Sampling for Reliability of High Scale Systems

Arjun Reddy Lingala

arjunreddy.lingala@gmail.com

Abstract

As companies grow they often deal with vast amounts of data and with modern data analytics use cases companies would like to capture as much data as possible and use it for multiple reasons like improving user experience and understanding which new products to build. Building new systems alone is not going to improve the user experience and bring more users to applications. High volume systems these days demands having reliability at each step of the product to not harm the user experience of the product and bring in more users. Few of the reliability use cases that a product has to built at minimum include — Are the users able to land on the webpage of the product and get response? How is p95 latency of a web page request or internal service request? Is latency affected the app for all regions across the world or is it for specific regions only? Is there any change is type of data uploaded or searched in your app? Applications with reliability needs to have monitoring for all these scenarios and more. With growing amount of data across the world, the cost of capturing the data to get answers to reliability questions increases. In this paper, I discuss various sampling strategies that can be used for reliability of high volume systems and compare various approaches in terms of implementation complexity and associated cost

Keywords: Reliability, High Scale Systems, Sampling Strategies, Static Sampling, Random Sampling, Stratified Sampling, Dynamic Sampling, Adaptive Sampling, Monitoring, Time Series Data

I. INTRODUCTION

As modern systems evolve, the need for handling huge volumes of data is required and accurately working of these systems with almost no downtime has become more critical than ever. Ensuring reliability of a system includes a) monitor- ing if the system is working as expected — capturing all the requests coming to the app that includes each and every request within the app and notify the owners if there is any discrepancy (increased or decreased by X times), b) evaluating performance metrics — an action from a user in an app performs multiple service requests inside an app. Keeping track of response times of each request and having an SLA of response times is critical for an application. Usually p95 or p99 response times are used to avoid anomalies that occur due to bad connection on the client side c) detecting anomalies in systems — modern systems are expected to handle anomalies very efficiently. For example, if an app is spammed with multiple requests from the same IP and app has detected the anomaly and rate limited the spammer, then the resources can be used for good users. d) managing resources efficiently — with increase in data volume and complexity it becomes challenging to scale infrastructure to match the scale and it impacts performance and storage cost. Handling performance and storage for high volume systems is critical as it directly impacts the user experience and growth of an application.

To maintain reliability of the systems, sampling is used as a key strategy in capturing the data for managing large scale data. Sampling strategies extract smaller, manageable subsets of data that can be monitored without losing the actual value by providing weights which determine closer to actual behavior of the system.

With reduce in volume of data captured and processed, sampling minimizes the performance and storage overhead and improve latency while still serving critical use cases of the system.
This paper explores how sampling is used in capturing reliability of high volume systems focusing on how to approach various sampling strategies based on the use case maintaining service availability. Additionally, we also discuss various trade-offs, additional cost involved, complexity in implementation of various strategies and offer insights into design of his volume systems meeting modern scaling needs.

