Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

PREDICTIVE RESOURCE MANAGEMENT BY REDUCING COLD START IN SERVERLESS CLOUD

MUHAMMAD SULEMAN*
Department of Information Technology, The Islamia University Bahawalpur, 63100, Punjab, Pakistan.
*Corresponding Author Email: muhammad.suleman@iub.edu.pk

DOST MUHAMMAD KHAN
Department of Information Technology, The Islamia University Bahawalpur, 63100, Punjab, Pakistan.

MUHAMMAD ABID SALEEM
Department of Information Technology, The Islamia University Bahawalpur, 63100, Punjab, Pakistan.

OMER RIAZ
Department of Information Technology, The Islamia University Bahawalpur, 63100, Punjab, Pakistan.

ZAIGHAM MUSHTAQ
Department of Information Technology, The Islamia University Bahawalpur, 63100, Punjab, Pakistan.

Abstract

Serverless Cloud computing expanding its domain rapidly. This is simple, efficient, light-weight, secure and ubiquitous. All Cloud players provide it with different attractive names such as Amazone branding it with AWS Lambda, Goole using Cloud Run, Ali Baba calling it Function Compute and last but not least Microsoft providing serverless cloud with name of Azure Function. Normally, function service executes the core business logic of application and host's machine policy of execution create a significant impact of overall quality of service provided by CSP (Cloud Service Provider). To produce an effective execution policy, the host machine maintains a lean balance between Cold and Hot restart. Policy efforts to reduce Cold restart but manage resources during Hot restart. In this paper, we employed a machine learning based classification methodology that segregate the functions in terms of cold and hot functions. We implemented Naïve Bayes classifier and boosting the accuracy with Kernel Density Estimation. The overall best accuracy was observed up to 94.35%.

Index Terms: Serverless Cloud Computing, Cold Start, FaaS, Resource Management, Naïve Bayes Classifier, Kernel Density Estimation, Windows Azure Function.

1. INTRODUCTION

There were many issues behind the conventional Cloud computing and consequently, the reason of birth of Serverless Cloud computing. Eight major issues were explained by [1] including redundancy of availability, geographical distribution, load balancing, autoscaling, continuous monitoring, logging performance, continuous upgrading and service migration. The above-described architecture [1], [2], [3] of Serverless added some new layers in conventional Cloud architecture. The new layers are "Serverless" and "Triggers". The Serverless is segregated in two main services: BaaS (Backend as a Service) and FaaS (Function as a Service).

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

BaaS is related to backend tasks such as database, file system, messages etc. and FaaS belongs to user deployed function. In fact, BaaS work as a backend service for FaaS. User will generate functions by using trigger and these functions will utilize BaaS to complete their jobs. Many times, the term “Serverless” and “FaaS” are used in similar context because users are familiar with both of these. It means that user only interacts with FaaS, and, how to manage the FaaS execution that increase the thruputs of productivity and reduce the latency in function executions, is the main concern of our thesis.

The term “Serverless” is also interesting. There are servers in Cloud and they involve in every operation in computing but with respect to user, we find that users (or application developers) just upload the code in to application without considering the settings and profile of host machines (either virtual, container, bare metal or physical machines). That’s why we called it “Serverless” [3].

As phrase “Serverless” explains the relationship between consumer and Cloud, we can see that all functions are incorporated inside application. Furthermore, the function also increases the granularity of Cloud utilization which impact the billing method. Now the Cloud monitoring systems are enabled to find the resource occupation by function and can measure the utilization at milli seconds (normally 100ms scale). It is almost a real time monitoring and bill generation is more precise and nearly true for end user [1], [4], [5]. This capability provides an extra ordinary economic benefit to Cloud computing. Utilization based billing override the provision-based billing where user have to pay for the resources which are provisioned to it. There are many cases where incorrectly estimates its requirement and demand extra resources and these resources not utilized but user has to pay for them because Cloud infrastructure was unable to identify the resource consumption. Serverless shift the scenario to actual consumption-oriented billing that the actual demand from user and consequently, the Cloud more affordable solution.

1.1. Features Of FAAS

The utilizations of FaaS are long list and almost all use cases of conventional Cloud computing are covered in Serverless. Here we will present some the features which commonly discussed in literature.

- **Functional Programming:** We know that function is the basic building block of any application regardless of programming language. This is main component that acquire the domain in main memory and utilize the CPU cycle. Moreover, it is the entity that call storage and network for consumption. While other computing usage methods are concerning with machines (real or virtual), the FaaS only concern with execution nevertheless of machines. This feature made FaaS is realistic and intuitive and Cloud users feel easy with it [4].
- **Microservices:** Microservices is method of developing and deploying large applications on Cloud with independent units. Conventionally, application development is performed in form of a complete package where common

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

programming framework, run time libraries and supporting tools uses. This method called monolithic architecture while microservices architecture based loosely coupled components which integrated with each other with APIs. FaaS in intent implementing microservices architecture [6].

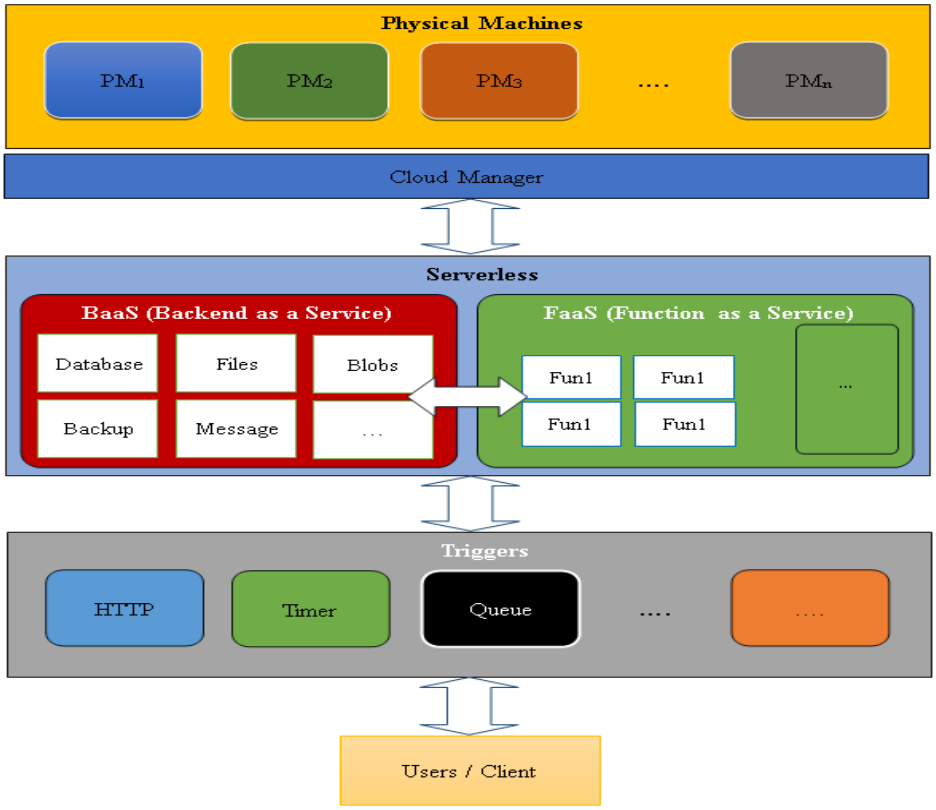

Figure 1: General Architecture of Serverless Cloud

- **Decoupling:** As mentioned earlier that functions are independent from underneath services, this is also made free the deployment to couple with attached auxiliaries such as operating system, protocols, hardware and connection [7]. Now application developers put their all emphasize of application development and business logic. Serverless inherits fundamental property of scaling from conventional Cloud computing so there is no problem with elasticity of resources with newly uploaded code [8].
- **Stateless:** In old fashioned applications, system tries to remember the communication between components. It may maintain some session or some other things to setup a state among the participants but FaaS is stateless. This is a great idea because when a function completes its execution than all memory footprints washouts and lot of memory is available for upcoming functions and also reduce the overhead for host management system maintain a state [3, 4].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

- **Function Hosting:** Function execution is performed on some host but users have no direct interaction with server. It removes the managerial expenditures and make development and deployment more cost effective. The server execution also include in conventional Cloud computing Serverless reduces this expense head too [3, 7, 9].
- **Language Independence:** Normally consider that applications are written in a single language but Serverless removes the language barrier too. Now developers can program the different functions of a single application into different languages according to their connivence and these components interacts each other with some transportation languages such as JSON or XML. Language runtimes already provided by host and if some language support is not available in default package, then will be add on minimal amount [7].
- **DevOps Support:** Since 2007, DevOps is getting popularity and first choice of application developers and IT managers. Serverless architecture, by default has the support to DevOps due to different features such as language independence, function hosting and statelessness. DevOps is complex procedure with multiple cycling steps and can be easily manage in a Serverless environment. Development procedures may be automated by using CI/CD pipelines that integrates the development, testing and deployment in seamless way [10-13].
- **Service Consolidation:** Service consolidation or migration is also the basic features of Cloud. When host system identifies some imbalance or over utilized nodes, then for internal management it can shift the running procedures to some other physical node which is less occupied. Serverless can also migrate the FaaS if it faces some disturbance because overall architecture is rather light weight [14-16].
- **Support of other Cloud Models:** We discussed three other usage models of Cloud computing, VMs, containers and bare metal machines. All these methods can also easily manage with Serverless [17, 18]. In such a case that user wants to deploy a Serverless model over a virtual machine or container or even bare metal, then it can be work. In this scenario, FaaS will not directly interacts with host an abstract layer will be ensemble between them to produce require model.
- **Reduced Computational Workload:** Functions are inside an application and applications load only require function which are requested to be execute. As operating system are intelligent enough to maintain the good scheduling and dispatching the binary for processing, the overall Serverless environment less overcrowded. Consequently, the overall computation procedures reduced and resource requirement also minimize [3, 4, 14, 17].
- **Billing Mechanism:** Commercially available Cloud system charge their clients to use their infrastructure. Cloud architecture allows vendors to produce bills variably because there is a separate monitoring system continuously logging the utilization of resources. To make Cloud affordable to clients, vendors generate bills on base of consumption not the resources provisioned to client. Serverless architecture is more

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

granular because it is easy to record. Normally, Serverless monitoring system use the 10ms slices of logging the execution of a function which enough to real time costing. As unit of time smalls, the users are more satisfy with expenditures because they are paying what they are really using [14].

