analytics a lot whereas like uh it made it so every everyone was just looking at
3:08:18
the last 90 days of data and like they didn't like look back further and they weren't trying to figure out like okay how do we get like the other days of
3:08:24
data in here as well but um anyways this is uh a crazy problem that can happen
3:08:31
but uh so fact data the key thing I'm trying to take get you guys to take away from this slide is
3:08:39
that this schema is very flexible very
3:08:44
good at answering very specific questions but it's very not good at answering
3:08:50
questions over a longer time Horizon once you have a longer time Horizon you're going to have a hard time using
3:08:57

this schema I promise okay so let's go to the next schema that kind of that daily aggregate right that was the next
3:09:04
one that we were talking about so here's daily aggregate so uh daily aggregate you have a couple things here right you
3:09:10
have again you have user ID then you have like a metric name in this case likes given you have the date and then
3:09:16
this is how many met or how many likes were given so now we have our daily
3:09:22
Aggregates uh this is going to look a little bit nicer um now in this case uh
3:09:28
you have one row per user per metric per day this is a little bit nicer uh
3:09:35
instead of having like you know for likes like for example this first row here there was 34 records in the old
3:09:42
table and now there's one and so that is a big difference right that's a that
3:09:47
aggregation is going to make this table a lot easier to work with and you can do a lot of really powerful things with
3:09:54
this table as well so one of the things that's really nice about this daily aggregate table is you
3:10:02
get um a lot longer time Horizons now
3:10:07
like I noticed at least when I was working in big Tech that like if you have this schema this works for like if
3:10:14
you want to look at like a year or two years years this works great this works
3:10:20
great and uh this like can work really well and the thing that's really nice
3:10:25
about it is you get these nice joins uh because say you have like an SCD right
3:10:30
so you wanted to know like what my primary device operating system was like
3:10:36
whether I was on Android or iPhone and like which was my favorite and that might change over time but the thing
3:10:42
that's nice about these Aggregates is you have that at least down to the Daily granularity obviously you don't have
3:10:48

like the hours you lose you lose a little bit there like if you have something that's changing like on an hourly basis then it's not going to work
3:10:54
but like most SDS are on a daily basis anyways so you can still join this table with scds and you can still do like um
3:11:03
Aggregates at the higher level of bringing in other dimensions and that can be a very powerful way to use this
3:11:10
schema so and it works great um those uh those year-long or twoe long queries
3:11:17
they do take a little bit of time though they take like I mean at least back then when I we were working with Hive and uh
3:11:23
those queries would take like like two days right is two days to work with and
3:11:32
that's like kind of a showstopper it's kind of not the way to go and you can't
3:11:38
analyze Dimensions very well uh if it takes that long like or like obviously 2
3:11:44
days is not that long but it's like still long enough to be paid painful so one of the things about this uh
3:11:50
aggregated data is it's very powerful so this this schema here this Daily
3:11:57
aggregated Facts is um has another name um it's called like a a metric
3:12:04
repository think of it that way where like in this case uh your partition on
3:12:10
this table is actually not date your partition is date and Metric name you
3:12:15
it's a you have a sub partition on Metric name so that you have two partitions for this table so that you
3:12:22
can you can dump in all your metrics into this daily day daily table and this schema is very very powerful like so
3:12:29
many of the um SCH like I don't know if yall have heard of deltoid deltoid is an experimentation framework at Facebook
3:12:36
that powers a lot of like the AB testing and all sorts of different powerful things it uses this schema to determine
3:12:44
if an AB test impacts uh different sort of metric values and all that kind of
3:12:49

stuff so um one of the things I noticed when I was uh working at Facebook though
3:12:55
was uh this schema could still be improved on we can
3:13:02
still make this smaller and not lose anything right there's still one more
3:13:08
layer we can do so this schema is what was very powerful when I
3:13:17
was working at Facebook so this schema probably looks a little bit crazy but we
3:13:22
are going to be working with this schema today and we're going to be building out literally this exact uh data model uh in
3:13:29
our lab today but let's kind of go over like how this works right so you have a
3:13:37
user ID you have likes given and then in this case there's only one row per
3:13:44
month right and then you have your value array so one of the things that I recognized
3:13:52
when I was working on this problem because a lot of this stuff like why I even had to come up with this model
3:13:58
right was I was working on these problems of like uh there was the first decline in growth in in in at at
3:14:06
Facebook in the history of the company and they were like what's going on what's going on we need to look at all the metrics we need to look at all the
3:14:12
metrics over the whole period of Facebook like how do we like figure out what the hell is going on here and like we can like troubleshoot this right and
3:14:19
that's where like we were mostly working with these like daily metrics and I was like wow these daily metrics are pretty
3:14:24
fast but like we had that two-day threed day backfill problem and I was like
3:14:30
that's not going to work right that's not like that's not going to work so then we move to this like uh we move to these monthly metrics right which
3:14:37
essentially reduces the volume again 30X because of the fact that now we and
3:14:44
but the thing is is like keep in mind this is not a monthly L aggregate that's the thing that I hope yall keep in mind
3:14:51

