Science & Technology Development Journal – Engineering and Technology 2024, ():1-8

Open Access Full Text Article

Bài nghiên cứu

MSDF: Memory Statistics Data Format used in system monitoring

La Quoc Nhut Huan[1,2,3,*], Nguyen Manh Thin[1,3], Nguyen Le Duy Lai[1,3], Nguyen Quang Hung[1,3], Thoai Nam[1,2,3]

ABSTRACT

High performance computing (HPC) system or computing system is very different from ordinary service system. In general, service system only run some specific applications, e.g. web server or mail server to serve as many requests from users as possible while in computing system, users have the permission to run their own applications and isolated with each other. Monitoring technique is the key to ensure system efficiency and users satisfaction, and by combining monitoring together with data analysis, system administrators can solve several operating problems specific to computing system such as resource allocation, application scheduling, abnormal detection, etc. Different from service system while administrators usually prefer system overall information rather than information of each individual user applications in computing system. Since computing system usually contains many applications executed simultaneously, monitoring computing system with traditional approaches would potentially consume a huge amount of storage space and would cost more charge fee if system is deployed in cloud environment.

This article focuses on analyzing monitored memory usage data retrieved from computing program in order to benefit its next resource allocation. Different from traditional approaches with batch processing technique in which collected data is all stored in database before analyzing, we utilized online analysis approaches in which every new coming data is captured, processed, cached in order to transform into useful information, and only allow necessary data be stored in database. We propose Memory Statistics Data Format (MSDF), an on-the-fly processing technique used in monitoring memory usage of computing application for saving storage space while still preserve enough information to solve resource allocation problem. MSDF can help to save more than 95% of storage space while allocation efficiency is always guaranteed depend on the εparameter and MSDF can be extended to solve other operating problem or adapted to montior and analyze other remaining application metrics.

Key words: System Monitoring, Memory Monitoring, Streaming Processing, Online Analysis, Memory Allocation

[1]*High Performance Computing Laboratory, Faculty of Computer Science and Engineering (HPC Lab), Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam*

[2]*Advanced Institute of Interdisciplinary Science and Technology (iST), Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam*

[3]*Vietnam National University Ho Chi Minh City (VNU-HCM), Linh Trung Ward, Thu Duc City, Ho Chi Minh City, Vietnam.*

Correspondence

La Quoc Nhut Huan, High Performance Computing Laboratory, Faculty of Computer Science and Engineering (HPC Lab), Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam

Advanced Institute of Interdisciplinary Science and Technology (iST), Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam

Vietnam National University Ho Chi Minh City (VNU-HCM), Linh Trung Ward, Thu Duc City, Ho Chi Minh City, Vietnam.

Email: huan@hcmut.edu.vn.

INTRODUCTION

High Performing computing (HPC) system or computing system or computing system in general is critical for scientific research. One of its prominent examples of application is the artificial intelligence training for Smart Village project. Computing system contains a large number of nodes with special network topology and technologies to boost the parallel computing ability as much as possible. Different from ordinary service system such as web or mail server, users in computing system have the permission to access and execute their own programs, i.e. application. system resources are shared between multiple users and is allocated based on user's requirements and allocation policies defined by system administrators. As a consequence, efficiently managing and operating computing system with multiple users and a vast number of applications running simultaneously would cost admins much more effort.

Monitoring computing infrastructure helps admins continuously follow system operation, profile abnormal behavior, etc. and facilitate admins to update their management and operating plan in the future. "*We are drowning in data but starving for information*"[1], system monitoring often lacks of analysis ability to transform raw data into useful information and knowledge, which leads to the situation where every piece of data collected from any metrics considered potential for later analysis has to be permanently stored. As a result, since monitored metrics are collected at application level, applying traditional monitoring in computing system would potentially consume a huge number of storage space and cost more charge fee if system is deployed in cloud environment. The gap between data saved in storage and actual data in use can be filled with online-analysis which stores only necessary information retrieved from processing raw monitoring data.

Cite this article : Huan L Q N, Thin N M, Lai N L D, Hung N Q, Nam T. **MSDF: Memory Statistics Data Format used in system monitoring**. *Sci. Tech. Dev. J. – Engineering and Technology* 2024; ():1-8.