- **Workload fluctuation:** Serverless workload is unexpected and uneven. System cannot foresee the volatility of function calls in future because there is no machine (servers) involve in it. All the applications are running directly and any time their frequency of invocation change. Either it may exceed or decrease. This phenomenon adds a challenge FaaS about elasticity and resource management [16].

Academia and industry both are working and exploring more aspects of FaaS

1.2. Triggers

Triggers are the events which invoke a function. Every Cloud vender defines a set of triggers that provide an interface to user for invocation of function. Every trigger has parameters that use as a payload of a function. Whenever consumer desire to execute a function, it generates an event or trigger for example, a user want o store data in database using REST API call. It will generate an HTTP trigger that take data from client interface (mobile, web, IoT or event some other Cloud output) and put it into some JSON or other format and pass to API for further processing. A function may be calls some other function using triggers [19].

When a trigger invokes a function, it produces a binding with this function. Two types of bindings, input binding and output binding. User can also add both of them to a function or even if a no binding requires than it may be left blank. For instance, user want to read message from message queue then he will generate ‘queue’ trigger that invoke the function to read the message. This message contains the logic of reading mechanism. In this case function don’t require any parameter or data, thus, no input binding will be attached while it returns the message which available in queue so its output binding will be ‘message queue’. Similarly, if a user wants to read data from storage and put it into Cosmos DB with a REST API, then first he will generate an HTTP trigger with ‘storage’ input binding and output binding is Cosmos DB.

1.3. Function Life Cycle

For complete understanding of working of functions, it is important to understand the life cycle of function. Normally, FaaS functions called in stateless environment with no prior footprint in memory if function is already called. It means that function has to get its resources every now and then each time. It is called cold start. Sometimes, Cloud orchestrator remain loaded the function resources after execution in a sandbox. If resources of function are still remained sandbox and function utilize these resources on next turn, then this is called hot restart. The decision of maintaining state of a function is critical and require to discuss with comprehensive details but first we will explain the function life cycle.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

1. **Start form Idle State:** A function will remain in idle state until it called. When a function uploaded first time in Serverless, it become the part of application. When application executes, its functions also move from idle state to execution state but not all function start execution, only such function executes which are invoked by some trigger.
2. **Acquire Resources:** When some trigger invokes the function, then this function start execution. The step of execution to prepare resources for this function. Many times, this a concurrent operation that is running asynchronously while function code is loading into memory but many resources require on runtime. Resource acquirement is a complex task that depends not only code written for this function but also libraries, frameworks and runtimes. Furthermore, the resource requirement is rather dynamic operation because Cloud also implements the scalability and it also effect the monitory aspects of Cloud.
3. **End of Execution:** When a function completes its execution then it will go to again idle state. Here again, the idle state may be in parking area or sandbox. If function is in the list of favorite functions, then it will go to sandbox. The orchestrator also saves the resources such as files, network ports or other data into the sand box because when function call again then it requires to use these resources and saving these items in some cache, reduce the turn-around time. If a function is saved in sandbox after execution than it is ready for hot restart and if function is not in the list, then it will vanish from memory and on next turn, it will be cold restart.
4. **Sandbox storage:** Cloud orchestration develop a strategy called keep-alive policy that decides which function should be treat as stateful. It may be such functions which are frequently invoke, or demand bulk of resources or scheduled with shorter idle span. These functions stored in sandbox which work like a cache and not only the function code is keep-alive but also its resources may also archive.
5. **Hot / Cold Restart:** If a function is loaded from sandbox or hot restart it looks very efficient to restart working because its turn-around time is very much shorter. It may take its resources from sandbox which also added support. This looks an iota of time which is discounted from other Cloud operations but on a larger scale, it is very helpful, specifically, in terms of QoS, QoE and network consumption and consequently, user satisfaction.
6. **Monitoring:** All processes including, loading, resource acquisition and utilization is monitor by a separate system. There are two benefits this operation. First is to managing the scalability [13]. There are two levels of scalability as performed by Cloud management. First is vertical scaling, in which Cloud resource demands changes normally. For instance, a function in execution state and demands 10 percent of increase in any resource which is nominal and management adds some extra resources to manage this demand. It may be possible that demand gigantically, up to hundred times or even more, to cope this situation orchestration may employee other machines or nodes. This is called horizontal scaling. It is also important that

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

scaling just not belongs to increase in demand, it's also including shrinking. When a function leaves resources then monitoring system capture these resources and put them into an idle pool and further provide these resources to such functions which growing or loading.

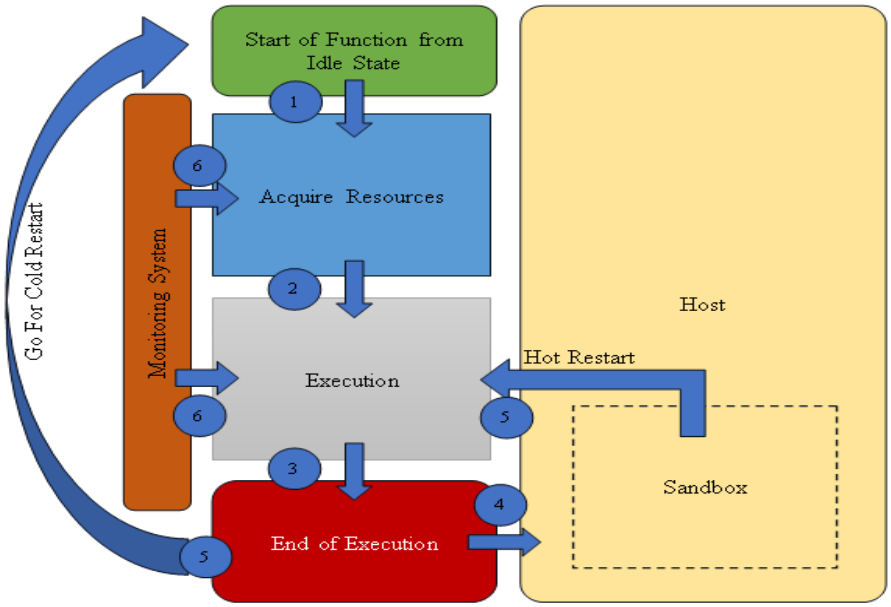

Figure 2: Function Life Cycle

1.4. Challenges In FAAS

Serverless is robust and simple that's why it gaining popularity in comparison to other Cloud infrastructures but the meanwhile its challenges and limitations are still there. Let's have look on these challenges because these are discussed and solution of some these challenges are defined in this document [20].

- Lazy Start: We mentioned earlier that there are two kinds of starts available in Serverless. First, is cold and other is hot start. If a function direct start without and context then it is cold start but if a function was selected in keep-alive policy then it will start from sandbox and called hot start. The both methods of start (or restart) have their own pros and cons. If a function always starts in cold then, it will take time and dent the user satisfaction and consequently the revenue of business. A hot start is the solution of issues in cold start where function is already in preemptive state with its resources and looks very early to start working but there is cost of sandbox also here. Host system has to setup a cache like structure where functions can reside during their idle time. Host cannot put all functions which are waiting for next call because this will demand extra resources.
- Keep-alive Policy: Which function will be available in sandbox to reduce the restart time is calculated under the keep-alive policy. How to define this keep-alive policy is also critical task. This is double edge sword because if there are glitches in this policy

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

then profit of hot restart can be convert to lose. If incorrect functions selected which are not require then they will occupy the space of potential functions and therefore system resources will be lost. Another issue is, calculation overhead. If policy consuming too much computational resources than it is also damage the quality of service. It is also important that this policy work around continuously because it will record the all activities of function performing and generate a selected set of function which are included in sandbox. Policy calculation should be smart and lightweight that cannot haunt the normal operations of Cloud infrastructure.

- Caching: Caching is a common strategy for Serverless to reduce the cold start. For example, SOCK is one of the solutions developed by Oakes [21]. This is container framework that reduced the cold starts in Serverless. A customized lean container is introduced that prevent the latency. The proposed solution targeted two main causes of latency, first, the language runtime initialization latency and second is container initialization latency. Their solution emphasizes on Docker isolation. For this purpose, they proposed more resources to store the data of functions when function sleep after execution. Similarly, Windows Azure Function focus on shared libraries and SUESS depends on snapshot to minimize the latency.

We only present the challenges which are concerned research and area of interest

2. LITERATURE REVIEW

FaaS is rather new paradigm in Cloud services but handsome amount of work is produced about the challenges of Serverless. Here we will display a brief overview of proposed solutions to reduce Cold Start issue in cloud computing. The researcher community developed solutions with different methodologies including machine learning, prediction, time series analysis, and statistical inferences.

Hybrid histogram-based policy generation was developed by Shahrad et al [22]. The collect and explain the dataset “AzureFuncitonDataset2019”. Two main goals they attempted to achieve in their research are characterization of workload and developed a policy that reduces the number of cold starts. The data collection was performed during 15th July to 28th July, 2019 and the record the function invocation frequency per minute, triggers of function, execution span and memory consumption. They identified three fundamental observations from dataset. First, the most of the functions have execution span as almost equals to cold start included, the second observation was about the fluctuation invocation by applications, some applications are invoked very often while most of the applications remain calm and very rarely called. Similarly, the prediction of next invocation of an application is also challenging. After identification of challenges and goals to achieve, the defined their methodology that check out the out of bound (OOB) function invocations, if the idle time of a function is exceeded of some preconfigured time, then it will label as OOB function. If a function display too many OOBs then it will treat by ARIMA to predict its next invocation otherwise, system will analyze its invocation pattern and if it is utility is according to some decoration, it will be treated by histogram method.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

If function shows no pattern, then it will be managed by standard policy generated by host. The results are method implemented were significant in some cases. 75% applications reduced their cold start up 50% but as the keep-alive policy extended to 1 hour, it reduced to 25%. The impact of memory wastage also reduced and up to 50%. The system was implemented on OpenWhisk, an open-source implementation for FaaS workload and all the evaluation is performed on said system.

