**Slide mở đầu – Giới thiệu vấn đề (mở rộng)**

**Tiêu đề gợi ý:** *“Vì sao cần tối ưu hiệu năng trong Spark Streaming?”*

**Nội dung chính:**

1. **Bối cảnh thực tế**
   * Spark Streaming thường được dùng trong: giám sát IoT, phát hiện gian lận giao dịch, xử lý log hệ thống, phân tích clickstream…
   * Các ứng dụng này yêu cầu **xử lý real-time** với độ trễ thấp (tính bằng giây).
2. **Vấn đề đặt ra**
   * Hiệu năng Spark Streaming **không tự nhiên mà có**, nó phụ thuộc vào cách ta cấu hình và lập trình.
   * Nếu xử lý không kịp nhịp dữ liệu đến → backlog tăng, **Scheduling Delay phình to**, cảnh báo/report ra chậm → ảnh hưởng trực tiếp đến nghiệp vụ.
3. **Yếu tố trọng tâm sẽ phân tích**
   * **Batch Interval**: quyết định nhịp gom và xử lý batch.
   * **Parallelism (số partition)**: quyết định mức độ song song.
4. **Dẫn dắt sang phần chính**
   * Ngoài 2 yếu tố cốt lõi, còn có các yếu tố khác cũng ảnh hưởng mạnh:
     + **I/O**: tốc độ đọc/ghi nguồn dữ liệu.
     + **Skew dữ liệu**: phân bố không đều làm một số task kéo dài bất thường.
     + **Cấu hình runtime**: GC, serialization, backpressure.
   * Hai yếu tố trọng tâm (Batch Interval, Parallelism) sẽ được đi sâu trước, vì chúng tác động rõ ràng và dễ quan sát ngay trên Spark Web UI.

**Slide 2 – Batch Interval (chi tiết)**

**Tiêu đề gợi ý:** *“Batch Interval – Nhịp tim của Spark Streaming”*

**Nội dung chính:**

1. **Định nghĩa**
   * Là khoảng thời gian gom dữ liệu thành 1 batch để xử lý (ví dụ: BI = 2s → mỗi 2 giây tạo 1 batch).
2. **Ảnh hưởng**
   * **Quá ngắn**: latency nhỏ, nhưng dễ backlog nếu xử lý không kịp.
   * **Quá dài**: hệ thống ổn định hơn, nhưng độ trễ end-to-end tăng.
3. **Cách chọn hợp lý**
   * Quy tắc: BI ≥ p95 Processing Time mong muốn.
   * Bắt đầu từ 1–2s rồi điều chỉnh sau khi đo.
4. **Ví dụ minh họa**
   * Biểu đồ đơn giản: trục X = Batch Interval, trục Y = Latency → đường cong thể hiện BI ngắn quá hoặc dài quá đều bất lợi.

**Slide 3 – Parallelism / Số partition**

**Tiêu đề gợi ý:** *“Parallelism – Khả năng song song hóa”*

**Nội dung chính:**

1. **Định nghĩa**
   * Partition = đơn vị nhỏ của RDD.
   * Mỗi partition → 1 task → có thể chạy song song trên 1 core.
2. **Ảnh hưởng**
   * **Partition ít**: CPU không dùng hết, job chạy chậm.
   * **Partition nhiều vừa đủ**: tận dụng tài nguyên, giảm thời gian xử lý.
3. **Quy tắc chọn**
   * Số partition ≈ 2–4 × số core khả dụng cho stage nặng.
4. **Ví dụ minh họa**
   * Chạy Spark ở local[4] với 8 partitions → 8 task chạy song song, tận dụng hết 4 core.

**Slide 4 – Cạm bẫy Partition**

**Tiêu đề gợi ý:** *“Nguy cơ khi điều chỉnh partitions”*

**Nội dung chính:**

1. **Overhead khi quá nhiều partition**
   * Quá vụn → nhiều task nhỏ → overhead scheduling, shuffle tăng.
2. **Skew dữ liệu**
   * Một số partition rất lớn (key hot) → 1 task kéo dài → toàn batch chậm.
3. **Giải pháp**
   * **Salting** key để phân tán đều.
   * **Map-side combine**: gộp dữ liệu trước khi shuffle.
   * Tùy chỉnh partitioner nếu cần.