History
- Received: 25-9-2023
- Accepted: 29-3-2024
- Published Online:

DOI :

Memory usage which directly reflects system health, is always listed as one of the most critical metrics. Information get from analyzing the consumption of application memory benefits from solving several problems including resource allocation, application scheduling, abnormal detection, etc. In this article, we focus on monitoring memory data which is retrieved from our monitoring framework, and analyzing these data to update memory allocation policy for the next execution. Different from normal approaches in which data is collected and immediately passed to storage, we utilized online-analysis in which every new coming data is being processed, cached, and later converted into useful information before being stored in database. We named this method *Memory Statistics Data Format (*MSDF), for later reference convenience. MSDF highlights the ability to potentially save a massive amount of storage size when apply in monitoring and analyzing application memory while still preserves enough information for allocation problem.

Monitoring framework and MSDF is currently being developed at high performing computing center (HPCC) from HCMUT-VNU for our specific SuperNode-XP system. The following briefly summarizes the contribution of this paper:

- Propose Kafka based monitoring framework which can collect application metrics and perform online-analysis.
- Propose dynamic allocation in terms of memory and extract its efficiency boundary.
- Propose online analysis in memory monitoring to save storage space relating to memory allocation for applications in computing system.

The remain of this paper is structured as follow. Section 2 the related work that we have surveyed. Section 3 introduces the architecture overview of our monitoring framework and shortly describes its components. Section 4 demonstrates how we apply MSDF in memory monitoring and allocation and Section 5 will discuss the evaluation results. Section 6 further discusses about analyzing memory data and MSDF. Finally Section 7 summarizes all of our work and outlines the future works.

RELATED WORK

There are immense of open source and commercial monitoring tools for computing system with different feature, different architecture. For wide range of used, *Zabbix* provides a stable solution for monitor host level metrics and hardware through Simple Network Management Protocol (SNMP); *Prometheus* is a new generation tool with flexible user defined metrics through custom exporter and a powerful build-in query language PromQL; *Nagios* highlights the continuous real time network monitor with sensitive failure detection and many other similar softwares can be listed as *Datadog, Icinga, SolarWinds, etc.* In general, these tools are all lack of streaming processing ability when directly forward collected metrics into long term storage.

In recent years, *Apache Kafka* always stays as one of the best distributed data streaming platforms. Kafka provides both message queue and pub sub server at the same time with consistency through synchronize and fault tolerant through controller replicated mechanism and can be scaled horizontally by simply adding more instances to expand the traffic bandwidth. Besides, Kafka come along with a vast amount of processing framework including it own Application Programming Interface (API) in JAVA, PYTHON, SCALA, etc. which has the lowest overhead and mostly used in small scale problem; *Apache Spark* [2], a micro-batch processing framework with build in Kafka compatible Structured Streaming library, mostly used in large scale big data problem; *Apache Flink* [3], a real streaming processing framework with build in Kafka library also used for large scale problem.

Analyzing data from system monitoring is not a new task, there have already been published several researches about this field in recent year. For instance, analyzing power usage of Central Processing Unit (CPU) by streaming linear regression using big data processing framework [4]; reconstructing application heap from monitoring tool trace file to detect memory leak by offline-analysis [5,6]; detecting software aging based on memory leak investigation at software runtime [7]. But to the best of our knowledge, this is the first work which is applied streaming analysis in monitoring of application memory to save storage space with memory allocation problem case study.

MONITORING FRAMEWORK

As depicted in Figure 1, framework contains four different layers. Communication Layer positions in computing nodes and hardware devices, where metrics is directly collected. Communication and Analyzing Layer position in head nodes, i.e. management nodes, where contain central processing logic of monitoring framework in order to minimize computing node overhead. Finally Storage Layer positions in storage nodes where long term data is accumulated for later purposes such as visualization or offline-analysis.