Hossein et al [23] proposed a heuristic oriented approach for function scheduling method to reduce cold starts. This method implements dynamic waiting technique that can change the waiting time on the fly in mutable scenarios. Four types of decisions were implemented that are reasons of change the waiting time, consequences of the technique were noteworthy, they showed their method improves 32% in comparison of fixed time method. Six main features use to generate policy, score of previous executions, degree of collaborations, maximum merging time, score of correct merging, cost of the environment and timeline. The proposed approach evaluated on three criteria. First the response time that measure between trigger generated for execution and start signal from function, second, the turnaround time that span between start signal to end of execution and third is operational time that use to perform overhead calculation.

In [8] proposed a general purpose abstraction layer for application deployment. A ServerlessOS is introduced that have three major parts. First a disaggregation model that disintegrates the abstraction but allows resources to migrate one server to other while second component is an orchestrator that granularly manages the available resources and support local and global decision making. An isolator also incorporated that implements the data and resource isolation over the servers. These three components of SeverlessOS are heavily impact the performance of Serverless environment. First it decouples the physical resources like memory, CPU and IO. Grass root level orchestration is important because this is significant in management decisions. The provided orchestration is on detailed level and can be perform local decision belongs to host hardware and higher level such as nodes deployment. The proposed solution also implements the isolation with data privacy and multitenant resource management.

SEUSS [24] is a framework that increase the Serverless execution using limiting the restart. The system improvised both speed and memory. The improvisation was achieved using rapid deployment and high dense caching. To reducing cold restart, the arrangement of unikernel snapshots was maintained which directly deploy the function to execution rather cold. To fix memory issue, page level sharing on software stack was implemented. The results were significant. Deployment time of function reduced up to ten times or even lesser, and the overall system throughput improves to 51x. Their approach built on four foundational elements, first is Unikernerlization. This is responsive caching that can be transform to some host without changes. The snapshot that is immutable data object which reflects the state of execution. In fact, snapshot work as a template which helps to initialize other operations on base it. Multiple operations can be generated from snapshot. Third is anticipatory optimization. This reduced the extra space and execution resources of current execution. Finally, the snapshot stack, that enables the

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

communication between different snapshot images that are executing in parallel. The main purpose of snapshot stack is to increase the number functions in cache which have common base provided by snapshot image.

Another cache-oriented solution Faa$t provided by Romero et al [25]. They defined their cache on base of anticipatory sizing of the cache. A transparent cache is created for each application that store the function data inside it to reduce the number cold starts. At the end of application execution, the memory is empty but on restart, this cache helps to prewarm the functions. Faa$t is also elastic that can be add or remove items and increase / decrease its footprint according to situation. They implement their solution in Windows Azure Function to show its practicality. The results showed that performance improvement about 57% in average. Storage items and blobs are the primary target of Faa$t because, these items not only lethargic in loading but also costly in bandwidth consumption. It allocates a region in where the cache data stored. It declares demon services that bring data from remote storage and resides in cache until the application is running. This cache is located in container or virtual machine which is hosted on some physical machine. Each application has its own cache called Cashelet. Every time when data is demanded the cache, search it out in its repository and if it found than marked it as cache hit. If did not find than this is cache mis (old terminology). Memory daemons monitor the cache and data all the time to scale it accordingly.

Machine learning a responsive method that can produce dynamic decisions. A machine learning technique is deployed by Djob et al [26] with name of OFC (Opportunistic Cache System). They target two main issues, first, the overprovisioning of resources and second is unnecessary utilization of sandbox to avoid cold start. They employed decision tree method to true prediction of requirement of resources. The overall architecture is inherited from three available infrastructures. OpenWhisk that provide basic functionality of FaaS service, Swift, a storage mechanism and RAM-Cloud an in-memory storage mechanism. The implementation is simple and intuitive, the algorithm finds the input of a function and predict about required memory. In first step when a function invoked, the controller will inquire about the memory requirement from predictor. The predictor not only declare the memory but also the decision of caching or not. If a function is caches, then it means that this function is take too much time to reload and significant throughput will be earned in case of caching it. The predictor is consisted on four basic operations. Model updating work in two cases. First time, the model is blank and unable to make any decision, this time model will be update to perform its function and model updating also require when it perform any incorrect decision. Model outputs are defined in two terms, the amount of memory required and cached decision. Model inputs are the specifications of the function which is invoked. The model calculates the amount of memory that it required and also the calculate the cashing decision. Final operation is prediction speed. The system showed overall 82% improvement in single stage and 60% in pipelined functions.

An automotive keep-alive caching is defined by [27] for variable caching size. They studied the cache hitting and missing. They implement their mode for Hawkes processes to find the optimization of policy. As Hawkers processes are past dependent, they also

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

introduced a history independent model. They testify the Hawkes processes. These are such process which self-discard and define to intensity of sequence of events. They defined empirically identify the window of keep-alive functions. Conventionally, this window is defined by the analyzing of historical background of process but they discover that average also work finely. For evaluation, they taken 600 points in function invocation to mark them as Hawkes process than calculate the mean of 100 policies and finally the calculate the cost of keep-alive and cost of missing in cache.

FaaSCache [28] based on greedy dual keep-alive policy. This policy is had impact on limiting cold start up to 3 times in contrast of contemporary practices. Caching concepts including as reinvoke span proportion found in cache can also be used for server resource allocation and scalability. It is also studied that this policy is able to minimize the resource provisioning of Serverless up to 30% for real-world variated functions. they enforced caching-based keep-alive and resource allocation methods in FaasCache system, which is implemented in OpenWhisk.

Chen et al define [29] a special framework for edge computing in Serverless. The offered an online distributed mechanism which specifically targets the size, number of cold starts and invocation frequency. They compared their method with fixed cache policy and histogram-based policy.

To reduce the container delays and overprovisioned memory [30] with the name of μFuncCache. This method did not disturb the internal architecture of host and other Cloud management system. Every element has its own cache which completely independent to all other elements. Every request first checks the demanded item in cache and if it is not found than it will demand from other resources such as storage or database.

Palette load balancing maintain such functions which are waiting for resources and just arrived for execution, while allowing the platform to place consecutive calling to simultaneous on the same host. They equate a proposed the Palette load balancer for a contemporary location aware load manager. For a FaaS-based application along a residual cache, Palette improvise the cache hit proportionality by six times. For a Serverless version of Dask, given mechanism upgrade run times up to 46% on Task Bench and TPC-H, respectively. While a serverless flavor of NumS, Palette update the run times by 37%. These improvisations showed a highly impactive the performance of serverful implementation of the same systems.

Snapshot of previous execution can reduce the cold start as an approach described by Paulo et al [31]. The system is defined for Linux tools that create some checkpoint/restore in host. They examine the prototype by performing experiments that equivalence its start-up time against the normal process initialization procedure. They investigate the three important aspects. First, a idle function. Secondly, an Image operation function while third, a function that apply on Markdown data items. The results were significant that improved the initialization time of function copies up to 40% in worst case of an idle function while second case it jumps to 71% for the Image operation. Detailed examine of system unleashed that the runtime startup is a main issue, and it is confirmed by performing a

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

sensitivity analysis based on hypothetically produced operation of multiple function sizes. These evaluations present that it is vital to make decision every now and then to develop a snapshot of a function. When one snapshot is created, function became in warm state, and then boost in startup achieved by the prebaking technique is even higher. The velocity increases from 127.45% to 403.96%, for a small, testing function, and for a bigger, artificially generated workload, this reached to 1932.49% from 121.07%.

Lin and Glikson [32] developed a pool of warm containers that help to reduce the cold starts. Their method to resolve the cold-start is is to develop a collection of warm pods that called pool. This pool will be ready and immediately available for code (function) as demand is growing to eradicate the cold restart time lapse by assigning the pods which prebooked. This pool can also be distributed in different kinds of services that use the same function. The first describe the enforcement of a pool, the system based on Knative Serving. Knative is an open-source infrastructure to deploying FaaS workload. They examine the performance evaluation of the proposed solution compared to counter parts such as Knative without a collection or pool of pre-warmed Pods, and elaboration of results and insights.

An extensive solution is developed by Sethi et al [33] to manage the notorious reason in Cloud, the cold start. They presented a management method to eradicate the cold start invocation by administrating the containers warm for a larger span of time with the Least Recently used warm Container Selection (LCS) approach on Affinity-based arrangement. In addition, they also performed an evaluation and compared the obtained results with the Most Recent Used (MRU) container method. The developed LCS method showed much better performance than up to more than 48% against the MRU conventional method.

FaaSLigh [34] is an application-based method to accelerate the function loading by Lie et al. FaaS applications by application-level management. They initially perform a evolutionary research to discover the primary issue of the cold-start contention of FaaS. The research elaborated that application function initialization delay is a painful enough for Cloud users that involved extra overhead for Cloud delivery services. Hence, initiating only vital function(s) from applications can be a worthwhile approach. According to on this citation, they discover function(s) associated to application functionalities by producing the function-level invoke graph and disintegrate other function including optional functions. By disintegration of extra function, system can accelerate the performance and loaded on demand to avoid the incorrect discovery of crucial code reason behind the application failure. Specifically, an important guideline the design of FaaSLight is general in nature, such as programming language and platform independent. In real world, FaaSLight can be impactfully enforced to Serverless applications developed in multiple programming languages, and can be intuitively deliver on available FaaS platforms such as Google Cloud Functions or AWS Lambda or anyone else, without having to change the base operating systems, hypervisors or host, similarly, without any extra techniques or efforts for application developers. The testifying results on actual FaaS applications showed that FaaSLight can clearly reduce the function starting delay up to 78.95% with

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

average of 28.78%, hence avoiding the cold-start latency. They explained the result, the total response delay of code may be reduced to 42.05% with average of 19.21%. In comparison with the contemporary, FaaSLight achieves 21.25 times better in limiting the mean total response delay.

Shen et al [35] present a novel approach name Defuse. This is dependency-based function scheduler for FaaS platforms. Specially, this approach discovered couple of dependencies between FaaS, such as strong and weak dependencies. It employed frequent pattern explorations and useful information to identify including dependencies from function invocation monitoring. By this method, Defuse deduced a function dependency graph. The associated components such as relative functions on the graph can be manage to eradicate the invocation of cold starts. They testified the implications of Defuse by implementing it to n commercial FaaS dataset. The evaluation results displayed that Defuse can minimize 22% of memory footprint while having a 35% decrease in function cold-start rates with the latest methods.

