databricks

Fine-Tuning: Choosing the Right Cluster

LECTURE

Pick the Best Instance Types

This lecture guides you on selecting the best instance types by considering machine features, sizing, spot market, and shuffle partition strategies.

Have an Open Mind When Picking Machines

- For AWS. i3's aren't always the best. Explore m7gd and r7gd

  Enable caching if needed.
  - Graviton instances work well, try those first
  - M7gd and r7gd have better processors, similar (albeit smaller) local disk and much more stable spot markets than i-series
- For Azure, try the eav4, dav4 and f-series over L-series
  - The ACU is very useful
- GCP defaults are pretty good
- Usually don't need network optimized instance types some occasions they help with Photon

When choosing machine types for cloud workloads, don't just rely on familiar options. In AWS, besides i3, try m7gd and r7gd for better processors and more stable spot pricing. Enable caching if needed, and consider Graviton instances for cost-effective performance. In Azure, use eav4, dav4, or F-Series before the L series, and check the ACU metric to compare VM performance. For GCP, the recommended defaults usually perform well.

Network-optimized instances aren't often needed, but can help with Photon or bandwidth-heavy workloads. Always test different instance types and use spot markets carefully to balance savings and stability. Being flexible and open to alternatives is the best way to find the most suitable resources for your workloads

How to Choose the Right Machine Is Pretty Simple

- Just a series of IFTTT questions and rules of thumb!
  - Side note – if you 2x the cluster and it runs in 1/2 the time, it costs the same

- Rules of thumb
  - First run: set spark.sql.shuffle.partitions = 2x # of cores
  - Keep total memory available to the machine less than 128gb
  - Number of cores should be a ratio of 1 core to 128mb -> 2gb of *reads* (Some caveats may apply)
  - Avoid setting any other configs at first (don't carry over configs from legacy platforms unless absolutely necessary)

Choosing the right machine for your workload is straightforward when you follow a set of basic rules of thumb and simple "if this, then that" decisions. It's worth remembering that if you use a cluster that's twice as large and it finishes the job in half the time, the overall cost will be roughly the same—but you'll save valuable time. This means sometimes a larger cluster isn't more expensive if it delivers faster results, since time saved can be just as valuable as cost savings.

For your first run, a good rule is to set spark.sql.shuffle.partitions to double the number of cores in your cluster. Aim to keep the total available memory of any machine under 128 GB. When configuring cores, use a ratio of one core for every 128 MB to 200 GB of data reads, noting that this is a guideline and some exceptions may occur depending on the specifics of your workload.

It's best to avoid copying over configuration settings from other environments or previous projects when you start, unless there's a strong reason to do so. Begin with these basic guidelines, test your setup, and only adjust configurations as needed based on observed performance. This careful, incremental approach helps ensure you choose the best resources and achieve optimal performance and cost efficiency for your workload.

How to Choose the Right Machine Is Pretty Simple

- Rules of thumb

Scan parquet +details

Stages: 655.0

file sorting by size time	2 ms
cache writes size (uncompressed) total (min, med, max)	262.4 MiB (576.8 KiB, 1031.5 KiB, 1085.7 KiB)
time spent in the cache locality manager in milliseconds total (min, med, max)	37 ms (0 ms, 0 ms, 37 ms)
number of files read	1,027
filesystem read data size total (min, med, max)	279.8 MiB (620.5 KiB, 1101.1 KiB, 1155.4 KiB)
cache async file status fetch waiting time total (min, med, max)	0 ms (0 ms, 0 ms, 0 ms)
scan time total (min, med, max)	14.6 m (1.9 s, 3.2 s, 7.1 s)
filesystem read data size (sampled) total (min, med, max)	279.8 MiB (620.5 KiB, 1101.1 KiB, 1155.4 KiB)
filesystem read time (sampled) total (min, med, max)	12.4 m (1.7 s, 2.8 s, 5.5 s)
metadata time	8 ms
size of files read	236.7 GiB

The file system read data size, shown in the Spark UI, tells you how much data is being read from disk during a job. Checking this value helps you decide if your cluster's size and setup are suitable, and can guide any needed adjustments for efficiency or performance.

What You Care about with the Instance Type

- Core to Ram ratio
- Processor type
- Local vs remote storage
- Storage medium

Cloud	Family	Core:Ram	Processor	Storage
AWS	c5	1core:2gb	Intel Cascade 3.6 GHz	(d) Local NVME
Azure	f-series	1core:2gb	Intel Xeon 2.4 GHz	Local SSD
GCP	n2-highcpu	1core:1gb	Intel Cascade 3.4 GHz	Local SSD

When selecting instance types, the key factors to focus on are the core to RAM ratio, processor type, local versus remote storage, and the storage medium. These elements—such as how much memory you have per core, the speed and generation of the processor, and whether your storage is fast local NVMe or slower remote disks—will have the biggest impact on your query performance. For example, in AWS, C5 family instances offer a one core to two gig RAM ratio with Intel processors and local NVMe storage. Similar details apply to Azure and GCP instance choices. Prioritizing these basic hardware specs ensures you're set up for efficient and fast query execution.