here and that's why this is a beautiful thing if it was a monthly aggregate this would be boring and like y'all would
3:14:58
like not like this and this would be kind of stupid but this is actually not a monthly aggregate so how this works
3:15:05
right is we have this month start and then we have this array of values so the
3:15:10
date is an index so in this case 34 is how many likes user 3 gave on July
3:15:21
1st and then three is July 2nd 3 July
3:15:26
3rd and then nine is July 31st so what this does is instead of
3:15:34
storing the date as a string right in
3:15:39
multiple rows we store the date as an offset or as an index so you do date
3:15:48
math based on the index the position of the position in the array is equal to
3:15:55
what what the date was so this is very similar like in a lot of ways this is similar to um yesterday's lab on uh date
3:16:04
lists like if you all remember like how we had that crazy like bits of ones and
3:16:09
zeros and the position based on like where if that bit was the first bit that meant it was for today and then the last
3:16:16
bit bit me it was like 30 days ago or whatever this is a similar concept except you are doing it
3:16:23
for um uh nonbinary things right you're doing it
3:16:29
for uh things that are either like a decimal number or uh like a count an
3:16:35
integer value I I talked about a lot of this stuff already where I said like the
3:16:40
first index is for the date at the beginning of uh the month and the last index is for the date at the end of the
3:16:46
month right and I was talking about how dimensional joins get weird if you want things to stay performant because you
3:16:52
don't want to like have to bring in a dimension in the middle of the month that was actually one of the things that like I was trying to figure out like

3:16:59
that was like one of the last things I was looking at at Facebook was like how to do that like how to actually make it
3:17:06
so that uh it the dimension gets assigned to the right value based on the
3:17:11
date and like based on the index in the array and then the sum happens the right way without exploding it and I couldn't
3:17:18
figure it out that was like I was like it's too crazy too crazy and I was like I and that's one of the beautiful things
3:17:24
about data is that like if you explain to people the constraints of what is like what are The Oddities and what are
3:17:30
the weirdness of things that can happen with this and then they're and they can accept that as like okay that is
3:17:35
inconsistency but like it's fine for like what I'm trying for the purposes of what I'm trying to use this for like
3:17:42
it's great right and so that can be um a a really powerful thing that will work
3:17:48
right so what this did was we were able to Ena
3:17:54
enable long-term analyses that were a lot faster so go to the next slide so
3:18:01
here's the impact from this new schema right so before if we wanted to do a
3:18:06
10-year backfill with that you know that intermediate layer that daily data uh a
3:18:11
10-year backfill would take about a week a week to maybe a little bit more
3:18:17
and once you got the data into this format instead of taking a week it took
3:18:23
a couple hours it took like three or four hours to run because it's just dramatically smaller dramatically
3:18:30
smaller data and um and it made it so we could cut up the data and look at every
3:18:35
look at all every possible Dimension Under the Sun like for the entire
3:18:40
history of Facebook as quick as we wanted and that's like I talking about this like unlock the decade long slow

3:18:47
bur analyses at Facebook which is like okay is there a certain type of user that is not doing a certain type of
3:18:54
thing over a long period of time but the decline is so slow that our alerts don't
3:18:59
pick it up and our alerts don't see it right so then what we were able to do is just like kind of look at all that stuff
3:19:05
and like I don't know if y'all have ever heard of like a root cause analysis or RCA so RCA is u a very powerful thing to
3:19:13
try to troubleshoot what was going wrong and what happened right and that is a big thing that this framework unlocked
3:19:21
and so uh yeah this is very powerful I felt very proud of building this out and
3:19:26
a lot of people in the analytics space at Facebook were like wow Zach you're actually doing stuff that is kind of innovative and new things here that are
3:19:33
kind of changing the company and so this is really powerful and uh I think I I think I have one more slide here so I I
3:19:40
I loaded this uh framework up with 50 50 core metrics and 15 core dimensions and
3:19:45
this uh Ena you know 10e analyses to happen in hours instead of weeks and uh
3:19:51
they didn't promote me they uh they said that that was not enough uh of an impact
3:19:58
to be a senior engineer and I was like very angry about that because I thought that that was because I thought I solved
3:20:04
some really in insane problems with this framework but yeah that was uh so the fact that Facebook did not recognize the
3:20:11
impact of this but it's crazy because then later on they did because like a couple of my teammates who stayed kept
3:20:17
working on this project right and uh they got this thing called discretional equity which is where uh Facebook will
3:20:24
give people like another big Equity Grant to keep them and maintain them and
3:20:29

like uh and so they they they made millions of dollars off of all my hard work and then I I and they're grateful
3:20:35
for it I'm still good friends with these people but I'm always like I tell them sometimes I'm like y'all owe me y'all owe me y'all owe me money dude like so
3:20:42
yeah um I think that is what we got for for the the the the lecture today
3:20:49
congrats on getting to the end of the day three lecture reduced facts are crazy right I'm excited for you to check
3:20:55
out the lab and how to minimize Shuffle and get in there and get handson if you're in the platform make sure to skip
3:21:01
to the next link so that you get the credit that you deserve