Pigeon [36] specifically designed for private Cloud base FaaS and Serverless. This provides a method for enterprises to manage applications. Pigeon develops function-oriented FaaS framework by presenting a sovereign and granular lowest level resource manager over the of Cloud orchestration tool Kubernetes. A new overprovision oriented rigid pre-warmed container solution is the part of approach that generously reduce function startup delay and improves resource discovery mechanism for short term Cloud functions. The results present that Pigeon framework improves function cold-start trigger rate by 26% to 80% with respect to AWS Lambda Serverless platform. Relative to Kubernetes own orchestrator, the performance obtained 3x upgradation for handling short term functions.

A reinforcement learning oriented method is propose in [37]. The presented Q-Learning agent work with the Kubeless, a Kubernetes based tool for function deployment in FaaS, to manage serverless platform by disintegration the environments, operations and results with the use of CPU utilization by instances, available code items and success or failure rate of response. The overhead is regenerate with the Apache JMeter non-GUI toolkit and agent is examined in comparison of the default auto-scale feature of Kubeless. The agent presents the capacity of learning the invocation detail, to develop decisions by generating the optimal number of functions for the given time span of learning, under controlled environment settings.

An implementation of application knowledge is used to minimize the cold starts in Serverless [38]. proposed approach has fundamentally concentrated on limiting the span for cold starts. In this treatise, they studied and applied three methods, in comparison of related studies, that minimize the number of cold starts and treating the Serverless as a black box. In the methods, applied as part of a lightweight pattern middleware, they implement knowledge on the construction of functions to initiate cold starts and, therefore, the assignment of new containers before the trigger and related function calls the associated application. In evaluation on AWS Lambda and OpenWhisk, the identify that

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

proposed method eradicate with the mean of approximately 40% and even in some cases goes to 80%, of all cold starts while causing only a small cost overhead calculation.

[39] presents two-layer approach that implements two-layer adaptive method to manage late start. The first layer consumes a comprehensive reinforcement learning algorithm to find the function invocation outlines in given for identify the best time to manage alive the containers active, while the second layer is intended foundation for a lengthy short-term memory (LSTM) to ascertain the function invocation times in next to find the availability of prewarmed containers. The evaluation of results on the Openwhisk platform displayed that the designed method minimizes the memory footprint up to 12.73% and improvised the execution invocations of functions on afore available containers by 22.65% complementary to the Openwhisk platform.

Rafael et al [40] developed novel approach to reduce the cold start in FaaS, they study the implications of container pre-warming and function reinvocation on both application execution delay and resource utilization, with a conventional data by a machine learning application for image processing (recognition) with multiple supplied data patterns. Thus, they designed an expansion the typical centralized cloud-based Serverless service to a two-layer distributed Edge platform in collaboration with Cloud to brought the framework proximity to the data source and eradicate network delays.

Temporal Point Processes (TPPs) [41] developed to minimize the function invocations in FaaS compositions with cold start. A probability distribution by time span and category of the function invocations according to background of invocations, is predicted using these probabilistic models. The anticipation can avoid latency by scaling application in preemption and minimize network load by maintaining the function-server provision. In this scenario, they designed an application in python named TppFaaS over the OpenWhisk. The approach consumes the neural TPPs LogNormMix for creation of model to the time using a log-normal mixture distribution and TruncNorm for anticipating a number for the time. They also develop a data collection tool for OpenWhisk traces that implanted into TppFaaS and produce datasets for different FaaS configurations to train and examine the produced models. For datasets without cold starts, the models achieved for most compositions a mean absolute error below 22ms and a percentage of correctly predicted function classes above 94%.

SARIMA (Seasonal Auto Regressive Integrated Moving Average) [42] is a time series based solution, for forecasting of models to predict the time at which the next function arrives, and accordingly growing or diminution the number of demanded container(s) to reduce the time wastage, therefore, minimizing the function initializing time. They also implement PBA (Prediction Based Autoscaler) and analyze it with the already given HPA (Horizontal Pod Autoscaler), which arrives with Kubernetes. The evaluations displayed that PBA performs clearly improved than the default HPA, while reducing the wastage of resources.

The work [43], the target the cold start issue of the FaaS framework. They introduced WLEC, a container orchestration structure to reduce the starting delay. WLEC utilize an

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

updated S2LRU architecture, called S2LRU++ with a newly updated third queue. they enforced WLEC in OpenLambda and examine it in both AWS and virtual machine settings with six different metrics. Moreover, to one image resizing application. In different improvisations in all metrics, 50% reduction in memory utilizations complementary to the complete hot approach and 31% mean of start span minimize compared to the no-warm manage are the most significant ones.

Adaptive Warm-Up Strategy (AWUS) developed by Xu et al [44] to forecasting the code invoking time and ready the code, therefore, minimizing the cold start delay. They utilize the function chain model to build the AWUS. The present the model to adopt a granular regression method to forecast non-first functions in the function structure accurately. Secondly, they presented an Adaptive Container Pool Scaling Strategy (ACPSS) to minimize the function invocation time. That autonomously regulate the capacity of the container pool to minimize the resources consumption. The AWUS and ACPSS work along to minimize the delay. Lastly, a FaaS platform-based evaluation conducted to examine they policy. The results explain the effectiveness of proposed strategies.

[45] presents a container lifecycle-aware management policy to orchestrate Serverless, CAS. The fundamental concept is to manage the dividing of requests and identify creation or ejection of containers according to multiple lifecycle levels of containers. They applied a initial idea of CAS on OpenWhisk. The experiments presented that CAS minimize 81% cold starts and thus, brought a 63% minimization at 95th percentile delay in contrast with available scheduling policy in OpenWhisk where is worker dispute between workloads, and does not add vital performance overhead.

Xanadu [46], a cascading cold starts in Serverless, function deployment. First, developed the base and extent of the cascading impact in cold start situations in different commercial platforms and cloud vendors. Towards justifying these calculating overheads, their proposed model and present several optimizations, that are built into their tool Xanadu. Xanadu displayed different initializations options oriented on the runtime isolation requirements and provides code chaining with or without predefine workflow features. Xanadu's optimizations to improve the cascading cold start problem are generate on speculative and just-in-time prebooking of resources. Our experiment on the Xanadu system unleashed approximately complete eradication of cascading cold starts at lowest cost overheads, extra ordinary performing the given contemporary platforms. Even relatively smaller workflows, Xanadu minimize overheads by almost 18 times compared to Knative and 10 times in contrast of Apache Openwhisk.

The present [47] an detail evaluation of cold start in the FaaS framework and present HotC, a container-based runtime orchestration framework that improvised the lightweight containers to alleviate the cold start and improvised the network performance of FaaS applications. HotC build a live container runtime pool, analyzes the consumer input data or setting file, and feed the available runtime for instant consumption. To perform better predict the efficiently and effetely orchestration the active containers, they developed an adaptive live container control mechanism method to combining the exponential smoothing model and Markov chain method. The results showed that HotC has such a

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

small overhead that can be consider as insignificant and can easily improve the performance of various applications with different network workload designs in both Cloud Edge components.

3. METHODOLOGY

In the following section, we will discuss the design of the solution for problem which is stated in section 1. After a comprehensive detail of the already proposed solutions, we are able to understand the root cause of problem, its features and attributes and spirit of issue. We have to define a method, approach or framework that divide the functions into the categories or hot and cold. For this purpose, we took following steps to define our solution.

1. Select a real dataset that consist on traces or logs of function in Serverless environment.
2. To categories the functions, identify the features of functions which help us to develop the categories of hot and cold.
3. Transform the dataset for accordingly fashion.
4. Apply classification algorithm to define categories for initial experiments.
5. Apply fine tuning method to increase the accuracy and precision on dataset and again perform the experiments.
6. Evaluate the results.

We divide this section into three subsections sections. In first sub section, we will explain the dataset and its features and describe how we will convert it into an input of classification model. In second part, we will explain the Naïve Bayes classification modeling algorithm and its details and in the final unit, we will show the fine-tuning method Kernal Density Estimation and discuss how effectively it can support the throughput of Naïve Bayes algorithm.

3.1. The Dataset

The dataset was collected by Mohammad Shahrad et al [22] in July 2019. This dataset is consisting on different random samples of applications, owners and functions. Total 14 days traces are recorded and stored into three kinds of files. These files belong to function invocation, CPU utilization and memory consumption. The basic need of these files to store the data of each application, owner, trigger and function. By a deep analysis of dataset, we will be able to develop a solution that can minimize the cold start. Obviously, the dataset has no information about cold start because execution span is separate the hot and cold start but we are still able to identify such instances which heavy in workload and by caching them can increase the throughput of Cloud service. Attributes of dataset are following.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

Invocation Frequency Table: First attribute is function frequency bins per minute. There one file for each day (total 14 files) with specific file naming convention, invocations_per_function_md.anon.d[01-14].csv, where number in bracket shows the day number which data is collected for example invocations_per-_function_md.anon.d01.csv for first day of collection and so forth. This file contains five fields. First four fields are straight including owner, application, function and trigger while next 1440 fields are bins of every minute in 24 hours. These columns contain the invocation frequency in a minute. Following table elaborate the frequency invocation file.

Table 1: Invocation of function file structure

Column	Description
HashOwner	Encrypted and unique ID of application owner.
HashApp	Encrypted and unique ID of application
HashFunction	Encrypted and unique ID of Function
Trigger	Name of Trigger that invoke the function
1 to 1440	Number of invocations in each minute

Function Span and CPU Utilization: Here, again we have 14 files marked with each day by name function_durations_percentiles.anon.d[01-14].csv. This file contains the information about function duration and consequently, CPU utilization. This file is use to identify the utilization of resource in percentile. This is also a CSV file that contains the 14 columns. Some columns are identical to function invocation table but some are different.

