How I designed a ML prediction service

Saket Joshi | April 15, 2023

Hi, I am

Saket Joshi 👋

Applying for a Summer internship at Twitter!

I am a graduate student studying Data Science at Harvard University. I did my undergraduate education in Computer Science at IIT Tirupati, and am especially interested in Software engineering, Machine learning, and computational finance (Physics is my favorite though). I love to read non-fiction and in my free time, you will find me tinkering with the latest and shiniest frameworks, and trying to get a new perspective wherever possible. I also enjoy a variety of sports, including Chess, Skiing, Table-tennis and running.

Problem Context : ETA as a service

- Fourkites is a supply chain visibility company
- The goal is to process streaming geolocation pings and provide ETA (Estimated time of arrival) and other ML based analytics estimates.

Problem Context : ETA as a service

We Track Over 3 Million Global Shipments Every Day… and Why that Matters

Size matters, and with FourKites, you connect with the largest global network of supply chain data on the planet. Period. This gives you the most accurate ETAs, greater network collaboration and a single source of truth for all your visibility data.

- The shipments are multi-modal, multi-day journeys and have to incorporate factors such as
  - driving regulation laws
  - driver rest patterns
  - Country border crossings (European, US-Canada-Mexico routes)
  - warehouse / cargo vessel appointment details
  - extreme weather forecasts

Description of Problem

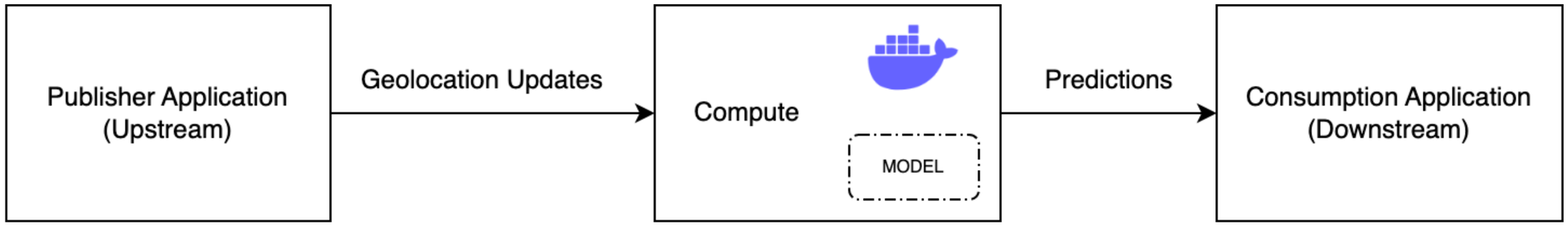
Objective

- Design a streaming service to serve ML model predictions

Description of Problem

Objective

- Design a streaming service to serve ML model predictions



Description of Problem

Objective

- Design a streaming service to serve ML model predictions

Unique Functional requirements

- Several ML models (100’s of models), expecting scalability requirements of up to 10^4 models
- ML models -
  - Ensemble models
  - High variance in model size (from ~10MB to ~2GB) **Why?**
  - Implemented in python
- High variance in request volume to different models
- Up to 1 minute of Latency is acceptable
- Every request has an associated model_id mapping which single model has to serve the request

