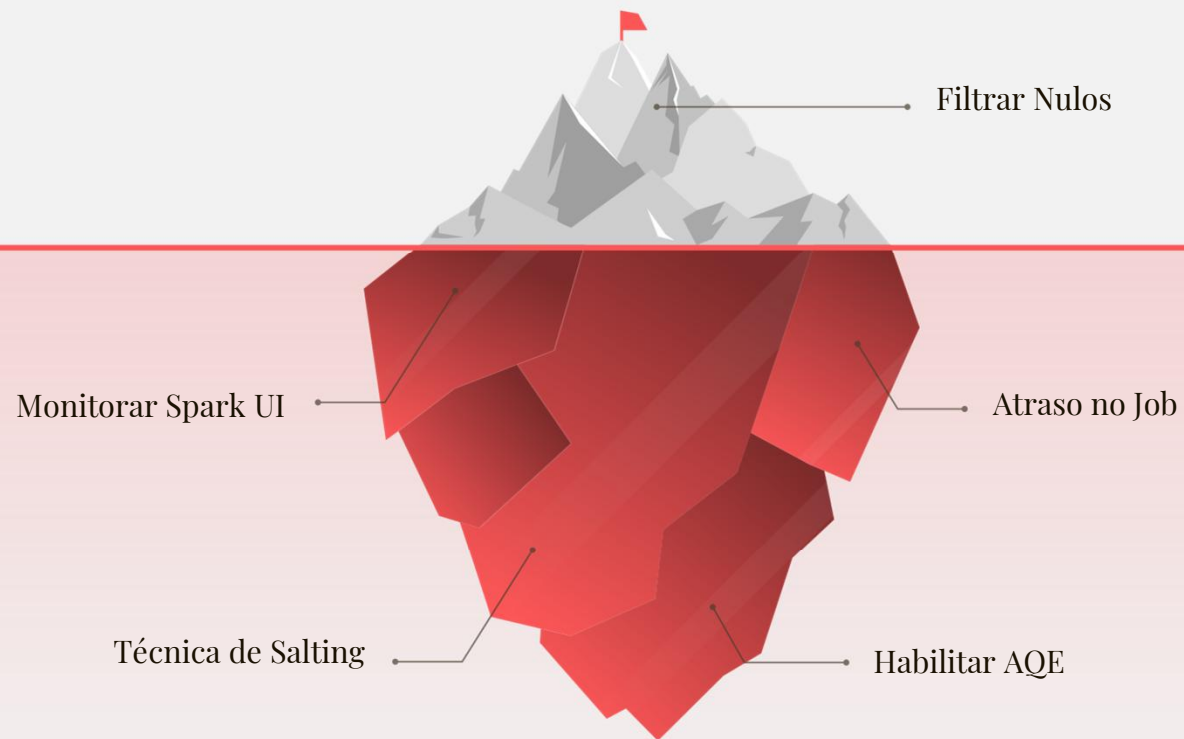


Identificando e Mitigando o Impacto do Data Skew



Técnica de Salting

Dataset de exemplo (dados com skew)

Tabela FATOS (muitos registros com a mesma chave)

id_cliente	valor
1	10
1	20
1	30
1	40
1	50
1	60
2	70
3	80
4	90
5	100

Distribuída em 4 partições (original)

Partição 1

id_cliente	valor
1	10
1	20
1	30
1	40
1	50
1	60

Partição 2

id_cliente	valor
2	70
3	80

Partição 3

id_cliente	valor
4	90

Partição 4

id_cliente	valor
5	100

Problema: A chave "id_cliente = 1" está concentrada em uma partição. Isso causa Data Skew → uma tarefa muito lenta e outras ociosas.

O que é Data Skew?

Quando algumas chaves têm muito mais dados que outras, causando desequilíbrio no processamento.

1 Como funciona o Salting

1 Adiciona-se um "sal" (número aleatório) às chaves mais frequentes

id_cliente	salt	valor
1	0	10
1	1	50
1	2	30
1	3	40
1	0	50
1	1	60
2	0	70
3	0	80
4	0	90
5	0	100

Exemplo: usamos 4 "sais" (0, 1, 2, 3) apenas para a chave = 1.

2 Reparticiona os dados (shuffle) pela chave + sal

Partição 1

id_cliente	salt	valor
1	0	10
1	0	50
2	0	70
4	0	90

Partição 2

id_cliente	salt	valor
1	1	20
1	1	60
3	0	80
5	0	100

Partição 3

id_cliente	salt	valor
1	2	30

Partição 4

id_cliente	salt	valor
1	3	40

3 Cada partição processa seu conjunto de forma paralela

- Task 1 (para id=1, sal=0 e outras chaves)
- Task 2 (para id=1, sal=1 e outras chaves)
- Task 3 (para id=1, sal=2)
- Task 4 (para id=1, sal=3)

4 Remove o sal e agrega os resultados finais

id_cliente	total
1	210
2	70
3	80
4	90
5	100

2 Comparação: Sem Salting vs Com Salting

Sem Salting (com skew)

Partição 1	Partição 2	Partição 3	Partição 4
 ~90% dos dados	 ~5%	 ~3%	 ~2%

Uma tarefa demora muito e as outras ficam ociosas. Tempo total de execução = tempo da tarefa mais lenta.

Com Salting (balanceado)

Partição 1	Partição 2	Partição 3	Partição 4
 ~25%	 ~25%	 ~25%	 ~25%

Carga distribuída entre as partições. Execução mais paralela e muito mais rápida.

Quando usar?

- Quando há chaves muito frequentes (ex: um id com milhões de linhas)
- Joins, groupBy, aggregate com forte desbalanceamento
- Quando o shuffle está muito desigual

3 Resumo rápido

Aspecto	Sem Salting	Com Salting
Distribuição dos dados	Desbalanceada (skew)	Balanceada
Shuffle	Desbalanceado	Balanceado (chave + sal)
Paralelismo	Baixo (algumas tarefas lentas)	Alto (tarefas similares)
Tempo de execução	Mais lento	Mais rápido
Complexidade	Simples	Um pouco maior (adicionar e remover sal)
Resultado final	Correto	Correto (após remover sal e agregar)

★ Dica de ouro

Salting quebra chaves "pesadas" em várias subchaves temporárias para distribuir a carga e depois reagrupa no final.

Escolha a quantidade de sal (ex: 4, 8, 16, ...) com base no tamanho do skew.

Regra prática: comece com 4 ou 8 e ajuste conforme necessário.

Exemplo em PySpark

```
1. Adicionar sal (ex: 4 sais)
from pyspark.sql import functions as F
NUM_SALT = 4

df_salted = df.withColumn(
    "salt",
    F.when(F.col("id_cliente") == 1,
            (F.rand(NUM_SALT).cast("int"))
            .otherwise(F.lit(0)))
)

# Nova chave composta
df_salted = df_salted.withColumn(
    "id_salted",
    F.concat_ws("-", F.col("id_cliente"), F.col("salt"))
)
```

```
2. Reparticionar (shuffle pela chave + sal)
df_shuffled = df_salted.repartition("id_salted")
```

```
3. Agregar e remover sal
from pyspark.sql import functions as F

resultado = (df_shuffled
    .groupBy("id_cliente")
    .agg(F.sum("valor").alias("total"))
)
```

Lembre-se: Salting não resolve todos os problemas. Use apenas quando o skew realmente impactar o desempenho. Monitore suas tarefas no Spark UI para validar os resultados!