**Slide 5 – I/O và External System**

**Tiêu đề gợi ý:** *“Khi hiệu năng bị bóp nghẹt bởi I/O”*

**Nội dung chính:**

1. **Ảnh hưởng**
   * Xử lý dữ liệu không chỉ CPU-bound.
   * Nguồn đọc (Kafka, Socket) chậm hoặc sink (DB, file) nghẽn → Processing Time tăng.
2. **Ví dụ**
   * Ghi từng record xuống DB → batch xử lý kéo dài bất thường.
3. **Giải pháp**
   * Batch I/O (ghi theo lô).
   * Async I/O.
   * Giới hạn throughput nguồn vào để tránh quá tải.

**Slide 6 – Code & Cấu hình runtime**

**Tiêu đề gợi ý:** *“Tối ưu code và runtime”*

**Nội dung chính:**

1. **Thực hành tốt**
   * Tránh collect() nếu không cần thiết.
   * Dùng mapPartitions thay vì map nếu có logic setup tốn kém.
   * Cache/Broadcast khi tái sử dụng nhiều lần.
2. **Runtime tuning**
   * Dùng **Kryo Serialization** cho object phức tạp.
   * Quản lý GC: tránh tạo nhiều object rác.

**Slide 7 – Các thông số phụ trợ**

**Tiêu đề gợi ý:** *“Thông số Spark quan trọng ảnh hưởng hiệu năng”*

**Nội dung chính:**

1. **spark.default.parallelism**
   * Ảnh hưởng mặc định số partition khi tạo RDD.
2. **spark.streaming.backpressure.enabled (DStreams)**
   * Tự động điều tiết tốc độ đọc nguồn vào, tránh quá tải.
3. **spark.streaming.kafka.maxRatePerPartition**
   * Giới hạn tốc độ đọc Kafka per partition, cân bằng tốc độ và hiệu năng.

**Slide 8 – Kết nối sang phần sau**

**Tiêu đề gợi ý:** *“Từ yếu tố → đo lường trên Spark UI”*

**Nội dung chính:**

* Các yếu tố ảnh hưởng chính:
  + **Batch Interval**
  + **Parallelism (partition)**
  + **I/O**
  + **Skew dữ liệu**
  + **Code & cấu hình runtime**
* Nhưng để đo lường tác động thực tế, ta cần theo dõi các chỉ số trên Spark Web UI:
  + **Processing Time**
  + **Scheduling Delay**
* Đây sẽ là nội dung phần tiếp theo.

**Slide 9 – Giới thiệu khái niệm đo lường trên Spark UI**

**Tiêu đề gợi ý:** *“Từ yếu tố ảnh hưởng → chỉ số đo lường trên Spark UI”*

**Nội dung chính:**

* Các yếu tố như Batch Interval, Partition, I/O, Skew chỉ là **nguyên nhân**.
* Cần một cách **đo lường tác động** để biết hệ thống chạy ổn không.
* Spark Web UI cung cấp 2 chỉ số quan trọng:
  + **Processing Time**
  + **Scheduling Delay**
* Đây chính là “thước đo sức khỏe” của ứng dụng Streaming.
* Dẫn dắt: ta sẽ lần lượt tìm hiểu ý nghĩa của 2 chỉ số này.

**Slide 10 – Processing Time**

**Tiêu đề gợi ý:** *“Processing Time – thời gian xử lý batch”*

**Nội dung chính:**

1. **Định nghĩa**
   * Thời gian Spark cần để xử lý toàn bộ dữ liệu trong một batch.
2. **Ý nghĩa**
   * Cho biết pipeline nhanh/chậm đến đâu.
   * Nếu **Processing Time > Batch Interval** → backlog bắt đầu hình thành.
3. **Ví dụ**
   * BI = 2s, Processing Time = 1.5s → chạy ổn định.
   * BI = 2s, Processing Time = 3s → không kịp nhịp, batch chồng dồn.