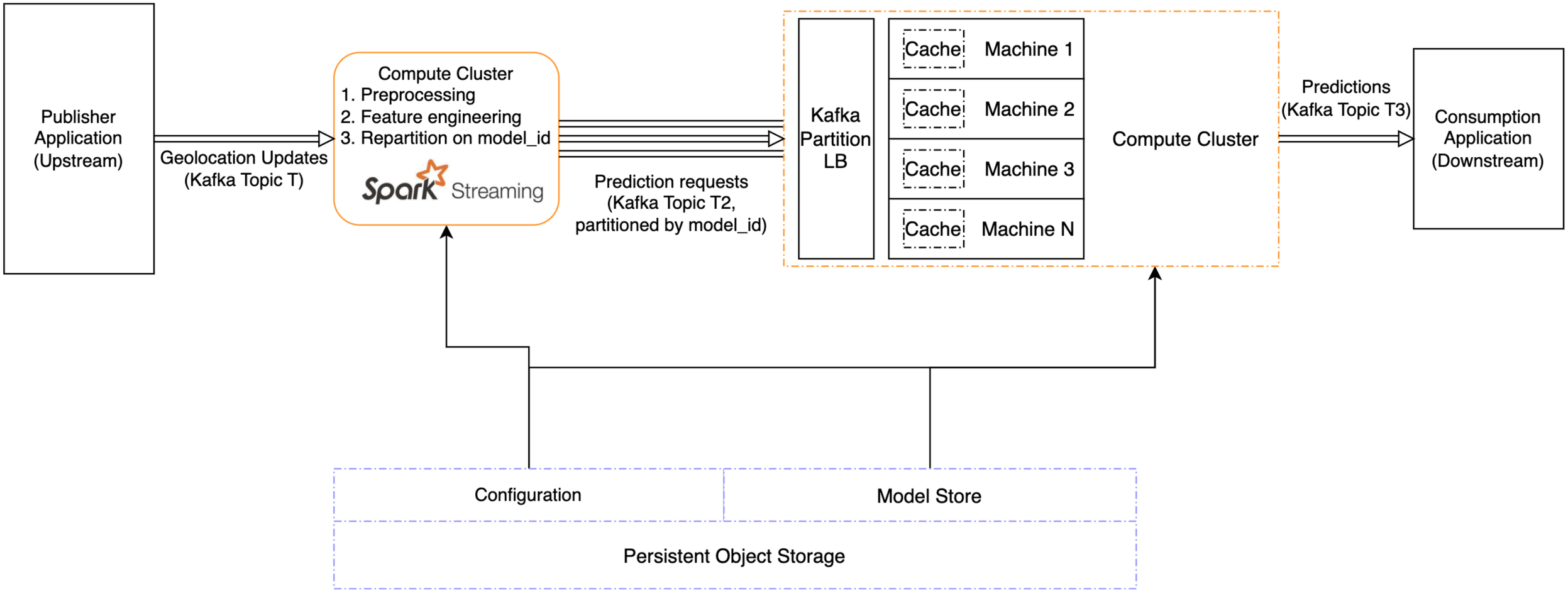
Streaming System Design

Challenge - How do you serve thousands of different ML models requests?

All models do not fit in Memory

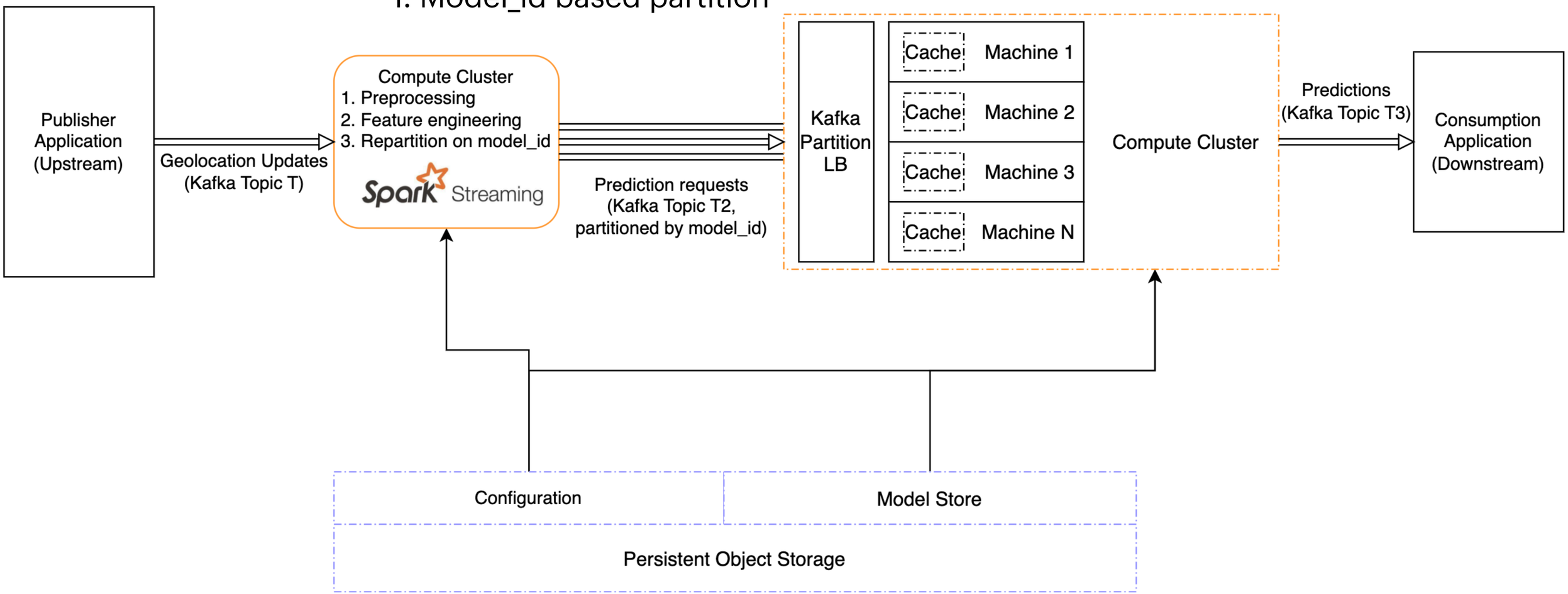
- Proposal 1 - Read batch request -> sort requests by model -> read model from disk -> process request
  - Every batch request can contain requests for all models.
  - Reading all them from disk introduces heavy latency.
  - May need to read up ~100GB of models from disk for every batch - Highly inefficient
- Proposal 2 - Use Large memory machines
  - To scale processing power, need several machines. Large memory machines are more expensive.
  - Vertical scaling ceiling - Cannot expand to additional capacity of 1000’s of models.