II. Background and Use cases

Modern systems are evolving rapidly and with it the need for systems to handle large volume data and these systems are designed with data as the important aspect and systems evolve by learning data patterns. Large volume data is processed and analyzed to discover trends, create models and use the analysis to evolve the system with an understanding of future needs of the application. Modern systems also heavily rely on machine learning that includes building deep learning models that thrive on large datasets, recommendation systems which continuously analyze user behavior to provide personalized content, real-time monitoring systems that forecast future trends and optimize operations and resources. Traditional systems relied on on premise data centers with limited capacity. Cloud systems on the other hand can scale horizontally across distributed systems. Modern data warehouses using HDFS and object storage service like S3 are being used commonly for large datasets. This enables big companies to process and analyze billions of events daily and providing deeper insights. In past, applications relied on static reporting tools for deci- sion making. Modern systems incorporate dynamic analytics engines to provider deeper analytics and real-time dashboards. Companies can monitor real-time business metrics, forecast future needs and optimize resources and dashboard tools can connect to large datasets visualizing trends and enabling data based decision making.
In all the cases it is clear that large scale systems are essential for the growth and addressing user needs of any application. With increase in data, it requires to scale the systems to process and analyze the data which can be achieved with horizontally scalable solutions like HDFS and S3. Based on various types of application requirements and deliverables, it is also required to scale memory and CPU in addition to storage. Though horizontal scaling of storage, memory and CPU are easy when compared to vertical scaling, it comes with additional cost and complexity of implementing distributed systems. With increase in various kinds of applications cover- ing multiple services for users and the impact any application being down due to some issue is huge. Systems should be ready to handle unexpected downtime, growth at each step by maintaining reliability. It is key for any modern system to build reliability metrics across multiple stages of the app. At minimum, each system requires to capture number of requests coming in from various groups or dimensions, p95 or p99 latency or response time of a request. Applications can buffer the event data of an application and send data to server in batches to minimize the data size of high volume systems, but it still comes with huge cost to any application with increase in size of data.
Sampling is the technique that allows organizations to build reliability metrics from large scale data without processing entire dataset. It involves selecting a subset of data by enabling efficient data processing while delivering reliable insights. As data volume grows exponentially, processing becomes increas- ing resource intensive for storing, managing and analyzing. Sampling allows systems to work with small manageable datasets by working with samples that represent the entire large dataset. As scale of the data grows, sampling can be adjusted to resources available and deriving reliable insights. By reducing the amount of data to analyze for reliability, sampling allows systems to make quicker decisions and enable real-time insights by maintaining accuracy.

III. SAMPLING TECHNIQUES

A. *Static Sampling*

Static Sampling is the fundamental method used in most of the cases which ensures that each event has equal probability of being selected by minimizing selection bias. Static sampling is easiest of all sampling techniques to implement and it will not include additional cost for storage of metadata. It uses a fixed sample rate N to determine whether an event can be selected for processing. Sample rate is selected by analyzing the data volume, storage and resource requirements. It is done by randomly assigning numbers to the events and from 1 to N and selecting the event if the number assigned to event is
N. Pseudo Random generators by taking initial input as seed
(N) is implemented in many programming languages. Time complexity of assigning a random number to an event is O(1) and not require any additional storage to implement which is a very cost effective technique. each day, capturing all the clicks and building real-time metrics for reliability whether the clicks are working as expected and how the trends are over time, it would cost in memory and storage a lot processing all the info. Using Static sampling with sample rate of 1000 will reduce amount of data captured from billions to millions which is within range of compute for modern machines.

B. ***Stratified Sampling***

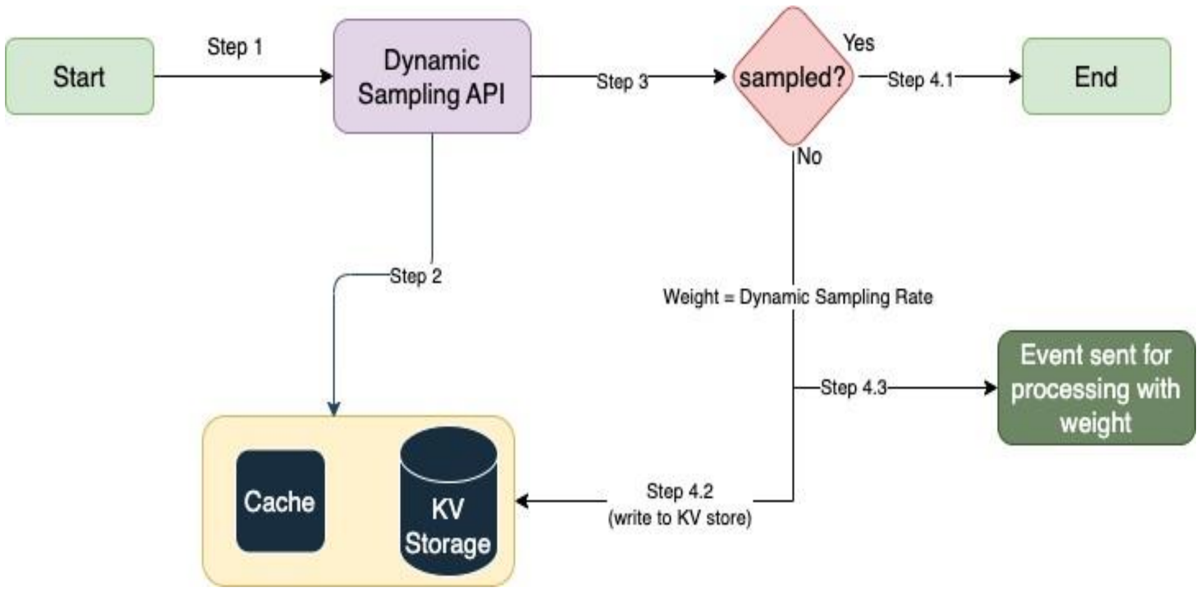
Start
Step 1
Dynamic Sampling API
Step 3
sampled?
Yes
Step 4.1
End
No
Step 2
Weight = Dynamic Sampling Rate
Step 4.3
Event sent for processing with weight
Cache
KV Storage
Step 4.2 (write to KV store)