Table 2: Function running Span and CPU utilization table

Column	Description
HashOwner, HashApp, HashFunction	IDs of owner, application and function same as invocation frequency table
Average	Average running time of function
Count	Number of runs
Minimum	Minimum running time
Maximum	Maximum running time
Percetile_Average_[0-100]	Percentile of execution. 0, 1, 25, 50, 75, 99, 100

Memory Utilization: As CPU utilization is recorded, similarly, the memory consumption is also logged and the file structure is almost similar. It contains HashApp and HashOwner to relate data with other two files while this file doesn't contain any data about functions. It works on application data. The third column is average memory allocation in MBs for this specific application according to specific file (24 hours data). Next eight columns indicate the percentile of memory allocation of 1, 25, 50, 75, 95, 99 and 100. This is not actual allocation but its average. This is calculated for 12 samples of in a minute after 5 second span and took weighted percentile. The weighted percentile is bit tricky to calculate. These the mean of invocations and not the number of invocations. These percentiles are computing on the base of weight of number of invocations which are observed on span of different timings. In fact, the percentiles are an attempt to display real utilization of CPU and memory.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

3.2. Naïve Bayes Classifier

For machine learning modeling we select Naïve Bayes classification algorithm. NB is classical family of classifiers that called Probabilistic Classifiers. These models based on applying Bayes' proposition with strong assumptions among features. NB set of models are robust, based on finite number of feature set and linear set of classes. These algorithms produce training by using the probability, that can be perform by evaluating a simple expression 3. This is also simple run time complexity which linear complementary to expensive recursive approximation used in many other classification models. In literature, Naïve Bayes classifiers are nominated by multiple identifications such as independence Bayes and simple Bayes 4.

Probabilistic theory is an event modeling theory where statisticians build a model that calculate the probability of occurrence of an event called Conditional Probability on the base of series of events $p(C_i|x_1, x_2, x_3, \dots, x_n)$ where C_i conditional event and x_1, x_2, x_3 up to x_n are events while p is the probability of occurrence.

The value of n might be larger enough but it should be finite. If we have a small set of events then the probability calculation process really efficient and straight forward. The complete Bayes theorem can be express as follows.

$$p(C_i \mid x) = \frac{p\,(C_i)p(x\,|C_i)}{p(x)} \quad (1$$

For multiple probabilities

$$p(C_i)\prod_{i=1}^{n} p(x_i|C_k) \quad (2$$

3.3. Kernel Density Estimation

Kernal Density Estimation is Curve Smoothing method for probability density estimation. In fact, this is non parametric method to find the probability density function. It works on the base of kernels and treat them as weights. It is also known as their inventors Parzen and Rosenblatt window. Kernal density estimation is can be blend with Naïve Bayes classifier. KDE helpful in acerating the conditional marginal densities in dataset and consequently increase the accuracy and precision.

As NB classifier, the KDE also assume that attributes are independent to each other. If feature set is defined as (x1, x2, x3, …, xn) and these independent and normally distributed values then we can define the KDE expression for this dataset as follow

$$\widehat{f_h}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=0}^{n} K\left(\frac{x-x_i}{h}\right) \quad (3$$

Here K is the kernel function that calculates the weights. These weights must not be less than 0 while h is the smoothing parameter. It is also referred as bandwidth. Kh named the scaled kernel. The value of h should be optimal. As bandwidth increases, the biasness of also increases and if value of bandwidth is narrow than it produces unexpected results. There are many types of kernels are available such as normal, Epanechnikov, biweighted or triweighted.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

We can see that different value of bandwidth produce different kind of effect in graphical representation of data. A general observation that as the value of h is increasing the smoothness of the curve increases. The above example ranges the bandwidth of from 0.1 to 1.0 on similar data and we observe that when bandwidth was at the lowest value than graph was very jaggy (the blue line). At the 0.3, we observe two distinct heights and a valley between them (yellow line). Similar, at 0.5, the curve became smoother, the height of peaks reduced and valley shorten its depth but still we can see that there are two significant heights while the value of 1.0, there is no peaks and we find a simple bell curve as normal distribution (red line).

There are multiple methods available to find the optimal value of h or bandwidth such as Scott Rule, cross validation methods, plugin methods, mixing rules but Silverman Validation is more efficient and the properties of our dataset are suitable to this method. It is also important all method to estimating bandwidth are based on minimizing the Mean Integrated Squared Error (MISE).

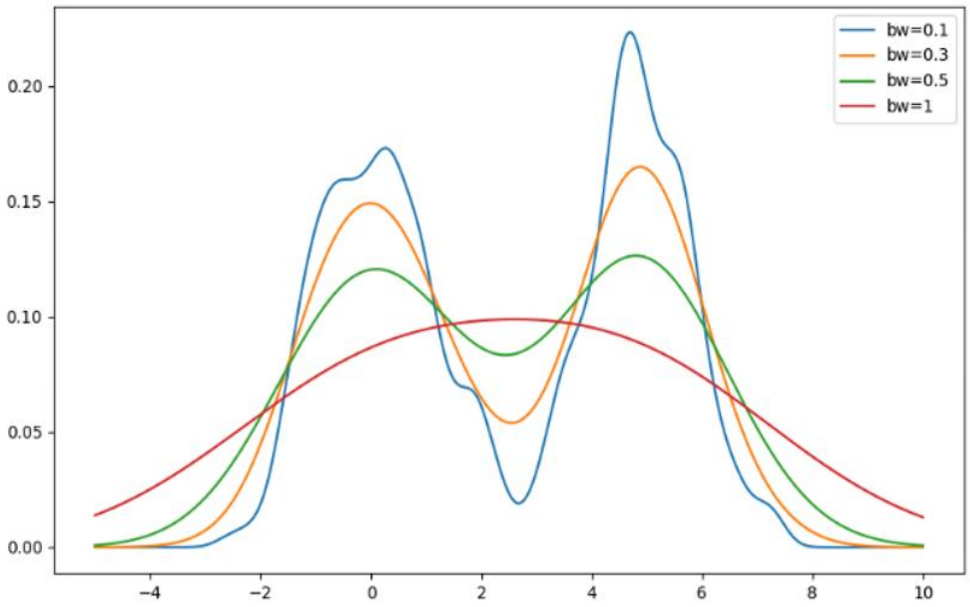

Figure 3: Different Bandwidths and their Graphs

Silverman method is important when we have no idea about the underneath distribution of data. If data is large enough, near to symmetric and using Gaussian kernel.

$$h = 0.9\, min\left(\hat{\sigma}, \frac{IRQ}{1.34}\right) n^{-\frac{1}{5}} \quad (4$$

Here $\hat{\sigma}$ is the standard deviation, n is the sample or population size, IRQ is the interquartile range that is between 25th percentile to 75th percentile while min is the method that choose the minimum value between standard deviation and quotient of IRQ and 1.34 (some literature also finds the value of 1.35). The main features of Silverman method are robust and less outliers.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

3.4. Classification

For classification of functions, we have to select such features of a function which are define it important. First, we have to discuss how a function become regular customer of keep-alive policy. Remember that some of the features in our dataset are discrete and some are the continuous.

Owner: This is the owner of application that might be some person or system that produces triggers for functions. Ultimately, the active owners will produce more calls and make the function a suitable candidate for keep-alive scheduling.

Application: Applications are the collection of functions. Whenever an application is active, it continuous invoke the function. Like owners, the highly provoked applications also be the reason of a function to be consider as motivated.

Triggers: Triggers are the main reason of invocation of function. Some triggers are very frequent such as HTTP other are really low. Most frequent triggers are also invoking too many functions. According to our dataset, the Timer trigger is the largest, just less than a half of all combined triggers.

Invocation Count: We can find invocation by two means. First, from invocation frequency table that is require extra computation while we also find the count from CPU utilization file. Both files may be produced some different results. To reduce overhead, we use utilization file data. The more invocation in number also increases the chances of invocation in future that is also a directly proportional to invocation probability.

Average: Arithmetic mean of the frequency of invocation is also important because it also support the high number of occurrences. As average is the representative value of all the time so the average will be use as a feature in classification operations.

Percentile: Percentile defines the utilization of resources. As the utilization of resources grows, the system should index the process for next call. A heavy process requires more resources to load for execution and increase the turn-around time and amplify the cold start problem.

Range: The utilization files provide us the minimum and maximum invocation over the time span. We can deduce the range which is difference between minimum and maximum. As the range increases, chances of an occurrences of event decreases.

Confidence Interval: In probabilistic models, the confidence level unleashes valuable information about the class of an instance. We use 70% to 80% confidence levels to ascertain the probability of a function that is an optimal number according to our data. Finding an optimal confidence level is not a straight forward method. It requires some extra experiments to find a usable number.

Kurtosis: We already explain the significance of Kurtosis that defines the tail of normal curve. We can employe it as feature of an instance that a thick tail of its time series data, defines a function have to chances of occurrences in next time bin.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

Skewness: Skewness is also a property of continuous data and we can also use it as the feature of the class. Skewness defines the orientation of data either the data is normally curved or skewed to left or right. As have high skewness (negative or positive), we have to take it as chances of outlier increases.

To build the machine learning model with Naïve Bayes, we implement the equation 8. There are two important points should be considered. First, the presented algorithm did not define the number of instances in testing and training but actually we use have to define these values in real implementation. For testing purpose, we execute the algorithm with multiple configurations. These configurations will be explained in next section. Second, the main output of the model is accuracy but other metrics also exists and compute those values too to support our thesis.

Algorithm1: Feature Selection	
Input: 1) Function invocation frequency table: *frequency_table*, 2) Utilization Table *utilization_table* **Output:** Table of functions with features *function_and_features*	
1	Read *owner_name, app_name, function_name, trigger, frequency_360_mins(list)*
2	Read *percentile* from *utilization_table* by *function_name*
3	Calculate following from *frequeny_360_mins* 1. *sum_invocation* 2. *average_invocation* 3. *range* ← *max – min* 4. *confidence_level* ← *FindConfidenceLevel(frequency_360_mins)* 5. *kurtosis* 6. *skewness*
4	Prepare *function_and_feature* table of all variables which are obtained from tables and computed
5	return *function_and_feature_table*

The above-mentioned algorithm prepares the data that use for classification. Our next step to develop an algorithm that produce actual classification. This algorithm will use result of the Algorithm 1 as input dataset that contains the descriptive information. The algorithm 2 implemented in two phases, first without KDE and then with KDE. In first phase, the algorithm produced results with low accuracy. The results of first algorithm are described in section 4. As the results of algorithm where KDE was integrated, we observed significant upgradation in accuracy and precision.