Streaming System Design



Streaming System Design

1. Model_id based partition

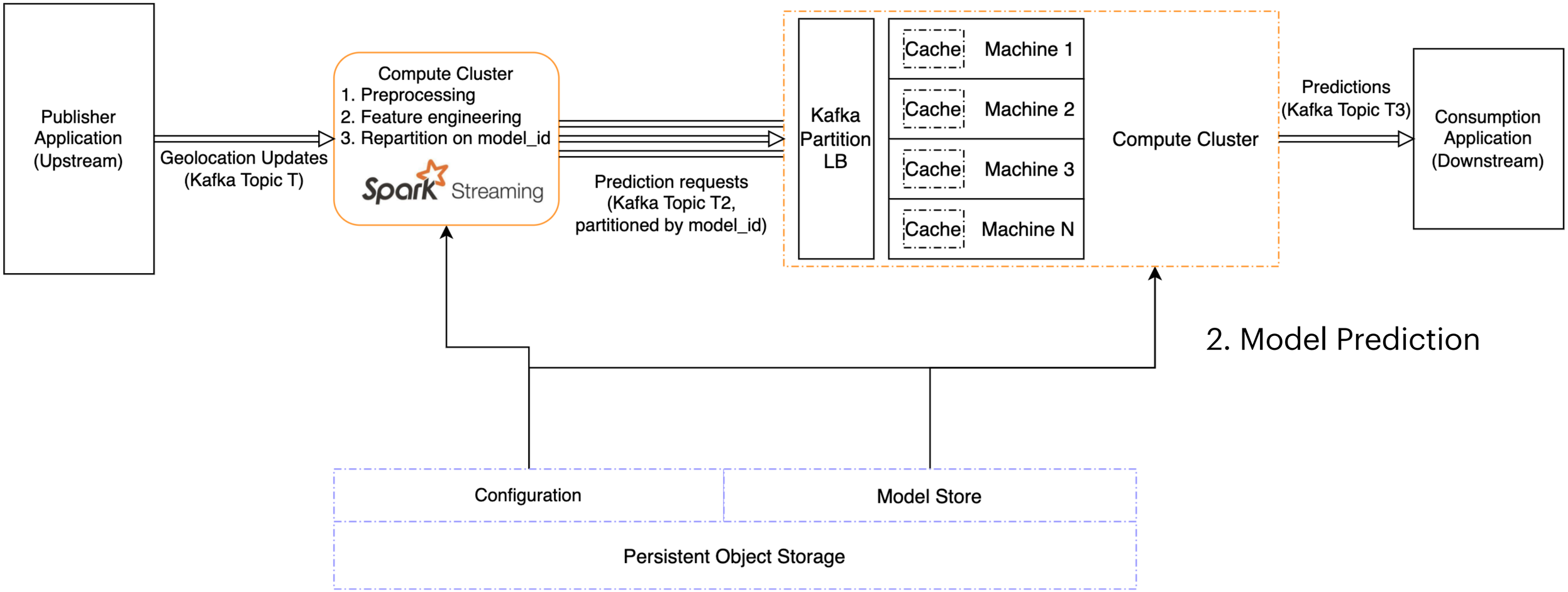


Streaming System Design

Challenge - How do you serve thousands of different ML models requests?