Fig.1. Stratfied Sampling Processing Flow

Dynamic sampling is the flexible method which can be used in situations for processing large data sets in real-time with efficient and scalable computation. Dynamic sampling adjusts the sampling rate based on certain criteria also called as dimension. Unlike static sampling where the sampling rate is fixed overall, dynamic sampling requires input as N samples per X time (seconds, minutes) per dimension. Dynamic sampling uses a key-value store like HBase or DynamoDB to keep track of the dimensions of events that happened in the given unit of time. Dimension or key for the dynamic sampling can have more than one variables. In case of multiple variables, dynamic sampling concatenates the dimensions and use it as key. If the number of variables increase to a point where the concatenation is more than 256 characters long (most of the programming languages have limit of 256 characters for string), then the key can be concatenated and then hashed to use it as key for key-value storage. If the number of events of a given key has reached the set target samples per unit of time, then the events received after that point will be sampled out for processing. When an event is received for processing, dynamic sampling API builds the key based on dimension selected and checks in key-value storage for all values with same key. If the aggregated value from the key-value storage is less than the input seed, then the event is not sampled out, sent for processing and it is written to key-value store with key

$1\,P(x) = N$ — (1)

and current timestamp as key and 1 as value with a TTL (time to live) equivalent to input X unit of time. If the aggregated Static sampling can be usually used in cases where there is only one type of metric involved in monitoring for building reliability metrics. For example, if the use case is to capture number of clicks of an app and if app has billions of clicks for value from the key-value store is greater than or equal to input seed, then the event is sampled out and not sent for processing. Dynamic sampling makes sure that the key-value store will have less than input seed number of samples for a given key at any given point of time. Entries into key-value store will be automatically deleted after the TTL has reached.

Dynamic sampling can be usually used in cases where there are multiple dimensions involved in building metrics. For example, if the use case is to capture number of clicks of an app across multiple regions of the world and alert us when there is spike or drop in number of clicks per region based on historical trends, dynamic sampling would be best case for this situation. Dynamic sampling comes with additional cost in terms of both compute and storage. For each event, it queries the backend key-value storage to see if the event can be processed or not which can increase the cost of compute and all keys are stored in key-value storage which can increase the cost of storage as number of keys grow. As most of the use cases for reliability involve monitoring metrics across various dimensions, dynamic sampling can be used widely.

C. *Adaptive Sampling*

Adaptive sampling is best suited technique for high vol- ume data and with more dimensions. With some additional complexity in implementation, it combines the techniques of both static and dynamic sampling techniques and builds an optimized solution addressing pain points in each technique. Usually data differs a lot in volume and size by dimension and for huge scale varying data applying fixed sample rates or having a fixed number of samples per unit of time and setting the weight at the time of the processing the data might lead to up-weight or down-weight of the actual metric. Adaptive sampling has two types of sampling rates involved as this technique applies sampling at two levels - static sampling rate and dynamic sampling rate. It uses different sample rates for static sampling step for each dimension based on the event volume and it will be updated periodically to adapt based on the event volume in last X units of time. Static sampling rates are stored as a config file on a different repository with higher and faster commit rate. In addition to this, dynamic sampling rate with Y number of samples per X units of time is defined at overall processing level. Dynamic sampling rate is defined at entire processing level, not defined per dimension. Adaptive sampling involves defining three key components to its technique