Algorithm3: Building Model with Training Set	
Input: Training dataset *training_set* **Output:** Generate a model	
1	Load *training_set*
2	Separate *discrete_features* and *continuous_features*
	Initialize *prior_probabilities*
3	For each *feature* in *discrate_features*
4	prior_probability ← count (*feature*) / count (*discrete_features*) add prior_probability to prior_probabilities
5	For each *feature* in *continuous_features*

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

	mean_feature ← sum (*feature*) / count (*feature*) *variance_feature* ← var (feature) Add mean_feature and variance_feature in list
6	Load *testing_set*
7	For each *instance* in *testing_set* For each *feature* in *instance* If *feature* is in *discrete_features* Computer probability using equation 1 and add to list of probabilities Else Compute probability using equation 8 and add to list of probabilitie Find the maximum *probability* and its class and add to *classified_list*
8	For each *instance* in *classified_list* If class of instance is similar to corresponding instance in *testing_set* Increase in *correct_count*
9	*accuracy* ← correct_cout / count (*testing_set*)
10	Return *accuracy*

4. RESULTS

After a detailed review of existing literature and defineing the problem solution, we have to examine and evaluate the solution. For this purpose, we have conducted a series of experiments. The experiments are not only have to perform to ascertain the final results but also find some critical values which require to fine tune the performance and metrics specifically, the bandwidth h and confidence interval. In following section, we discuss these experiments and analysis of results of these experiments.

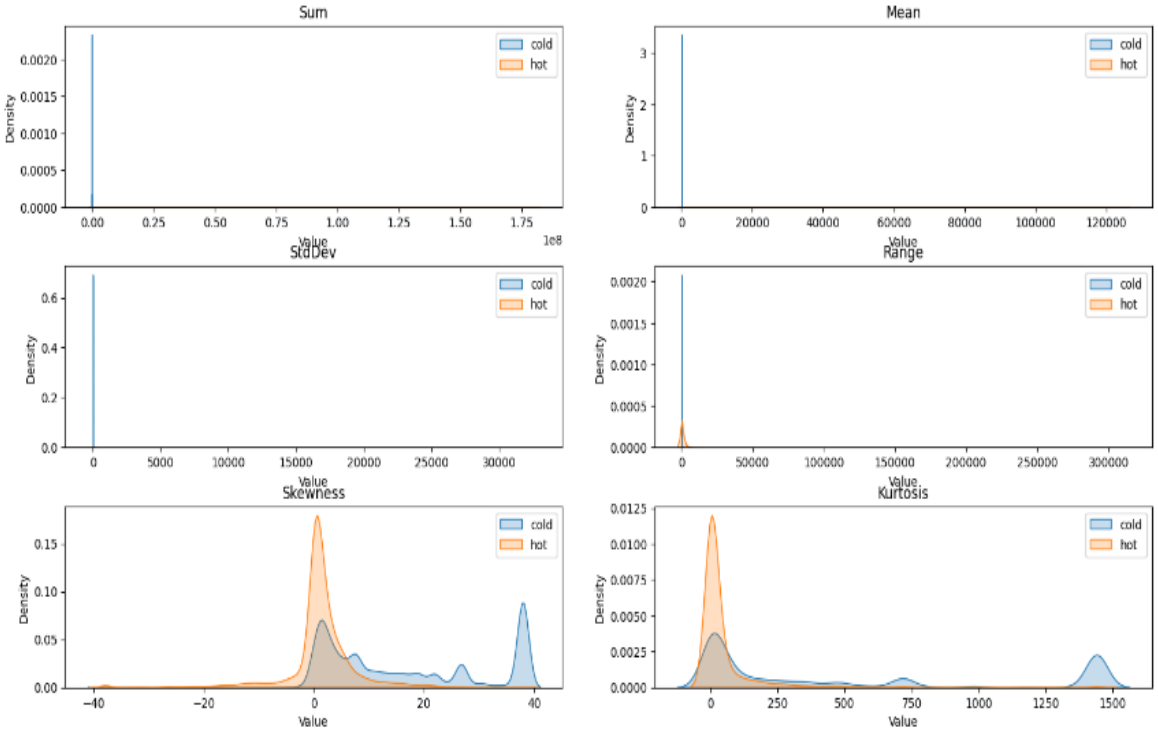

Figure 4: Density Graphs of Continuous Features

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

To find the diversity and density of features, we extract the density plots of each continuous feature. Density plots or measurement provide us the distribution of given feature values over the defined kernel. The shape of the feature normal graph shows the orientation, trustfullness and importance.

We can see that Sum, Mean, Standard Deviation and Range values single modular while Skewness and Kurtosis are multi modular. If we examine the structure of graph of first four feature, we found that they are homogeneous and most of the values lays around a central value. On the other hand, these curves are not very much symmetric that indicates that they are skewed to right. It is also very clear that these curves are clearly distinctive with respect to assigned class. The curves of Cold is too much vertical with also most not tapper in first four features while the Hot class almost zero bulge and long tapper goes with x axis. The orientation of first four features clearly indicates that classes are clearly distinitive and can be identify the by Sum, Mean, Statndard Deviation and Range features.

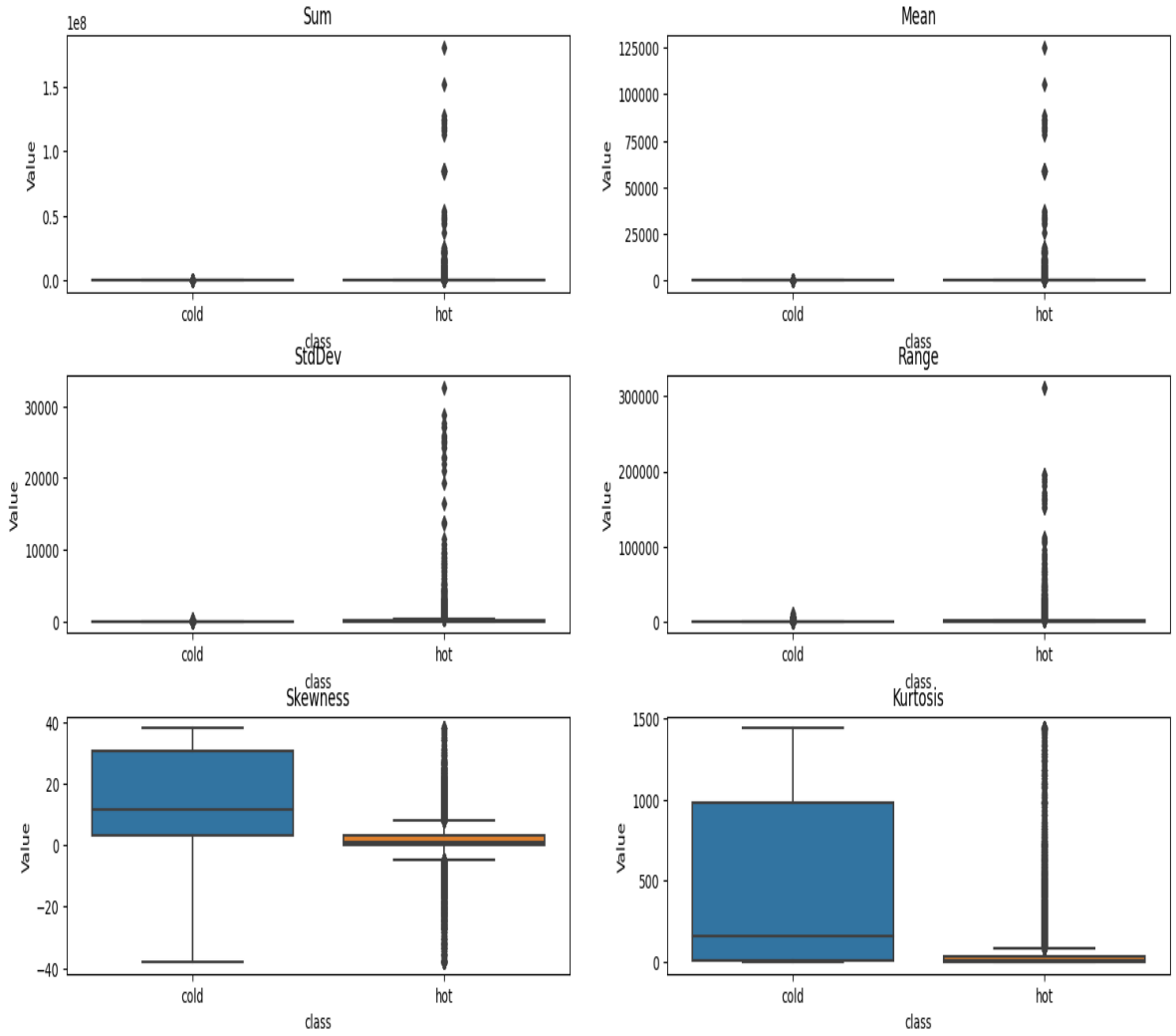

Figure 5: Box and Whisker Graph

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

Skewness of the graph show some significance in both classes. Cold class is rather more prominent than Hot in Whisker. We can also see the breadth of Q3 in the graph is very larger in Cold. It seems that more number of items are occurred in this segment while the box size is much small in Hot class and indicates that low number of instances. We can aslo observe that the location of graphs are overlapping. Here is clearly that some of the instances may be overlapped by other class and cause misleading classification.

As skewness, the Kurtosis Box-Whisker also significant in Cold and Hot and the difference is also protruding. Kurtosis also indicates that most of items are presented in Q3 of Cold class which is a healthy sign to identify the related instances but, in the meanwhile, it also overlapping with Hot instances and also almost all Hot items are hidden with Cold instances. This phenomenon may also direct to wrong results.

A correlation heatmap is a graphical representation of the correlation matrix, where individual cells of the matrix are color-coded based on the correlation coefficient values. It provides an overview of the relationships (or associations) between multiple variables at once.

Correlation values typically range from -1 to 1. A value close to 1 implies a strong positive correlation, meaning that as one variable increases, the other also tends to increase. A value close to -1 implies a strong negative correlation, meaning that as one variable increases, the other tends to decrease. A value close to 0 implies a weak or no linear correlation between the variables.

