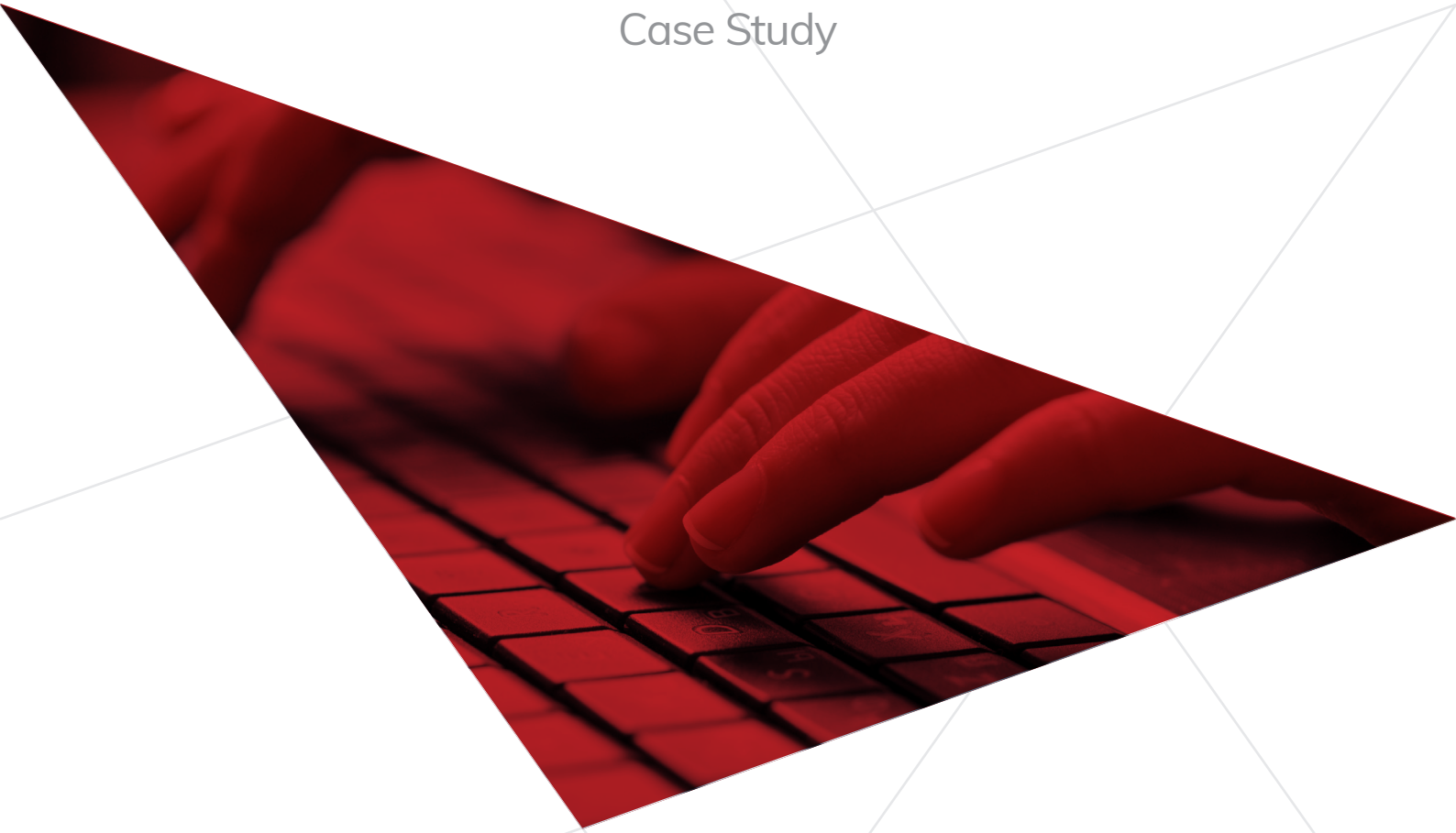
scalac

best scala hAkkers!