1) Adaptive Sampling Config: Static sampling rates are defined based on the event volume or size per dimension and are updated for every few minutes to adapt to changing volume of events per dimension. These sampling rates are recommended to store in a configuration file in different repository that accepts higher commit rates as the goal is to commit every few units of time and also higher read rates as the configuration file is read for every event. Higher read rate can be addressed by building caches that invalidates when a write happens. Sample rates are defined based on tiers which are determined based on event volume or size — tier 1, tier 2,. . . tier N. Tier to sample rate is defined in a config file which will be updated manually based on the analysis on resources and compute. A new job will be created to query time series data by dimension for last T units of time and tier is determined for each dimension based on the maximum value from the query. Static sampling rate from the config file that matches the tier definition of the dimension is considered as sample rate for the dimension. This job will be scheduled by chronos to run every P units of time and update the static sampling rate by dimension based on tier definition

2) *Sample rate per unit time:* Dynamic sampling rate is defined as target number of samples per unit of time. This is adjust mainly based on computing resources and storage point of view. As explained in dynamic sampling, this component has keys written to key value storage with TTL and keys will be deleted from the storage when the time expires.

$$Weight = Rate_{AdaptiveConfig} \times Rate_{PerUnitTime} \qquad (2)$$

3) *Adaptive Sampling API:* Adaptive sampling API queries both above components a) and b) to determine static and dynamic sampling weights and decides whether to sample an event or not. It first reads the static sampling rate configuration file and looks for sample rate for the dimension related to processing event. If the dimension is not available, then default sample rate is used. It then applies pseudo random function to based on sample rate to check if the event should be sampled out or sent for processing. If the event is sampled out, processing of the event ends here. If the event is not sampled out, then the processing is further sent to dynamic sampling step with static sampling weight equivalent to sample rate of the dimension. At this point, it then queries key value store to check if the event should be sampled out. If the event is not sampled out, then the event is sent to further steps to monitoring with weight. Weights are used to determine the approximate value of the event when queried for analysis.

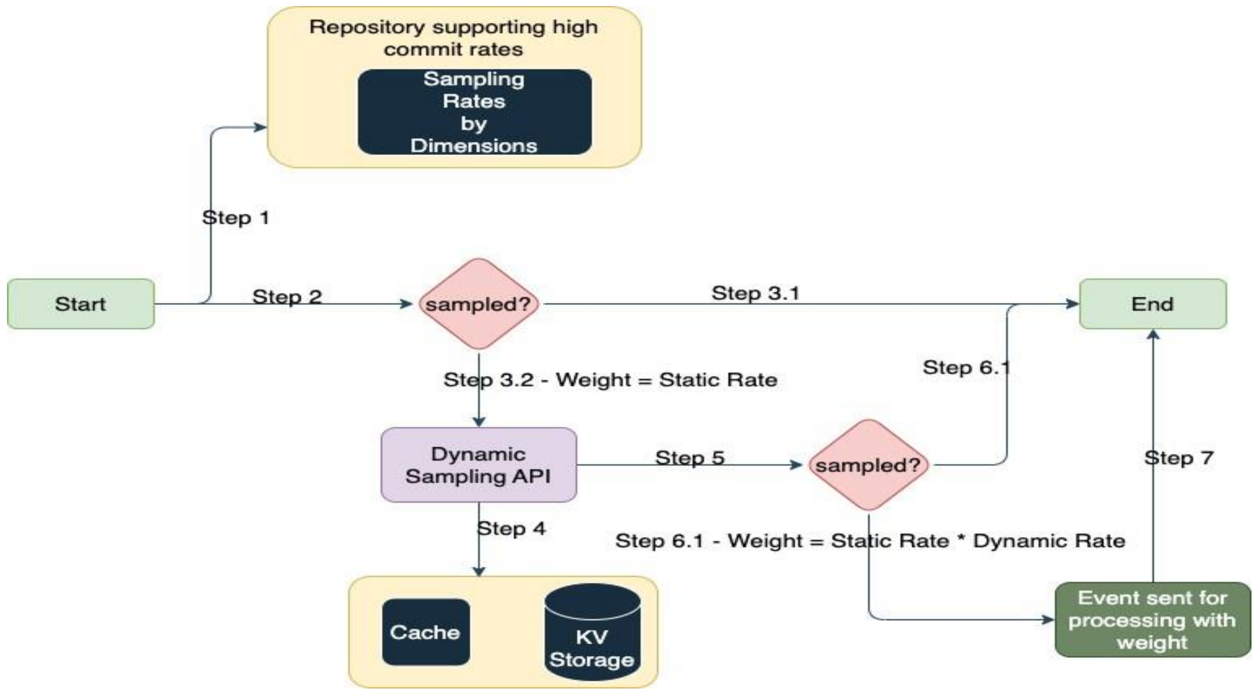
Repository supporting high commit rates
Sampling Rates by Dimensions
Step 1
Start
Step 2
sampled?
Step 3.1
End
Step 3.2 - Weight = Static Rate
Step 6.1
Dynamic Sampling API
Step 5
sampled?
Step 7
Step 4
Step 6.1 - Weight = Static Rate * Dynamic Rate
Event sent for processing with weight
Cache
KV Storage