**Slide 11 – Scheduling Delay**

**Tiêu đề gợi ý:** *“Scheduling Delay – thời gian batch phải chờ”*

**Nội dung chính:**

1. **Định nghĩa**
   * Khoảng thời gian một batch **xếp hàng đợi** trước khi được xử lý.
2. **Ý nghĩa**
   * Thể hiện khả năng “đuổi kịp” dòng dữ liệu.
   * Nếu **Delay cao liên tục** → hệ thống quá tải hoặc cấu hình chưa hợp lý.
3. **Ví dụ**
   * BI = 2s, Processing Time = 3s → batch sau phải đợi 1s ⇒ Scheduling Delay = 1s.

**Slide 12 – Quan hệ giữa Processing Time & Scheduling Delay**

**Tiêu đề gợi ý:** *“Processing Time và Scheduling Delay liên kết chặt chẽ”*

**Nội dung chính:**

* Công thức: **Total Delay = Processing Time + Scheduling Delay**.
* Nếu **Processing Time ≤ Batch Interval** → Scheduling Delay ≈ 0 (ổn định).
* Nếu **Processing Time > Batch Interval** → Scheduling Delay tăng dần, backlog phình.
* Quan sát trên UI: đường biểu đồ Delay sẽ dốc lên theo thời gian nếu không tối ưu.

**Slide 13 – Ý nghĩa thực tế & kết nối demo**

**Tiêu đề gợi ý:** *“Ứng dụng trong giám sát và tối ưu”*

**Nội dung chính:**

* 2 chỉ số này là **trọng tâm để giám sát** tab Streaming.
* Dùng để chẩn đoán:
  + Processing Time cao → tối ưu code, tăng partition.
  + Scheduling Delay cao → cân nhắc tăng Batch Interval, scale cluster, backpressure.
* Kết nối demo:
  + Thêm sleep → Processing Time tăng → Delay xuất hiện.
  + Tăng partition → Processing Time giảm → Delay giảm.

**Slide 14 – Biểu đồ minh họa (Visualization trên Spark UI)**

**Tiêu đề gợi ý:** *“Nhìn chỉ số qua biểu đồ trên Spark UI”*

**Nội dung chính:**

* Tab **Streaming** trên Spark UI hiển thị biểu đồ trend:
  + **Processing Time** theo thời gian.
  + **Scheduling Delay** theo thời gian.
* Cách đọc biểu đồ:
  + Đường Processing Time ổn định dưới Batch Interval → hệ thống khỏe.
  + Đường Scheduling Delay “leo thang” → dấu hiệu backlog tích tụ.
* Ý nghĩa: dễ dàng giám sát và phát hiện sớm vấn đề, không cần chờ lỗi xảy ra.

**Slide 15 – Kịch bản minh họa bằng số liệu**

**Tiêu đề gợi ý:** *“Ví dụ thực tế: Delay xuất hiện như thế nào?”*

**Nội dung chính:**

* Giả sử: Batch Interval = 2s.
* **Case 1:** Processing Time = 1.5s → Scheduling Delay ≈ 0.
* **Case 2:** Processing Time = 2.5s → Scheduling Delay bắt đầu = 0.5s.
* **Case 3:** Nhiều batch liên tục > 2s → Delay cộng dồn thành hàng giây, hàng phút.
* Minh họa bằng sơ đồ thời gian: batch đến liên tục, batch trước chưa xong thì batch sau phải xếp hàng.

**Slide 16 – Nguyên nhân thường gặp khi Scheduling Delay cao**

**Tiêu đề gợi ý:** *“Tại sao Scheduling Delay cao liên tục?”*

**Nội dung chính:**

1. Batch Interval quá ngắn → không kịp xử lý.
2. Thiếu tài nguyên (CPU, partition ít).
3. Code xử lý nặng hoặc I/O chậm.
4. Skew dữ liệu (1 task kéo dài bất thường).
5. Input rate quá nhanh → hệ thống không theo kịp.

**Slide 17 – Cách khắc phục Scheduling Delay**

**Tiêu đề gợi ý:** *“Giải pháp giảm Scheduling Delay”*

**Nội dung chính:**

* Tăng **Batch Interval** nếu pipeline quá nặng.
* Tăng **số partition / core** để xử lý song song hơn.
* Tối ưu code (tránh collect(), dùng mapPartitions).
* Chống **skew dữ liệu** (salting, map-side combine).
* Điều tiết nguồn (backpressure, giới hạn Kafka rate).