We can ignore diagonal line from top left to bottom right because these self-relation mapping that’s why these shown value “1”. Some relations are extremely dependent such as Sum with Mean (both showed 1) and Skewness with Kurtosis (0.94). Similarly, Range and Stardard Deviation are also looks tightly coupled. All other relationships are below zero and marked with gray cells. Another interesting relationship between Sum and Mean with Range and Standard what indeicates a moderated relationship with value between 0.45 to 0.42.

The overall heatmap shows are trustable relationships between all the features, although some of the relationships are very close but these are fewer and may not be cause problem overall system. If many of the relationships are close to eachother then these might be damaging the performance, because, Naïve Bayes classifier assumes that different features are ideally independent and each feature plays its part in the classification procedure.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

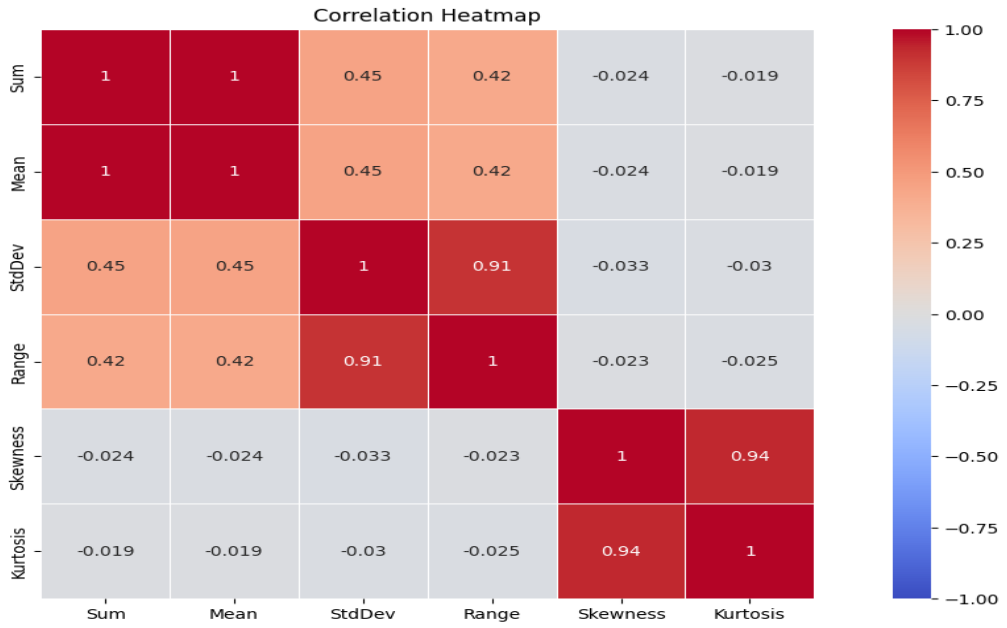

Figure 6: Correlation Heat Map

In previous section we determine the properties of features and find their orientations. Now we are able to perform our fundamental tasks of classification. To perform classfication procedure on dataset, we setup in two stages. On first stage, we implement the Naïve Bayes classifier on data. To make the classfication process more comprehensive, we reconcile the designated classes. Now we add Freeze (very rarely invoked), Cold (rarely invoked), Warm (frequently invoked) and Hot (very frequently invoked). After evaluation of first stage, we implement with Kernal Density Estimation to improve the pefromance metrics.

For comprehensive classification process we use multiple ratios of train and test sets. We start from 80 and 20 ratio (80% testing set and 20% training set) increase the training set with respect to 5% on each iteration. In the following section we will discuss the result of classification on different standard metrics.

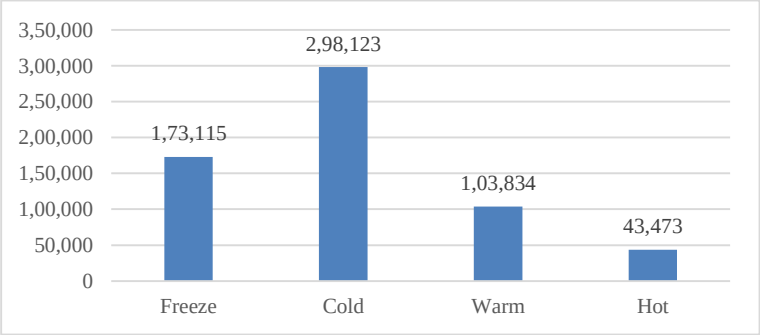

Figure 7: Class Distribution of Instances

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

As we examine the distribution of lables, we found that Freeze and Cold lables are much higher than Warm and Hot. It shows that classes are imbalanced and this may produce unevent results with respect to confusion matrics and precision. When we generate a confusion matrix for premitive algorithm, we found following results.

If we look at the figure 7 in the context of figure 8 then we can observe that lower frequent classes are less correctly labeled while higher frequent classes are more correctly labeled. For instance, Hot class only have 43,473 number of instances in dataset and classifier labeled the 23,853 instances correctly in this class which is about 55% accuracy. On the other hand, Cold class which contains 298,123 instances overall, marked with 209,174 instances correctly which is approaching to 70%. All these statistics displays that our basic algorithm facing challenges with correctly labeling the classes. Similarly, the overall accuracy of Warm and Freeze labels also remained lower with 50% and 63% respectively. Consider that we presenting the above statistics on the execution of 50% which is a moderated segregation. All other distribution of testing and training sets also endorsed the observation.

	Hot	Warm	Cold	Freez
Hot	23,853	39,118	10,772	5,296
Warm	12,587	51,390	21,916	14,739
Cold	4,021	11,052	209,174	44,718
Freez	3,012	22,74	56,761	108,362

Figure 8: Confusion Matrix

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

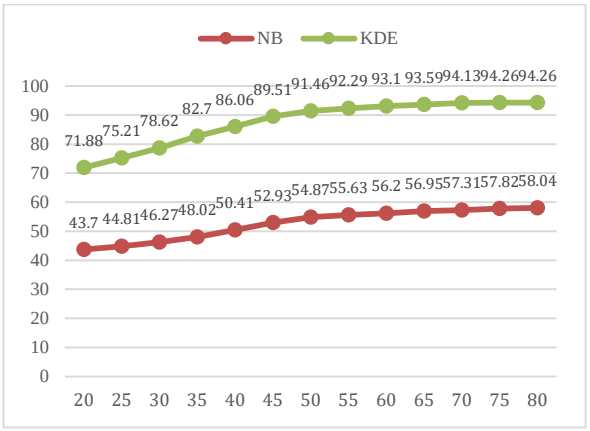

Figure 9: Accuracy of Hot label

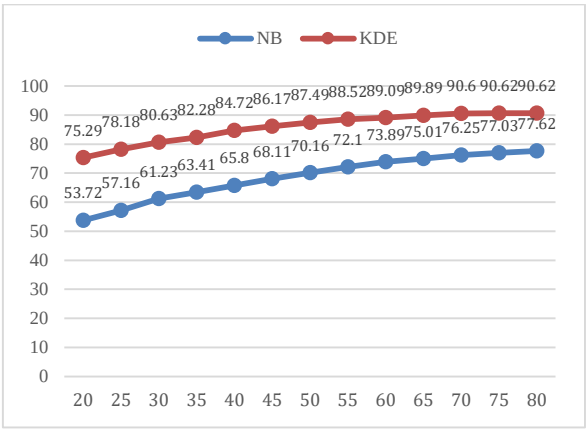

Figure 10: Performance of Accuracy of Warm Class

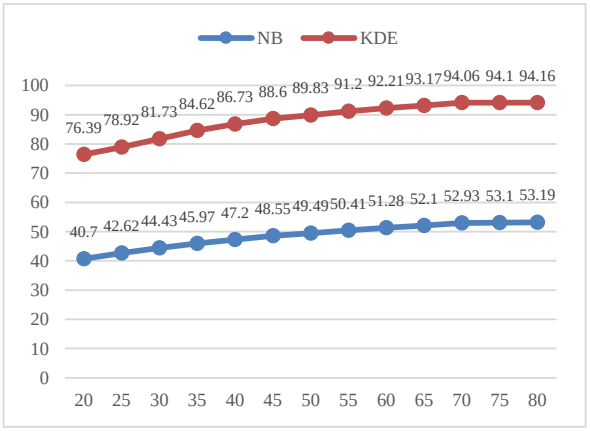

Figure 11: Performance Comparison on NB and KDE on Cold Class

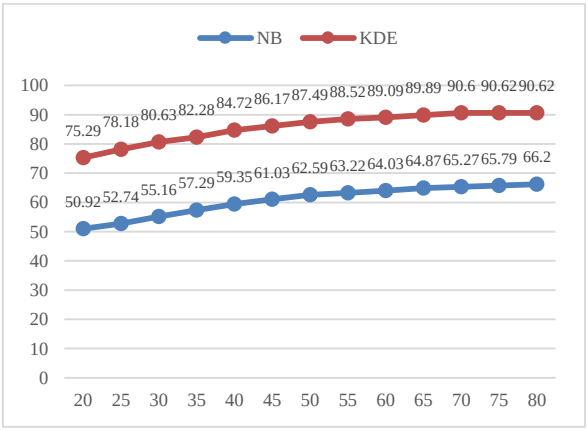

Figure 12: Comparison of NB and KDE in Freeze Class

Performance Comparison of Naive Bayes and Naive Bayes with KDE

The above discussion and visuals reveal that the size of dataset is a vital point too. As the number of instances increases, the accuracy also increases but there is diminishing point as well. In this research, it looks that 50% is mark where performance remain steady and not much higher increases in accuracy. We also validated the accuracy in both NB and KDE remains constant after passing pointed mark. Concluding remarks will also be discussed in next section.

5. CONCLUSION

Both of the measurements, accuracy and confusion matrix, were fundamental requirements to produce a solid result. We already knew that dataset will show an imbalanced approach because literature already identify that there not a great number of

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

instances which should be required hot restart. The phenonmenon make the prediction model more difficult because it may produce results that may look more accurate but inside they are not able to identify the hot instances. The foundation algorithm also shows inaccurate results that develop a motivation to implement KDE. The impact of KDE also influenced the percision of results and we me look that class wise results also enhanced after implementation of supportive algorithm.