Fig.2. Adaptive Sampling Processing Flow

Adaptive sampling comes with additional complexity of implementation, but it handles all the pain points of static and dynamic sampling. Events that have varying volume by dimension can be best tracked by adaptive sampling. For ex- ample, if the use case is to capture number of clicks of an app from different regions, applying static or dynamic sampling will not help in catching anomalies as the highly populated regions will have lot of data while less populated regions will have very less data and static or dynamic sampling samples out anomalies. With adaptive sampling, different regions will have different static sampling rates and less populated regions might not be sampled at all providing insights about anomalies. If the static sampling for high populated regions samples less events, then dynamic sampling in next step will address those events.

IV. Comparison of Sampling Techniques

All sampling techniques discussed have advantages and disadvantages and are recommended to use base on the use case and the resources available to the organization. We compare sampling techniques from three aspects

A. Cost

Static sampling is the best one of all three techniques in terms of cost as it just uses pseudo random function which is available in most of the programming languages and only takes O(1) time complexity, where as dynamic sampling requires additional storage to store keys in key value storage. Dynamic sampling involves checks whether there is a key in key value storage takes O(k) time complexity where k is target number of samples per unit of time. Each query to dynamic sampling API has to read the key value storage for all keys stored which would be maximum of n. Space complexity for dynamic sampling is O(p * k) where p is number of possible keys from dimensions. Adaptive sampling on the other hand has same time complexity as dynamic sampling — O(k) for querying the API to determine if the event should be sampled or not. It also has additional complexity of query last N unit of time of data from time series and updating adaptive config which takes O(p * N) with p being number of possible keys from dimensions. Space complexity for adaptive sampling is O(p) for adaptive config and O(p * k) for key value storage.

B. Data Accuracy

Adaptive sampling technique of all techniques provides more data accuracy. For example, if the use case is to capture number of clicks of an app by region. Static sampling with sample of 1000 runs only 1 event in 1000 events. This might lead to processing of events in highly populated regions as less populated regions will have less events and have high chance of getting sampled out. This results is understanding that there are no events from less populated regions. Dynamic sampling on the other hand address the problem in static sampling but not address it entirely. In the same example, dynamic sampling sets N target samples per unit of time. This results in capturing N samples for both high and less populated regions with weight which might lead to either up-weighting or down- weighting the actual metric value of the region impacting data accuracy. Adaptive sampling address both problems discussed above by changing the sample rate adapting to event volume

TABLE I: Sampling Types Comparison

Sampling Type	***Static Sampling***	***Stratified Sampling***	***Adaptive Sampling***
Time Complexity (Cost)	O(1)	O(k)	O(p * N)
Space Complexity (Cost)	N/A	O(p * k)	O(p * k)