Sizing a Driver

- Leave it the same size as your worker unless you care about being the absolute cheapest - dont make things more complicated than they need to be.
- Driver typically do very little work in a Spark application. Using a 4-8 core 16-32gb ram driver should be fine for most workloads
- Large commits to delta tables use more memory.
- This suggestion is voided when:
  - Running many streams/concurrent jobs on the same machine
  - Commiting a very large (100k+ files) amount of data to a delta table
  - Collecting large amount of data to the driver to use in Pandas/R

When it comes to sizing the driver compared to the workers in a Spark cluster, it's usually simplest just to match the driver's size to the workers. This avoids unnecessary complexity and works well for almost all workloads, since the driver handles far less work than the workers in typical Spark applications. A driver with 4–8 cores and 16–32 GB of RAM will be sufficient for most scenarios.

If you want to minimize costs as much as possible, you could consider a slightly smaller driver, but generally there's no need to complicate matters. However, keep in mind that if you're committing large amounts of data to Delta tables, the driver will require more memory. These guidelines no longer apply if you are running many streams or concurrent jobs on the same machine, or if you are handling especially large commits—like writing 100,000 or more files to a Delta table, or collecting large amounts of data to the driver for use with pandas or R. In those exceptional cases, you'll need a larger, more carefully sized driver to avoid running into memory issues

Spot Market Considerations

- The spot market is a great way to save money on infrastructure.
- Each instance type has a different level availability and price savings in each region.
- Example – i3s aren't great, r5d's look a lot better.

Instance Type	vCPU	Memory GiB	Savings over On-Demand*	Frequency of interruption
i3.xlarge	4	30.5	70%	>20%
i3.2xlarge	8	61	70%	>20%
i3.4xlarge	16	122	70%	>20%
r5d.large	2	16	85%	<5%
r5d.xlarge	4	32	85%	<5%
r5d.2xlarge	8	64	69%	<5%
r5d.4xlarge	16	128	80%	5-10%

Reference: https://aws.amazon.com/ec2/spot/instance-advisor/

When using spot VMs, it's important to know that each instance type offers different levels of availability and price savings. Spot markets can save you a significant amount on infrastructure, but the savings and reliability vary by instance. For example, while i3 instances might offer about 70% savings compared to on-demand pricing, you may face higher interruption rates, making them less attractive for some workloads. On the other hand, r5d instances often provide even better savings—up to 85%—and a lower frequency of interruption, sometimes less than 5%. Choosing instance types like R5d large or extra large can reduce your costs significantly while minimizing the risk of interruptions from the cloud provider. So, compare both the potential savings and the frequency of interruption when selecting spot VMs for your cluster.

IFTTT - Step 1

Want to use Photon?

No:
Next slide

Yes:

Cloud	Family
AWS	m6gd/r6gd/i4i m7gd/r7gd
Azure	Edsv4
GCP	n2-highmem n2-standard

When deciding how to choose your cluster configuration, start by asking whether you want to use Photon. If you don't plan to use Photon, proceed to the next set of considerations for your VM selection. If you do want to use Photon, you can begin with the list of recommended VM types that are optimized for Photon as your starting point. This simple decision process helps guide your machine choices for efficient and effective cluster setup.

IFTTT - Step 2

Is your job an ETL job that uses joins/windows/groupbys/aggregations

No:

Cloud	Family
AWS	c7g/c6g
Azure	fsv2
GCP	e2-highcpu

Yes:

Cloud	Family
AWS	c7gd/c6gd
Azure	fsv2
GCP	n2-highcpu

If you're not using Photon, the next question is whether your job is an ETL workload involving joins, windows, group by, or aggregations. If the answer is no, there are specific instance suggestions best suited for lighter or different workloads. If the answer is yes, there are different recommendations focused on supporting those more complex data operations. This approach helps you align your machine type to the demands of your specific job.

IFTTT - Step 3

Run the job with the instance type, follow our rules of thumb and go to the SQL UI of the longest running query - do you see spill?