Accuracy is the fundamental metrics of any classification mode. Accuracy depends on multiple aspects of dataset and the most important is distribution of trianing and testing dataset size. To ascertain the actual impact on accuracy we applied multiple sizes of both sets. We start from 20-80 ratio where 20% was the training set size and 80% was testign set size and gradually increased the training and reduced testing set size and move the distribution to 80-20 ratio.

We can clearly observe that accuracy increases as size of training set size increases but the it remains constant after a particular point and afterward that point not great improvement may observed. In our case, this particular point was 50%. Another vital observation that Naïve Bayes algorithm showed low performance. We deployed Kernal Density Estimation to increase the performance and the results were significantly upgraded but the accuracy pattern remains same. The actual difference between both results were the magnitude of accuracy. When Naïve Bayes algorithm start execution, its accuracy about 43% which is really low and not up to the mark while with the support of KDE, it shows 70% with first run where training set was the minimal. The accuracy approached to 94%.

6. FUTURE WORK AND CONCLUSION

We have addressed a contemporary issue and tried our best to develop a comprehensive solution for that but there are some certain limitations which we observed during experimental process. We did put these issues in to the weakness of our work but these limitations are also a motivation to next phase knowledge production and research. We have mainly identified two limitations including robustness of dataset and evolatuation of production systems.

We utilized the dataset published by Windows Azure Function. Microsoft a big player in Cloud arena but there are also other player too like Google, Ali Baba, Amazone. We did not examine our results on their datasets. It is also important to study and evaluate datasets published by other Cloud venders and make the results more ethentic. The structure of applied dataset is in a specific format and it may quite different from other published dataset. It looks another worthwhile research work to obtained other datasets, study and then apply some machine learning technique that enhance the productivity of Cloud systems.

We have performed a detail series of experiment to measure the productivity of Cold restart issue but the main purpose of the proposed solution to increase the throuput of Cloud infrastructure and the athenticity of the claim may examine on some production

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

systems. Unfortunatly, the production system are very complex, expensive and large and not easy to maintian them for a research purpose. This is another improtant motivation too to implement the evaluate the proposed solution some real time environment such as Windows Azure Function based some Cloud service.

References

1) E. Jonas *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," 2019.

2) I. Baldini *et al.*, "Serverless computing: Current trends and open problems," pp. 1-20, 2017.

3) H. B. Hassan, S. A. Barakat, and Q. I. J. J. o. C. C. Sarhan, "Survey on serverless computing," vol. 10, no. 1, pp. 1-29, 2021.

4) Y. Li, Y. Lin, Y. Wang, K. Ye, and C. J. I. T. o. S. C. Xu, "Serverless computing: state-of-the-art, challenges and opportunities," vol. 16, no. 2, pp. 1522-1539, 2022.

5) M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 1063-1075.

6) www.microservices.io. *What are microservices?* Available: https://microservices.io

7) T. Yu *et al.*, "Characterizing serverless platforms with serverlessbench," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 30-44.

8) Z. Al-Ali *et al.*, "Making serverless computing more serverless," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 456-459: IEEE.

9) L. Helali and M. N. J. C. S. R. Omri, "A survey of data center consolidation in cloud computing systems," vol. 39, p. 100366, 2021.

10) V. Ivanov and K. Smolander, "Implementation of a DevOps pipeline for serverless applications," in *Product-Focused Software Process Improvement: 19th International Conference, PROFES 2018, Wolfsburg, Germany, November 28–30, 2018, Proceedings 19*, 2018, pp. 48-64: Springer.

11) K. Kritikos and P. Skrzypek, "A review of serverless frameworks," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 161-168: IEEE.

12) A. Jindal and M. Gerndt, "From devops to noops: Is it worth it?," in *Cloud Computing and Services Science: 10th International Conference, CLOSER 2020, Prague, Czech Republic, May 7–9, 2020, Revised Selected Papers 10*, 2021, pp. 178-202: Springer.

13) D. Sokolowski, P. Weisenburger, and G. Salvaneschi, "Automating serverless deployments for DevOps organizations," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 57-69.

14) G. Adzic and R. Chatley, "Serverless computing: economic and architectural impact," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 884-889.

15) V. Yussupov, U. Breitenbücher, F. Leymann, and C. Müller, "Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 273-283.

16) J. Wen, Z. Chen, X. Jin, X. J. A. T. o. S. E. Liu, and Methodology, "Rise of the planet of serverless computing: A systematic review," 2023.

17) H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 442-450: IEEE.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

18) N. J. C. o. t. A. Savage, "Going serverless," vol. 61, no. 2, pp. 15-16, 2018.

19) Microsoft. *Azure Functions triggers and bindings concepts*. Available: https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=csharp

20) H. Shafiei, A. Khonsari, and P. J. A. C. S. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," vol. 54, no. 11s, pp. 1-32, 2022.

21) E. Oakes *et al.*, "{SOCK}: Rapid task provisioning with {Serverless-Optimized} containers," in *2018 USENIX annual technical conference (USENIX ATC 18)*, 2018, pp. 57-70.

22) M. Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 205-218.

23) H. Ebrahimpour, M. Ashtiani, F. Bakhshi, and G. J. T. J. o. S. Bakhtiariazad, "A heuristic-based package-aware function scheduling approach for creating a trade-off between cold start time and cost in FaaS computing environments," pp. 1-49, 2023.

24) J. Cadden, T. Unger, Y. Awad, H. Dong, O. Krieger, and J. Appavoo, "SEUSS: skip redundant paths to make serverless fast," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1-15.

25) F. Romero *et al.*, "Faa$T: A Transparent Auto-Scaling Cache for Serverless Applications," presented at the Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 2021. Available: https://doi.org/10.1145/3472883.3486974

26) D. Mvondo *et al.*, "OFC: an opportunistic caching system for FaaS platforms," presented at the Proceedings of the Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, 2021. Available: https://doi.org/10.1145/3447786.3456239

27) S. Narayana and I. A. Kash, "Keep-Alive Caching for the Hawkes process," presented at the Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence, Proceedings of Machine Learning Research, 2023. Available: https://proceedings.mlr.press/v216/narayana23a.html

28) A. Fuerst and P. Sharma, "FaasCache: keeping serverless computing alive with greedy-dual caching," presented at the Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual, USA, 2021. Available: https://doi.org/10.1145/3445814.3446757

29) C. Chen, L. Nagel, L. Cui, and F. P. Tso, "S-Cache: Function Caching for Serverless Edge Computing," presented at the Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking, Rome, Italy, 2023. Available: https://doi.org/10.1145/3578354.3592865

30) B. Li, Z. Li, J. Luo, Y. Tan, and P. J. E. Lu, "μFuncCache: A User-Side Lightweight Cache System for Public FaaS Platforms," vol. 12, no. 12, p. 2649, 2023.

31) P. Silva, D. Fireman, and T. E. Pereira, "Prebaking Functions to Warm the Serverless Cold Start," presented at the Proceedings of the 21st International Middleware Conference, Delft, Netherlands, 2020. Available: https://doi.org/10.1145/3423211.3425682

32) P.-M. Lin and A. J. a. p. a. Glikson, "Mitigating cold starts in serverless platforms: A pool-based approach," 2019.

33) B. Sethi, S. K. Addya, and S. K. Ghosh, "LCS : Alleviating Total Cold Start Latency in Serverless Applications with LRU Warm Container Approach," presented at the Proceedings of the 24th International Conference on Distributed Computing and Networking, Kharagpur, India, 2023. Available: https://doi.org/10.1145/3571306.3571404

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10153003

34) X. Liu *et al.*, "FaaSLight: General Application-level Cold-start Latency Optimization for Function-as-a-Service in Serverless Computing," vol. 32, no. 5 %J ACM Trans. Softw. Eng. Methodol., p. Article 119, 2023.

35) J. Shen, T. Yang, Y. Su, Y. Zhou, and M. R. Lyu, "Defuse: A dependency-guided function scheduler to mitigate cold starts on faas platforms," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 194-204: IEEE.

36) W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: A dynamic and efficient serverless and FaaS framework for private cloud," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 1416-1421: IEEE.

37) S. Agarwal, M. A. Rodriguez, and R. Buyya, "A reinforcement learning approach to reduce serverless function cold start frequency," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 797-803: IEEE.

38) D. Bermbach, A.-S. Karakaya, and S. Buchholz, "Using Application Knowledge to Reduce Cold Starts in FaaS Services," *Sac '20,* pp. 134–143, 2020.

39) P. Vahidinia, B. Farahani, and F. S. J. I. I. o. T. J. Aliee, "Mitigating cold start problem in serverless computing: a reinforcement learning approach," vol. 10, no. 5, pp. 3917-3927, 2022.

40) R. Moreno-Vozmediano, E. Huedo, R. S. Montero, and I. M. Llorente, "Latency and resource consumption analysis for serverless edge analytics," *Journal of Cloud Computing,* vol. 12, no. 1, p. 108, 2023/07/19 2023.

41) M. Steinbach, A. Jindal, M. Chadha, M. Gerndt, and S. J. I. A. Benedict, "Tppfaas: Modeling serverless functions invocations via temporal point processes," vol. 10, pp. 9059-9084, 2022.

42) A. P. Jegannathan, R. Saha, and S. K. Addya, "A Time Series Forecasting Approach to Minimize Cold Start Time in Cloud-Serverless Platform," in *2022 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2022, pp. 325-330: IEEE.

43) K. Solaiman and M. A. Adnan, "WLEC: A not so cold architecture to mitigate cold start problem in serverless computing," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*, 2020, pp. 144-153: IEEE.

44) Z. Xu, H. Zhang, X. Geng, Q. Wu, and H. Ma, "Adaptive function launching acceleration in serverless computing platforms," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 2019, pp. 9-16: IEEE.

45) S. Wu *et al.*, "Container lifecycle-aware scheduling for serverless computing," vol. 52, no. 2, pp. 337-352, 2022.

46) N. Daw, U. Bellur, and P. Kulkarni, "Xanadu: Mitigating cascading cold starts in serverless function chain deployments," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 356-370.

47) K. Suo, J. Son, D. Cheng, W. Chen, and S. Baidya, "Tackling cold start of serverless applications by efficient and adaptive container runtime reusing," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 433-443: IEEE.