Data Accuracy	No dimensions involved	Dimensions having equal amount of event volume	Best
Implementation Complexity	Very easy	Key value stor- age	Key value stor- age and adap- tive config

k - target number of samples per unit of time p - possible number of keys from dimensions N - last N hours of data for adaptive config

C. *Implmentation Complexity*

Static sampling is easiest one to implementation as it just uses in-built function of the programming language where as dynamic and adaptive sampling requires implementing API which queries and saves data to key-value backend storage. Adaptive sampling requires setting up additional repository with high commit rate and a script to update adaptive config every X hours based on event volume.

V. Conclusion

In this paper, we discussed how modern systems evolve, how data at various stages of the app and event processing are required to improve user experience and growth of the app; we also discussed the importance of reliability of high scale systems and how the down time can impact the growth and user experience of any system; we also discussed what is cost and operational overhead for processing entire data for reliability use cases; we introduced how sampling can help with addressing reliability use cases by introducing various sampling techniques suitable for reliability use cases that include static sampling – which is implemented by pseudo ran- dom functions of the programming language and sampling out random events based on the sample rate, stratified sampling – which is implemented by using key value storage as back-end and samples out events by ensuring target number of samples per unit of time and adaptive sampling – which is implemented using key value storage and adaptive config which does both static sampling and dynamic sampling ensuring data accuracy; we also compared these sampling techniques in terms of cost, implementation complexity and data accuracy.

References

1. K. Shvachko, H. Kuang, S. Radia and R. Chansler, ”The Hadoop Distributed File System,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 2010, pp. 1-10, doi: 10.1109/MSST.2010.5496972.
2. Mehul Nalin Vora, ”Hadoop-HBase for large-scale data,” Proceedings of 2011 International Conference on Computer Science and Network Technology, Harbin, China, 2011, pp. 601-605, doi: 10.1109/ICC- SNT.2011.6182030.
3. M. Elhemali, N. Gallagher, N. Gordon, J. Idziorek, R. Krog, C. Lazier, Mo, A. Mritunjai, S. Perianayagam ,T. Rath, S. Sivasubramanian, J. Christopher Sorenson III, S. Sosothikul, D. Terry, A. Vig, “Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service,”, 2012, https://aws.amazon.com/dynamodb/
4. Amazon Web Services, Inc., ”Amazon Simple Storage Service (S3) – Cloud object storage,” Amazon Web Services, Inc. Available: https://aws.amazon.com/s3/.
5. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, 2004

6. G. Chaudhuri, K. Hu and N. Afshar, ”A new approach to system reliability,” in IEEE Transactions on Reliability, vol. 50, no. 1, pp. 75-84, March 2001, doi: 10.1109/24.935020.
7. J. Kostolny, M. Kvassay and S. Kovalik, ”Analysis of system reliability by Logical Differential Calculus and Decision Diagrams,” The Interna- tional Conference on Digital Technologies 2013, Zilina, Slovakia, 2013, pp. 170-175, doi: 10.1109/DT.2013.6566306.
8. G. Goldberg, D. Harnik and D. Sotnikov, ”The case for sampling on very large file systems,” 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 2014, pp. 1-11, doi: 10.1109/MSST.2014.6855542.
9. O. Arslan and P. Tsiotras, ”Machine learning guided exploration for sampling-based motion planning algorithms,” 2015 IEEE/RSJ Interna- tional Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 2646-2652, doi: 10.1109/IROS.2015.7353738.
10. D. Shankar, X. Lu and D. K. Panda, ”High-Performance and Re- silient Key-Value Store with Online Erasure Coding for Big Data Workloads,” 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 2017, pp. 527-537, doi: 10.1109/ICDCS.2017.224.
11. N. Yang, J. Liu, P. Zhang, C. Zheng and Q. Liu, ”MES: A memory- efficient key-value storage with user-level network stack,” 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA), Shanghai, China, 2018, pp. 156-161, doi: 10.1109/ICBDA.2018.8367668.