HashAggregate +details	
spill size	0.0 B
time in aggregation build total (min, med, max)	15.7 m (2.1 s, 3.5 s, 7.4 s)
peak memory total (min, med, max)	64.3 MiB (256.0 KiB, 256.0 KiB, 256.0 KiB)
passthrough output rows	0
avg hash probe bucket list iters	0
rows output	514

Tasks: Succeeded/Total	Input	Output	Shuffle Read ▾	Shuffle Write
1/1			156.0 KiB	
1/1			136.8 KiB	
1/1			132.1 KiB	
1/1			131.9 KiB	

No:
Stop this is good enough

Yes:
Set spark.sql.shuffle.partitions to the largest shuffle read stage / 200mb
spark.sql.shuffle.partitions=auto

FYI: Spill is much less impactful when using Photon

Once you've chosen an instance type and set up your cluster based on the rules of thumb, run your job and then check the Spark UI for the longest running query. Pay particular attention to whether there is any spill occurring. If there's no spill, your setup is sufficient. If you do see spill, that's your signal to make further adjustments. Start by setting spark.sql.shuffle.partitions to match the size of the largest shuffle read stage—if everything fits within 200 MB, use that as your reference point. Then set your shuffle partitions to auto, letting Spark manage partitioning to optimize performance and reduce spill. This approach lets you tune your environment step by step, addressing problems only as they appear.

IFTTT – Step 4

Run the job with the updated shuffle partitions – do you still see spill?

No:
Stop this is good enough

Yes:

Cloud	Family
AWS	m7gd
Azure	dav4/dasv4
GCP	n2-standard

After updating your shuffle partitions, check again in the Spark UI to see if there is still spill. If there's no spill, your cluster and settings are well-tuned and you're good to go. If you continue to see spill, apply the next set of suggestions, then rerun the job.

IFTTT – Step 5

Run the job with the updated instance type – do you still see spill?

No:
Stop this is good enough

Yes:

Cloud	Family
AWS	r7gd/r6gd
Azure	Edsv4
GCP	n2-highmen

Keep repeating this process—adjusting settings and observing the results—until you eliminate spill and your job runs efficiently. This step-by-step tuning ensures your configuration is precisely matched to your workload's needs.

Reminder on Shuffle Partitions

spark.sql.shuffle.partitions = auto

OR

Go to stage UI, find the largest shuffle read size, divide that by 200mb

Tasks: Succeeded/Total	Input	Output	Shuffle Read ▾	Shuffle Write
600/600		368.9 MiB	743.3 MiB	
600/600		368.9 MiB	743.3 MiB	
600/600		368.9 MiB	743.3 MiB	

For shuffle partitions, keep it straightforward. You can set shuffle partitions to auto and let Spark handle the adjustments for you. If you want to set it manually, go to the stage UI in Spark and look for the largest shuffle read size. Divide that size by 200 to determine the number of shuffle partitions you should set. This approach helps you match partition size to your workload and keeps things efficient.

Don’t Forget to Double Check the Event Log!

Spot failures happen. They slow things down. We know. Don’t forget to double check the event log. It’s probably the first thing you should do.

Configuration | Notebooks (3) | Libraries | Event log | Spark UI | Driver logs | Metrics | Apps | Spark

Filter by Event Type...

Event Type	Time	Message
RESIZING	2022-03-10 10:17:19 EST	Autoscaling from 7 down to 1 workers.
RESIZING	2022-03-10 10:07:19 EST	Autoscaling from 11 down to 7 workers.
RESIZING	2022-03-10 09:57:19 EST	Autoscaling from 13 down to 11 workers.
RESIZING	2022-03-10 09:47:19 EST	Autoscaling from 14 down to 13 workers.
UPSIZE_COMPLETED	2022-03-10 07:59:22 EST	Cluster upsize to 14 nodes completed.
RESIZING	2022-03-10 07:58:54 EST	Autoscaling from 13 up to 14 workers.
RESIZING	2022-03-10 07:58:24 EST	Autoscaling from 14 down to 13 workers.
UPSIZE_COMPLETED	2022-03-10 07:05:12 EST	Cluster upsize to 14 nodes completed.
RESIZING	2022-03-10 07:01:19 EST	Autoscaling from 9 up to 14 workers.
UPSIZE_COMPLETED	2022-03-10 07:01:10 EST	Cluster upsize to 9 nodes completed.
RESIZING	2022-03-10 06:58:24 EST	Autoscaling from 1 up to 9 workers.
RESIZING	2022-03-10 03:58:34 EST	Autoscaling from 7 down to 1 workers.
RESIZING	2022-03-10 03:48:34 EST	Autoscaling from 11 down to 7 workers.

Don’t forget about the event log. The event log is extremely useful for monitoring your cluster’s behavior and diagnosing issues, especially when it comes to spot failures. It can show you important details like how the cluster is resizing during autoscaling—from higher to lower worker counts and vice versa. By reviewing the event log, you can determine how many workers you actually need, what types of VMs are working best, and if you’re encountering any errors or interruptions. The event log should always be the first place you look when tuning or troubleshooting your cluster.

Thank you for completing this lesson and continuing your journey to develop your skills with us.