**Slide 18 – Kết nối sang phần Demo**

**Tiêu đề gợi ý:** *“Từ lý thuyết → minh họa bằng Demo”*

**Nội dung chính:**

* Trong demo sẽ:
  + Chạy ứng dụng baseline (Delay ≈ 0).
  + Thêm sleep để Processing Time > Batch Interval → Delay xuất hiện.
  + Tăng partition để giảm Processing Time → Delay hạ xuống.
* Mục tiêu: minh họa rõ quan hệ giữa **yếu tố cấu hình** và **chỉ số đo lường**.

Ok Don, mình nối tiếp **từ Slide 18** và kết thúc bằng **bài tập điền mã** giống demo (có chỗ khuyết). Ngắn – rõ – cầm là làm được.

**Phần 4: Demo & Thực hành (tiếp từ Slide 18)**

**Slide 18 – Kịch bản demo tổng quan**

* Pha 1: **Baseline** (BI=2s, partitions=4) → Delay ≈ 0
* Pha 2: **Cố tình chậm** (sleep) → Processing Time > BI → Delay tăng
* Pha 3: **Tối ưu** (tăng partitions/cores) → Processing Time ↓ → Delay ↓
* Mục tiêu: thấy được tác động cấu hình → chỉ số trên Spark UI

**Slide 19 – Chuẩn bị môi trường**

* Spark 2.4–3.x; mở 2 terminal
* **Nguồn socket (9999):**
* while true; do
* echo "word$((RANDOM%50)) word$((RANDOM%50)) word$((RANDOM%50))"
* sleep 0.01
* done | nc -lk 9999
* Chạy app bằng spark-submit --master local[8] ...
* Mở UI: http://localhost:4040

**Slide 20 – Pha 1: Baseline**

* Kỳ vọng UI (tab Streaming):
  + Processing Time < 2s
  + Scheduling Delay ≈ 0
  + Queued Batches = 0
* Check Jobs/Stages: số task ≈ số partition

**Slide 21 – Pha 2: Cố tình chậm**

* Thêm time.sleep(20ms) ở map
* Kỳ vọng: Processing Time > 2s ⇒ Delay bắt đầu tăng; Queued Batches > 0
* Ghi ảnh màn hình 1–2 batch để so sánh

**Slide 22 – Pha 3: Tối ưu bằng Parallelism**

* Tăng repartition + reduceByKey(numPartitions) (ví dụ 8)
* (Nếu máy đủ) tăng local[8] → tận dụng CPU
* Kỳ vọng: Processing Time ↓ rõ; Delay ↓ hoặc về ≈ 0

**Slide 23 – Đọc tab Streaming đúng trọng tâm**

* Cột cần nhìn: **Processing Time**, **Scheduling Delay**, **Total Delay**, **Queued Batches**
* So sánh Processing Time với **Batch Interval** bạn cấu hình

**Slide 24 – Drill-down: Jobs → Stages → Tasks**

* Số task ≈ số partition
* Nếu 1 task cực lâu → nghi **skew**/I/O
* Dùng trang Stages để tìm bottleneck

**Slide 25 – Lưu ý khi trình diễn**

* Port 9999 rảnh; nếu UI 4040 bận → spark.ui.port
* Nếu socket lỗi, có ảnh chụp UI để backup
* Chỉ thay **1 biến mỗi lần** (sleep, partitions) để thấy tác động rõ

**Slide 26 – Tương tác lớp: câu hỏi tình huống**

* “**Nếu Scheduling Delay liên tục cao**, nguyên nhân & khắc phục?”
  + Nguyên nhân: BI quá ngắn, thiếu CPU/partition, operator nặng/I/O chậm, skew, input quá nhanh
  + Khắc phục: tăng BI, tăng partitions/cores, tối ưu code/I/O, salting chống skew, backpressure/giới hạn input