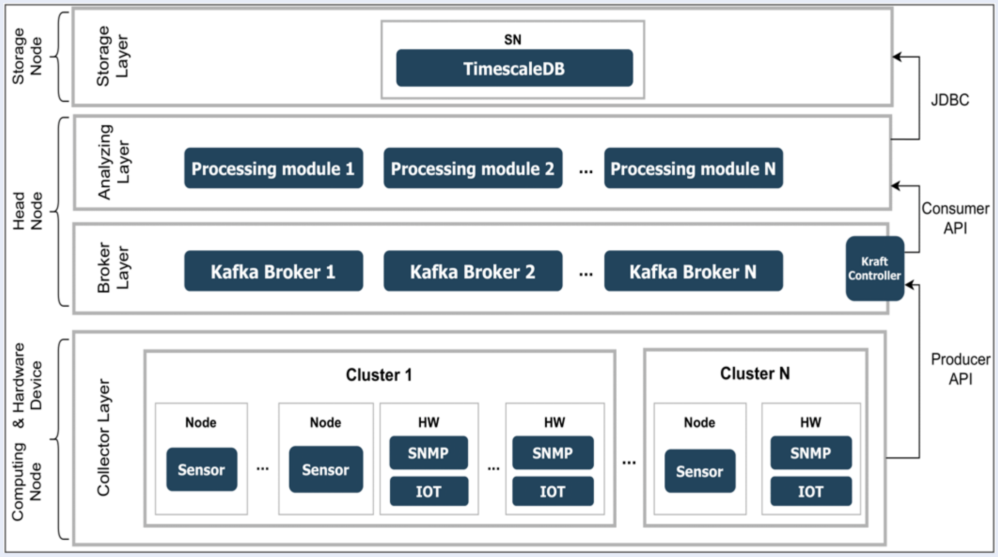

Figure 1: Monitoring Framework Architecture.

Collector Layer. The lowest layer responsible for collecting monitoring metrics. Since load from computing system mostly from scientific programs executed by system users, monitoring framework must be able to collect metrics from each user application. Sensor from monitoring framework, which is a daemon program installed at every computing node, is responsible for collecting resource usage corresponding to each user and application such as CPU utilization, memory usage, network traffic, etc. Beside, Sensor also considers to collect other hardwares metrics such as network devices which exports through SNMP protocol; Uninterruptible Power Supply (UPS) devices which exports through SNMP; temperature, humidity sensor through Message Queuing Telemetry Transport (MQTT) protocol.

Theoretically, each cluster in computing system has different role, different behavior hence Sensor must be able to configure to collect only applicable metrics with appropriate interval. Additionally, sensors also can perform simple preprocessing step without costing too much computing resources if necessary. These data will eventually be pushed up to Communication Layer through Kafka Producer API.

Communication Layer. Because of a large number of applications executed at the same time, the communication between Analyzing Layer and data source become extremely complicated. Apache Kafka takes role as an intermediate data broker to provide a reliable transmission channel. Kafka implements the message queue and pub sub server where Collector Layer take role as publishers push data to Kafka under specific topics and Analyzing Layer takes role as consumers fetch data from subscribed topics. Further more, Kafka provides a powerful streaming processing API for Analyzing Layer to conveniently perform online processing and analysis. Kafka is coordinated by Kraft Controller which acts as a gateway to receive all request from both producers and consumers and dispatch them to appropriate broker server.

Different type of metrics can be organized under different topic follow Linux hierarchical file system structure. For example *<organization name>/<cluster name>/<node id>* contains metrics of a specific computing node at host level such as CPU load, free disk, etc.; *<organization name>/<cluster name>/<node id>/<user id>* contains metrics of any applications executed under that user id in a specific computing node. Moreover, independent type of metrics can be put under different topic partitions to leverage parallel processing.

Analyzing Layer. This layer is responsible for handling streaming data from Communication Layer, each Processing Module will have different operations based on topic-name and partition-id. In relatively small system, Processing Module uses simple API which is JAVA Kafka Consumer to process streaming data and Java Database Connectivity (JDBC) driver to ingress into storage. But in large scale computing system with a vast number of clusters, users and applications where monitoring metrics from Collector Layer are considered big data, streaming processing with Apache Spark is seem to be more suitable.

Streaming processing ability can be utilized to create new metrics in more generalized level, for example aggregating each individual node health status to get the overall cluster health; or to immediately response when errors occur, for example auto shut down all infrastructure when detected power cut. Moreover, performing online-analysis in system monitoring can significantly reduce the total amount of data store and computing effort. We will further discuss how to apply online-analysis in case of memory monitoring in Section 4.

Storage Layer. This layer contains database for long term storing where data can later be used in offline-analysis phrase, e.g applying machine learning algorithm to predict future trend to generate the next upgrade plan. Naturally, monitoring data is time series but because of Extract-Load-Transform (ETL) process in Analyzing Layer, storage should also support data warehouse along with time series database. We use TimescaleDB an extension of PostgreSQL utilizes for time series data but still maintaining relational data model.