TAPAD

Case Study



Category:	Big Data, Personalized Advertising
Tags:	Streaming Data, Interactive Analysis, Reliable efficiency, Low latency
Industry:	Adtech
Website:	tapad.com

Challenge

As an Adtech organization, Tapad had been using an Apache Spark cluster as an important element of the ETL process to distribute large data sets to be processed and distributed to 3rd parties.

Our primary goal was to develop integrations using the above-mentioned Apache Spark. While working on the platform we found issues with Spark resource management.

We were also required to provide a different flow of integrations with various 3rd party services.

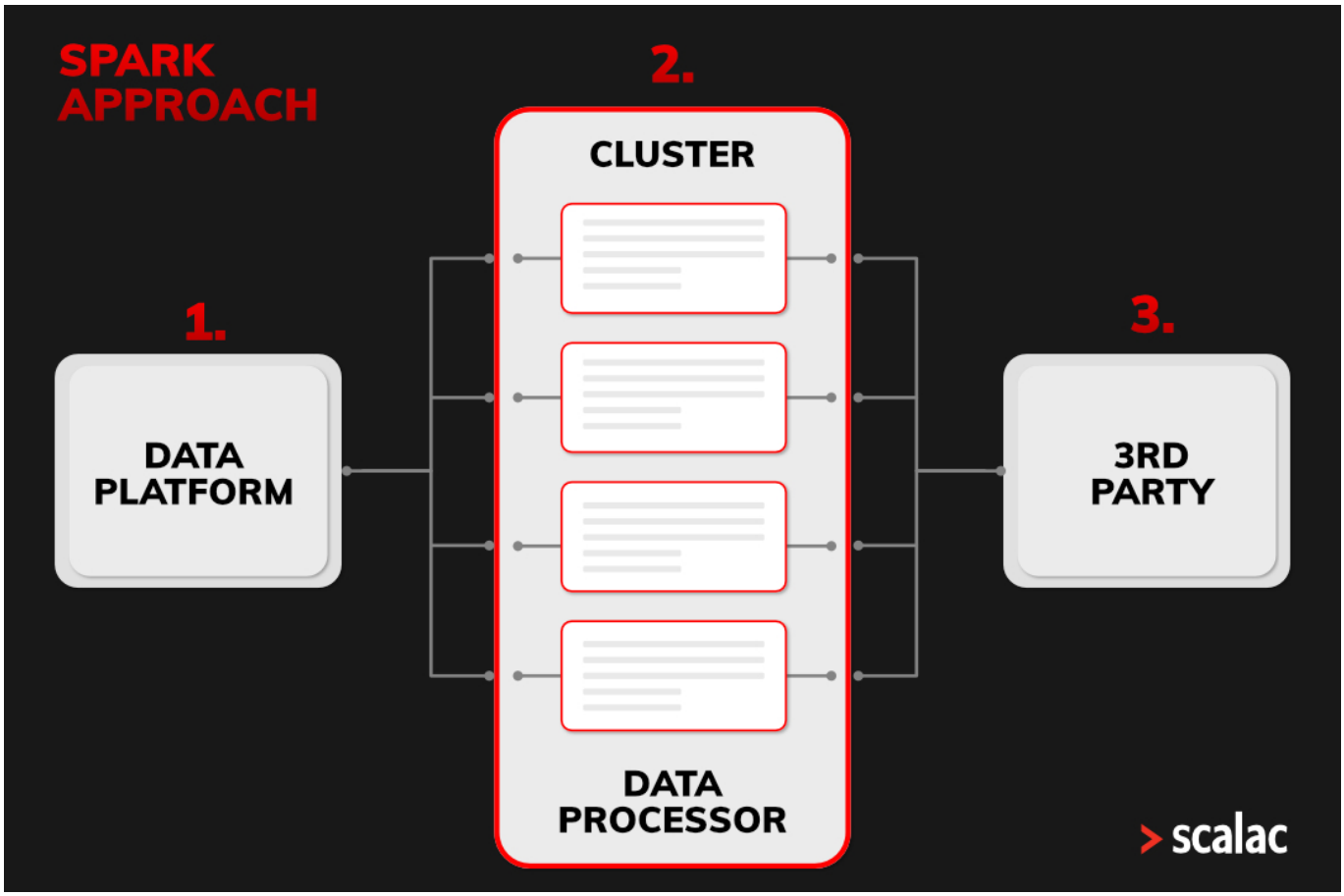
Apache Spark didn't meet our growing requirements, so we proposed a streaming architecture with Akka Streams.