- Partition requests by model_id
  - Messaging Queue - Kafka. Distributed Spark computing cluster
  - Read and redirect messages from input Kafka topic T1 to new topic T2 with model_id based partitioning.
  - For example, if T2 has 20 partitions, and there are 200 model_ids, each partition will be assigned on an average 10 model_ids.
  - Q : What configuration do you use for this mapping?

Streaming System Design



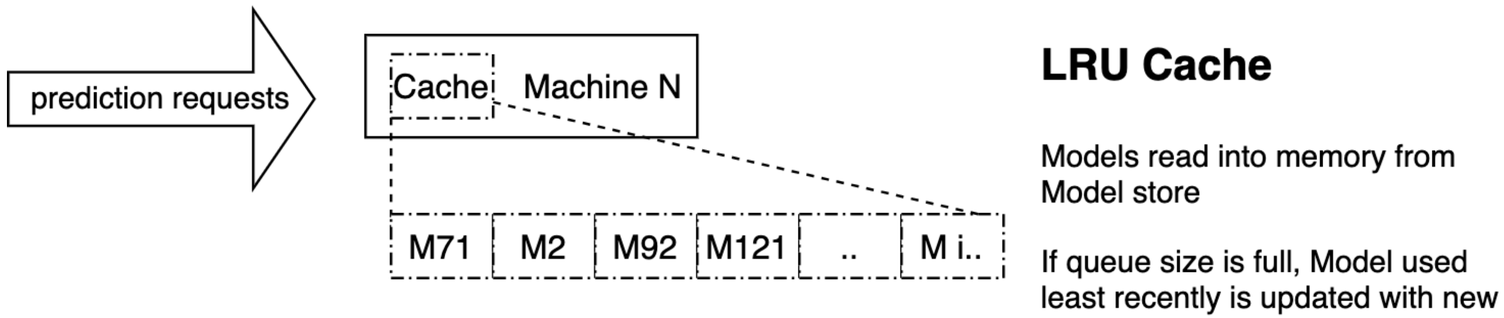
Streaming System Design

Challenge - How do you serve thousands of different ML models requests?

- Model agnostic compute units
  - Each compute unit subscribes to Kafka T2 topic. Gets assigned one or more partitions by the Kafka consumption Load balancer.
  - One compute unit is only reading and processing requests for model_ids for assigned partitions.
  - Which models does each compute unit keep in memory? A static assignment of models to compute units is not fault tolerant.
    - Replication of compute units serving the same model - Static and Inefficient
    - Hence we read models from persistent storage into a Cache Queue

