Monte Carlo simulation

Monte Carlo simulation is a **computational technique** used to model the probability of different outcomes in processes that are inherently uncertain. It's widely applied in **big data** contexts because it allows us to explore **complex, stochastic (random) systems** that can't be solved analytically.

🔹 What is Monte Carlo Simulation?

At its core:

1. **Random Sampling** → Generate a large number of random inputs (from probability distributions).
2. **Process Simulation** → Run simulations for each sample to see how the system behaves.
3. **Statistical Analysis** → Aggregate results to estimate probabilities, distributions, risk, or expected values.

The more iterations run, the closer the simulation approximates the "true" probability distribution of outcomes.

🔹 How Monte Carlo Simulation is Used in Big Data

In the **big data context**, Monte Carlo methods are useful because:

- Data is **high-dimensional** and often contains **uncertainty or missing values**.
- Exact analytical solutions are often **intractable**.
- Cloud computing and distributed frameworks (like **Hadoop, Spark**) make it possible to run millions of simulations in parallel.

Key uses:

1. **Risk Analysis & Forecasting**
    - Finance: simulate stock prices, credit risk, portfolio optimization.
    - Insurance: estimate probabilities of rare but costly events.
2. **Operations & Logistics**
    - Supply chain demand forecasting.
    - Traffic simulations in smart cities.
3. **Engineering & Manufacturing**
    - Reliability testing of systems with many uncertain factors.
    - Energy grid load forecasting.
4. **Machine Learning & AI**
    - Bayesian inference (sampling from posterior distributions).
    - Hyperparameter optimization.

- Model uncertainty estimation.

5. **Healthcare & Life Sciences**
    - Drug discovery simulations.
    - Disease spread modeling.
6. **Big Data Analytics**
    - Filling in missing values by simulating plausible replacements.
    - Stress-testing predictive models under uncertainty.

✅ **Summary**:
Monte Carlo simulation in big data is about **using large-scale randomized simulations to quantify uncertainty and predict outcomes** in finance, healthcare, engineering, and AI. The key advantage in big data is **scalability**—modern distributed systems allow running simulations on massive datasets that were impossible before.

Monte Carlo Simulation and Risk Estimation

Monte Carlo simulation is a computational technique that uses random sampling to model uncertainty and estimate the probability of different outcomes. It's widely used in **risk analysis** when dealing with systems or processes that involve randomness.

---

1. Core Idea

Instead of trying **to calculate risk analytically** (which may be impossible for complex systems), Monte Carlo simulation **generates a large number of possible scenarios by repeatedly sampling** from probability distributions.

- **Each run** of the simulation produces an **outcome.**
- Collectively, **thousands (or millions) of runs build up a distribution of possible results**.
- From this distribution, we can estimate **risk metrics** such as **probability of loss, Value-at-Risk (VaR), expected shortfall, or worst-case scenarios.**

Important Terminologies

- **Probability of loss** → How often do we lose?
- **Value-at-Risk (VaR)** → The maximum loss we expect not to exceed, with a chosen confidence (e.g., 95%).
- **Expected Shortfall** → The average loss in the worst cases, beyond the VaR threshold.
- **Worst-case** → The absolute largest loss possible in the simulations.

Example:
You invest **$100,000** in a portfolio and run **10,000 Monte Carlo simulations** of future returns. After running the simulations, you find:

- In **3,000 simulations**, you lose money.
- In the worst 5% of cases, losses are **$12,000 or more**.
- The **average loss** among those worst 5% is **$18,000**.
- The very worst simulated outcome is a **$40,000 loss**.

🔹 **Now, let's interpret each term:**

1. **Probability of Loss**
   - You lost money in 3,000 out of 10,000 simulations → **30% chance of loss**.
   - ✅ *This is simply how often the portfolio loses value.*

---

2. **Value-at-Risk (VaR)** (at 95% confidence)
   - The cutoff for the worst 5% is **$12,000**.
   - ✅ *With 95% confidence, you will not lose more than $12,000.*

---

3. **Expected Shortfall (ES)** (at 95% confidence)
   - The average of those worst 5% losses is **$18,000**.
   - ✅ *If things get really bad (worse than the $12,000 VaR), you can expect to lose about $18,000 on average.*

---

4. **Worst-Case Scenario**
   - Out of all simulations, the single biggest loss was **$40,000**.
   - ✅ *This is the absolute worst outcome observed.*

How to perform Monte Carlo simulation with PySpark

You create your own Monte Carlo simulation logic and then use PySpark's capabilities to run it at a massive scale. This generally involves these steps:

1. **Define your simulation logic:** You write a Python function that performs a single Monte Carlo trial. This function will use the drift (mu) and volatility (sigma) parameters to generate a random price path for a stock.
2. **Generate a parallel list of trials:** You use PySpark to create a distributed collection (e.g., an RDD or DataFrame) of the number of simulations you want to run.
3. **Distribute the simulation:** You then use PySpark's map transformation to apply your custom simulation function to each element of the parallel collection. PySpark takes care of distributing these independent trials across the cluster of machines.
4. **Aggregate the results:** After all the simulations are complete, you can use PySpark's aggregation functions to gather and analyze the final results, such as calculating the average final price, a value-at-risk, or plotting a distribution.

   This approach is extremely powerful for big data scenarios where you need to run thousands or even millions of independent simulations, as PySpark can handle the workload far more efficiently than a single machine.

Question

Design and implement a distributed Monte Carlo simulation system using PySpark to perform financial risk forecasting for a real-world stock.

Specifically, your task is to:

1. Acquire real historical stock data (for example, Apple Inc. or any Indian stock such as Reliance Industries) automatically using an open-source API such as yfinance, and save it in a structured CSV format.

2. Load the dataset into a PySpark DataFrame and compute daily logarithmic returns from adjusted close prices using Spark SQL functions.

$$\text{log\_ret}_t = \ln\left(\frac{P_t}{P_{t-1}}\right)$$

3. From these returns, estimate drift (μ) and volatility (σ) parameters within Spark.

4. Implement a Monte Carlo simulation model (e.g., Geometric Brownian Motion) to forecast future stock price paths for a fixed horizon (e.g., 252 trading days).

   Uses **Geometric Brownian Motion (GBM)** model:

   $$S_{t+1} = S_t \times e^{(\mu - 0.5\sigma^2) + \sigma Z_t}$$

   `z` is a standard normal random shock.

   Drift term (`mu - 0.5σ²`) captures expected growth.

   Diffusion term (`σ * sqrt(dt) * z`) adds randomness.

   Returns the **entire simulated price path**.

5. Execute the Monte Carlo simulations in parallel using PySpark — where each Spark partition or executor runs independent simulation paths concurrently.

6. Collect and analyze the simulation results, computing risk measures such as Value at Risk (VaR) at the 5% confidence level. Also compute the expected mean final price

7. Visualize the resulting distribution of simulated final stock prices using a Python visualization library (e.g., matplotlib or seaborn), marking the VaR and expected mean price on the plot.

- **Compute and interpret multiple risk metrics** from the simulated distribution of final stock prices, including:

  - **Value at Risk (VaR)** at a 95% confidence level
  - **Expected Shortfall (Conditional VaR)** at a 95% confidence level
  - **Probability of Loss** ($P(S_t < S_0)$)
  - **Worst-Case Scenario** (minimum simulated final price)

- **Visualize** the final price distribution and annotate the above risk metrics (VaR, ES, probability of loss threshold, and worst-case price) using a suitable Python visualization library (e.g., Matplotlib or Seaborn).