Solution

We re-wrote the process of data retrieval, processing and distribution (ETL) in a streaming manner. This allowed us to reduce the costs, as we no longer needed a Spark cluster, with only a single machine required to handle the same or larger amount of data in a resource-safe manner.

Such approach was possible thanks to the reliable implementation of Reactive Manifesto in Akka Streams, meaning we introduced a solution that was:

1. **responsive** - the solution provides a response no matter the process output
2. **resilient** - the solution remains responsive after a failure, which is ensured by isolating its components
3. **flexible** - the solution remains responsive no matter the size of the input workload
4. **message-driven** - Akka Streams handles message-driven architecture out of the box allowing us to maintain the process using convenient high-level API



Spark approach

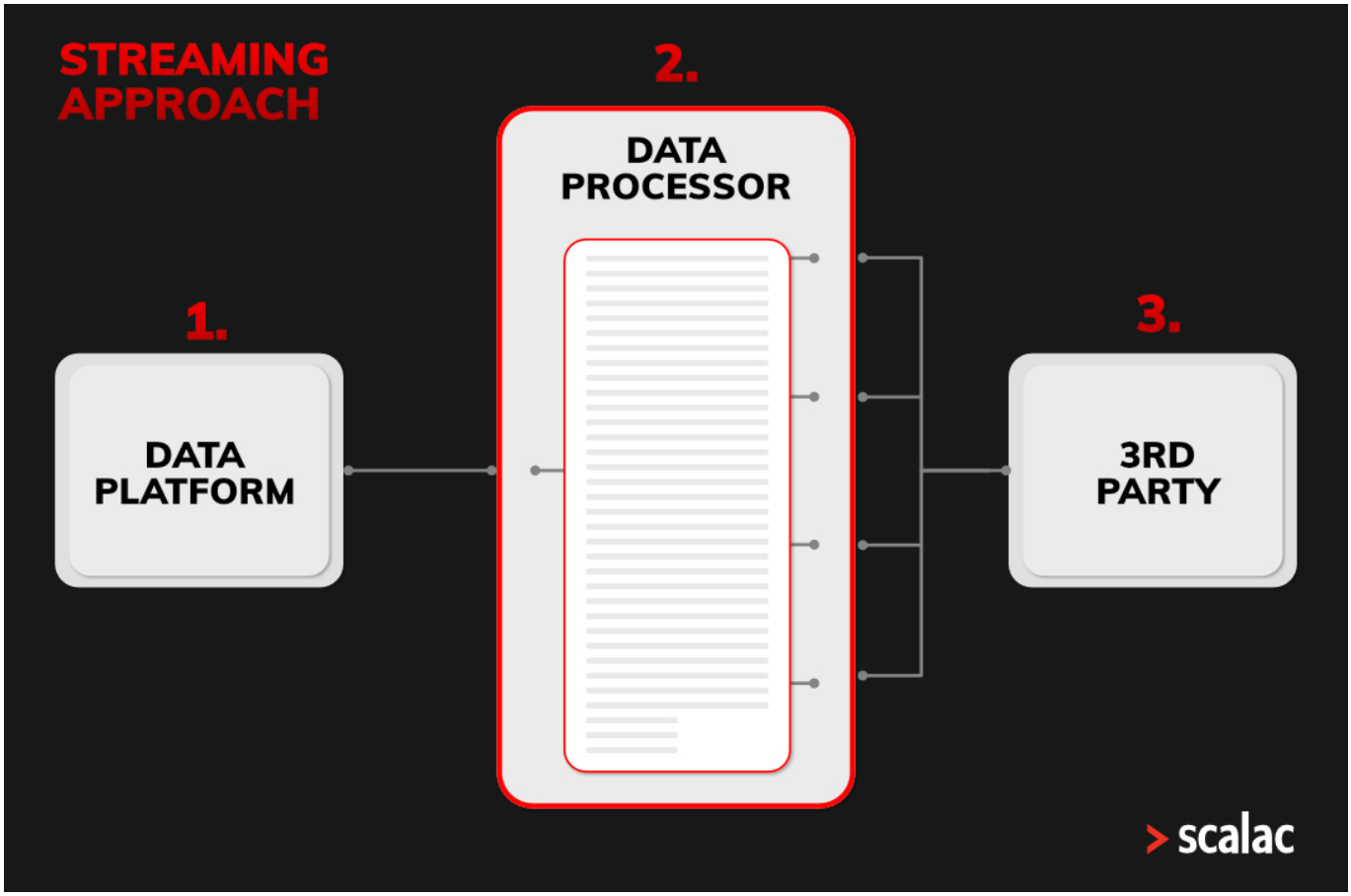
The Data Platform provides the data to the Spark cluster manager. Then, the data are spread across cluster nodes, so each node processes a subset of the whole dataset.