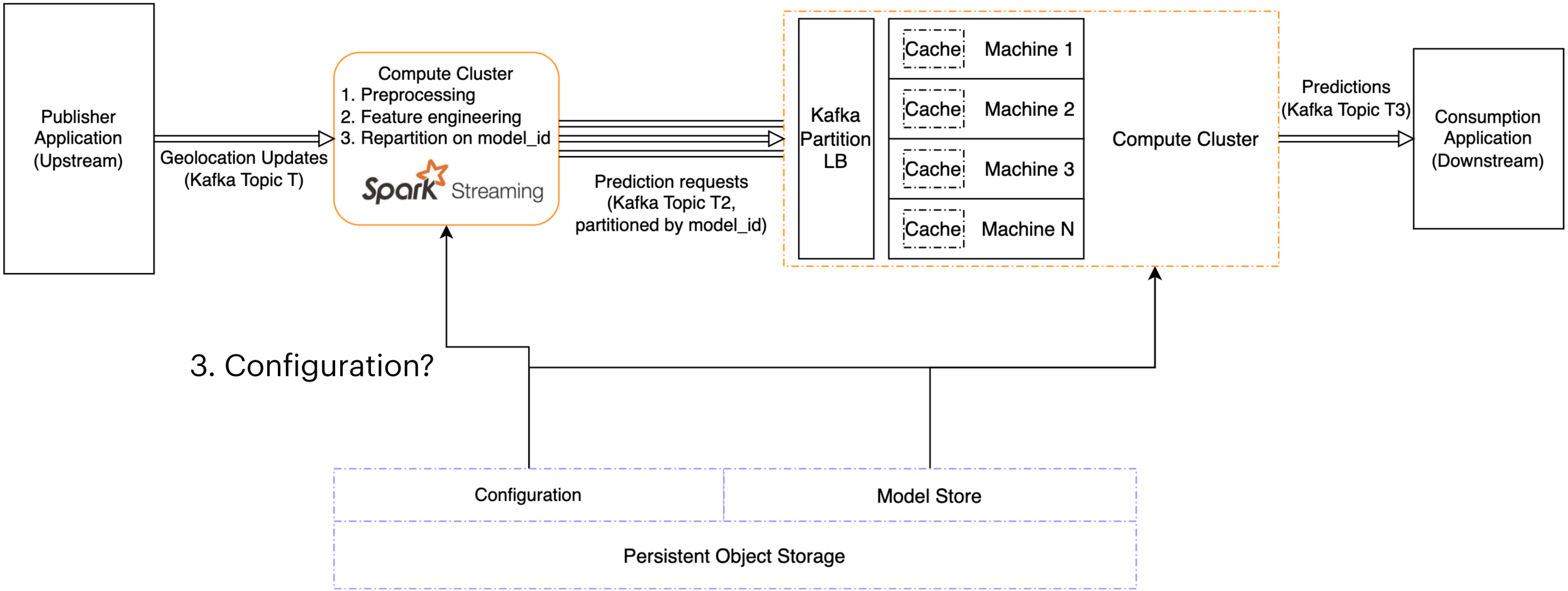
Streaming System Design

Challenge - How do you serve thousands of different ML models requests?



- Model agnostic compute units
  - Each compute unit instead reads models from model store into Cache queue. (LRU)
  - This allows dynamic reallocation of compute units and models mapping
    - If unit is reassigned to new partition, will fetch model difference from model store
    - If unit fails, corresponding partitions are mapped to other units, which will process increased model counts
    - Handles scenario where a large proportion of models get very few prediction requests but still have to be served
    - For increased capacity, this architecture can be scaled both horizontally and vertically

Streaming System Design



Streaming System Design

Challenge - Q : What configuration do you use for Kafka partition mapping?

- Request volume for model predictions have very high variance
  - Random assignment of equal models to partitions can lead to some partitions serving a very high volume of requests.
  - For eg. Certain models may get ~10 req/week, another might get ~10000 req/hour
- We want to balance 2 things across partitions -
  - Number of models
  - Volume of requests

- We use 2 additional services -
  - Prometheus for monitoring and alerting Kafka volume and performance
  - Grafana / Kibana for streaming data exploration and analysis
- We run a Knapsack problem on model request volume and number of to periodically generate appropriate mappings. The weight we want to assign to each item is [1 , volume].
- General persistence in model requests volume - Requires only Infrequent rebalancing.
- Example mapping configuration : { “partition_i” -> list([model_ids] }

Streaming System Design

Challenge - Q : What happens when there are model updates or there is a new model?

- Models are trained periodically and need to be updated
- New models are added frequently to the prediction service
- New models have a cold start problem - Prediction requests volume is not known

- Rebalancing Kafka partitions -
  - By default, N-1th partition is reserved for model_ids which are not mapped. New model gets assigned to this partition.
  - These models are mapped to partitions on the next periodic knapsack mapping cycle.
  - There is no wastage of resources by reserving a partition.
- Model updates are rolled out by repointing the compute units to latest model versions in the model store.
  - This also enables partial rollout redeployment where compute units are restarted/repointed sequentially.
  - No downtime!

Design choices

Why - Kafka?

- High Throughput: Kafka can handle and process millions of messages per second, making it suitable for high-volume data streaming applications.
- Fault-tolerant: Kafka is highly resilient and can replicate data across multiple nodes, so even if one node fails, data is not lost.
- Durability & Replay capabilities: Kafka stores data for a configurable amount of time, so that data can be retrieved and replayed if needed.

Why - Spark Streaming?

- Scalability: Spark Streaming is highly scalable and can handle large volumes of data streams in a distributed manner.
- Open Source: Like Kafka, Spark Streaming is also open-source software, which means it is free to use and has a large and active community of developers contributing to its development and improvement.
- High performance: Spark Streaming is built on top of Apache Spark, which is known for its speed and high-performance processing capabilities

Conclusions

We designed a service to
process ML prediction requests for a large number of models
with High variance in model size
and high variance in request volume

Other design and implementation challenges!

- How do you build a pipeline to train and do performance monitoring of such a large number of models?
- Monitoring and alerting mechanism for excessive model reads from disk
- Measuring the effectiveness of the ML system, both from a system standpoint (Fault tolerance, system availability, etc.) and model performance standpoint (accuracy metrics, data quality, etc.)
- Design Trade-Offs?

Discussion