**Phần 5: Bài tập điền mã (giống demo, có chỗ khuyết)**

**Slide 27 – Yêu cầu bài tập (Level 1)**

* Batch Interval = **2s**
* Thêm chậm **15ms/record**
* **repartition(4)** trước xử lý nặng
* reduceByKey chạy với **4 partitions**
* In **5** kết quả/ batch
* Chạy được với spark-submit --master local[4 or 8]

**Slide 28 – Mã bài tập (điền chỗ khuyết)**

# exercise\_fill\_gaps.py

from pyspark import SparkConf, SparkContext

from pyspark.streaming import StreamingContext

import time

SLEEP\_MS = 15

def slow\_map(x):

# TODO: thêm chậm theo SLEEP\_MS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return x

conf = (SparkConf()

.setAppName("Exercise-FillGaps")

.set("spark.ui.port", "4040")

.set("spark.default.parallelism", "4"))

sc = SparkContext(conf=conf)

# TODO: Tạo StreamingContext với Batch Interval = 2 giây

ssc = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Nguồn dữ liệu socket

lines = ssc.socketTextStream("localhost", 9999)

# TODO: tăng parallelism trước khi xử lý nặng

faster = lines.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.map(slow\_map)

# Tách từ và đếm

pairs = faster.flatMap(lambda line: line.split()).map(lambda w: (w, 1))

# TODO: reduceByKey với 4 partitions

counts = pairs.reduceByKey(lambda a, b: a + b, numPartitions=\_\_\_\_)

# TODO: in ra 5 dòng/ batch

counts.\_\_\_\_\_\_\_\_(5)

ssc.start()

ssc.awaitTermination()

**Gợi ý điền (đưa cho trợ giảng/nhóm):**

* time.sleep(SLEEP\_MS/1000.0)
* StreamingContext(sc, 2)
* repartition(4)
* 4
* pprint

**Slide 29 – Kỳ vọng & tiêu chí chấm**

* App chạy, UI xuất hiện tab **Streaming**
* Processing Time có tăng do slow\_map
* Đúng vị trí repartition(4) và reduceByKey(..., 4)
* In được 5 dòng/ batch
* Không lỗi socket/port

**Slide 30 – Bài tập nâng cao (Level 2, tuỳ chọn)**

**Mục tiêu:** nhận diện hot key (skew) và **salting** để giảm task “đỏ”.  
**Yêu cầu:** thêm muối 0..3 cho key "word0" trước reduceByKey, sau đó bỏ muối và gộp lại.

# exercise\_skew\_fill\_gaps.py

from pyspark import SparkConf, SparkContext

from pyspark.streaming import StreamingContext

import time, random

HOT = "word0"; SALT\_N = 4; SLEEP\_MS = 10

def slow\_map(x):

time.sleep(SLEEP\_MS/1000.0); return x

conf = (SparkConf().setAppName("Exercise-Skew").set("spark.ui.port","4040"))

sc = SparkContext(conf=conf)

ssc = StreamingContext(sc, 2)

lines = ssc.socketTextStream("localhost", 9999).repartition(8).map(slow\_map)

words = lines.flatMap(lambda l: l.split())

# TODO: Nếu w == HOT, thêm salt ngẫu nhiên 0..SALT\_N-1

salted = words.map(lambda w: ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , 1 ))

# TODO: reduce theo key đã salted với 8 partitions

part = salted.reduceByKey(lambda a,b: a+b, numPartitions=\_\_\_\_)

# TODO: bỏ muối, gộp lại theo key gốc

unsalted = part.map(lambda kv: ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , kv[1])) \

.reduceByKey(lambda a,b: a+b, numPartitions=8)

unsalted.pprint(5)

ssc.start(); ssc.awaitTermination()

**Gợi ý điền:**

* (w if w != HOT else f"{w}:{random.randint(0, SALT\_N-1)}")
* 8
* kv[0].split(":")[0]

**Slide 31 – Tổng kết & chuyển giao**

* Bạn vừa:
  1. Thấy **Processing Time/Delay** thay đổi theo cấu hình.
  2. Tối ưu bằng **partitions/cores**.
  3. Thực hành **điền mã** để chạy được app và kiểm tra trên UI.
* Gợi ý tự học: thử thay BI=1s, partitions=16; bật backpressure với nguồn nhanh.