After finished data processing tasks are finally send to the 3rd party to run.

Even though the processing time is not a critical factor in our case, given approach worked well only at the beginning with the initial 3rd party channel.

Soon it became clear there are more drawbacks than advantages:

- higher costs due to the need for spawning the cluster
- the data platform can produce very different amounts of data for each channel, from a few MB to tens of GB
- high memory consumption, but small CPU utilization
- the number of nodes in the cluster was fixed no matter what the data size is
- each node had to fit a subset of data into memory, sometimes just a few hundreds of MB, sometimes few GB
- it quickly becomes clear that some 3rd party channels can take data in parallel using HTTP calls, but some of them require to combine the whole data set into a single processed file. The cluster of nodes made it difficult to gather data back together effectively, combine and compress the data.



Streaming approach

The Data Platform provides the data in a streaming manner, never exceeding the resources of the VM.

Then, the data are processed by a single Data Processor, also in a resource safe, streaming way.

Now it's up to the implementation of the data processor to either push processed data to 3rd party in parallel (e.g., many parallel HTTP calls) or build a single file which later on is stored on GCS or S3 instance.

- lower costs, less VMs needed
- constant, safe memory consumption
- possibility to use multiple cores to process the data
- can handle infinite amounts of data
- 3rd party flexible - from a single to multi-parallel data file ingestion
- the stream can be parallelized during various stages of the process allowing us to use the VM power to the max
- whole process execution time is either the same or faster compared to the previous approach

Results

After implementing the above-mentioned solution, we were able to achieve more efficient integrations and lower overall resource consumption. The former was achieved by replacing Apache Spark with Akka Streams and the latter - by executing various ETL “steps “ in parallel, which allowed the handling of virtually infinite streams of data.With these achievements, we are now limited mostly by third-party capabilities rather than our own.

We have also decreased running costs, reduced the risk of experiencing runtime errors and gained greater control over error handling in general.

What’s more, our solution to substitute Apache Spark with Akka Streams produced the results expected in our initial project assumptions, which means our approach to the problem was well-founded from the beginning.

Our solution achieved the following key benefits:

- Improved integration efficiency
- Lower resource consumption
- Faster processing (limited only by 3rd party capabilities)
- Greater process control
- Reduced weakness to runtime errors
- Reduced costs

People who played a crucial role in the facilitation of the entire release process

Our Agile team:

- **Development Engineers (Scalac):**
    - We started the project with three experienced Scala engineers, who were committed to Tapad business needsFaster processing (limited only by 3rd party capabilities)
    - They carefully took into consideration the existing solution and proposed another approach based on their comprehensive knowledge Reduced weakness to runtime errors
    - After the successful implementation of the proposed Stream Approach, our engineers have been involved into further exploration of one of Tapad's core systems

The Client's team:

- **Team Lead (Tapad)**
- **Product Manager (Tapad)**
- **Head of Engineering (Tapad)**