MSDF METHOD

Memory Allocation

Computing resources is shared between multiple programs executed by different users. An effective allocation strategy can result in more applications execute at the same time and hence boosting system efficiency. Allocation mechanism is usually classified as static and dynamic allocating. With naive static allocation, memory is given for application based on the maximum usage which is collected from execution in the past. This strategy advantages can be listed as simple implementation, can prevent application from crashing and memory overflow, but is ineffective because of memory wasting. As in Figure 2, memory usage usually will fluctuate within a certain range of value for a specific time before significant change happen; and in most of the time, memory usage is far from its maximum value. Other applications may have more stable memory lines however allocating with static policy is not effective with this type of application and computing system in general. Thus, dynamic allocation should be utilized to only give application the most appropriate memory at specific period of time in execution to increase allocation efficiency.

Allocation from serverless computing [8] is rated as one the best among dynamic allocation mechanism, in which application continuously requests the amount of memory needed and allocator will then give exactly that amount of memory. Nevertheless, applications in computing system are mostly located inside container environment, although we can dynamically adjust the environment resources, we can not continuously update its configuration due to technology limited. Inspired from serverless computing, suppose the next execution is nearly similar to previous one, application runtime can be split into multiple continuous segments and each with different allocation of memory. Memory given in each segment should be around the maximum value recorded in history corresponding to that segment. Recall from Figure 2, theoretically data points in each segment should be stable and as close to the maximum value of that segment as possible to maximize allocation efficiency. Thus, any free data points, i.e have not yet belonged to any segments before significant change happen, should be grouped into the same segment.

MSDF Approach

Allocation strategy from Section 4.1 only requires the maximum memory usage. With normal approach, monitoring framework permanently stored every memory data of application collecting at different timestamp, allocator queries and traverses through all of that data to compute the maximum value at each different segment. Based on sensor interval and application run time, each execution could end up thousand to million of records in database which will largely cost storage size and computing effort. Instead with applying online processing in monitoring, when new data is coming, MSDF can calculate segment statistic value immediately and store only necessary data.

In particular, MSDF calculates the max, min, mean, and amount of data points of each segment. Suppose the new coming data at n^thoffset is M_n, and current max, min are MAX and MIN, new values can be easily calculated with below formula:

$$MAX = Greater(M_n, MAX).$$
$$Min = Lesser(M_n, MIN).$$

Calculating mean ($\bar{M}_n$) from previous mean n-1 value is a bit more complicated by following the formula below:

$$\bar{M}_n = \frac{\bar{M}_{n-1} * (n-1) + M_n}{n}$$

We defined $\varepsilon \in (0,1]$ is the Threshold parameter and can be configured, a significant change is considered to happen when new coming data exceeds the given εthreshold:

$$\frac{|\bar{M}_{n-1} - M_n|}{\bar{M}_{n-1}} > \varepsilon. \quad (1)$$

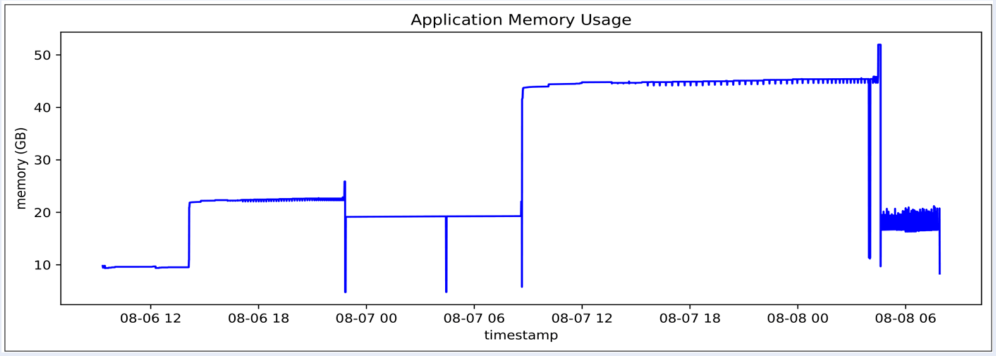

Figure 2: Efficiency and records stored in Storage Layer associate with different thresholds.

Then MSDF store the information related to that segment into database, reset n=0, move to the next segment and calculate new statistics value. By leveraging streaming processing technique, the final memory information of application execution saved in database is only statistics values including mean, max, min and number of data points corresponding to each segment.

Allocation Efficiency

We define a metric to estimate the efficiency of memory allocation corresponding to each threshold ε. Suppose f(t) is the memory usage function of application at time t, segment begins at t_a and ends at t_c, and the total memory used by application in this segment is defined as:

$$MemoryUsage = \int_{t_a}^{t_c} \square f(t)\, dt. \quad (2)$$

In fact, since the *f(t)* function is unknown, we only have the set of discrete data points, where M_iindicated memory used at a specific time twhich is collected from monitoring application. Suppose segment contains kdata points with Iindicated the sample interval of monitoring sensor. Integral (2) is now being calculated by discrete rectangle method:

$$\begin{aligned} &MemoryUsage = \textstyle\sum_{i=1}^{k} \square M_i * \triangle t_i \\ &\Leftrightarrow MemoryUsage = I * \textstyle\sum_{i=1}^{k} \square M_i. \end{aligned} \quad (3)$$

As mentioned before, the memory allocated to each segment is approximate to *MAX* value of that segment. Applying (3), the expected memory consumption is:

$$\begin{aligned} &SegmentEfficiency = \frac{I * \sum_{i=1}^{k} \square M_i}{k * I * MAX} \\ &\Leftrightarrow SegmentEfficiency = \frac{\frac{\sum_{i=1}^{k} \square M_i}{k} * 1}{MAX} \\ &\Leftrightarrow SegmentEfficiency = \frac{\bar{M}}{MAX}. \end{aligned} \quad (5)$$

With (5) is the allocation efficiency of each segment, suppose we have nsegments, T_i is the i^thsegment length and K_i is the number of data points in the i^{th} segment, the total efficiency corresponding to the defined threshold is:

$$\begin{aligned} &TotalEfficiency = \frac{\sum_{i=1}^{n} \frac{\bar{M}_l}{MAX}.T_i}{\sum_{i=1}^{n} \square T_i} \\ &\Leftrightarrow TotalEfficiency = \frac{\sum_{i=1}^{n} \frac{\bar{M}_l}{MAX}.K_i}{\sum_{i=1}^{n} \square K_i}. \end{aligned} \quad (6)$$

Efficiency Boundary

Efficiency boundary of each segment can guarantee the quality of whole allocation solution. Naturally, the upper bound reaches 1 and represented the most ideal scenario when application used exactly the same amount of given memory. And lower bound represents the worst situation that allocation strategy can be encountered. Thus this subsection will mainly focus on finding the lower bound of the solution from Section 4.1.

In each segment, suppose M_1 is the first data point of segment and $(\bar{M}_1) = M_1$. Maximum value of segment reaches its highest threshold when data points in segment progressively increase by a largest allowable value between any data points. From (1), we have:

$$\begin{aligned} &\overline{M_{n-1}} - M_n \geq -\varepsilon \overline{M_{n-1}} \\ &\Leftrightarrow M_n \leq \overline{M_{n-1}} * (1+\varepsilon). \end{aligned} \quad (7)$$

Equal sign from (7) occurs in any data points, for all k > 1, mean value of the first kdata points in segment can be calculated as below:

$$\begin{aligned} &\bar{M}_k = \frac{\overline{M_{k-1}} * (k-1) + \overline{M_{k-1}} * (1+\varepsilon)}{k} \\ &\Leftrightarrow \bar{M}_k = \overline{M_{k-1}} * \left(1 + \tfrac{\varepsilon}{k}\right). \quad (9) \\ &\square \\ &(9) \Rightarrow \bar{M}_2 = M_1 * \left(1 + \tfrac{\varepsilon}{2}\right) \quad (10) \\ &(9),(10) \Rightarrow \bar{M}_3 < M_1 * \left(1 + \tfrac{\varepsilon}{2}\right)^{k-1}. \quad (12) \end{aligned}$$

In contrast, Minimum value of segment reaches it lowest threshold when data points in segment progressively decrease by a largest allowable value between any data points. From (1), we have:

$$\begin{aligned} &\overline{M_{n-1}} - M_n \leq \varepsilon \overline{M_{n-1}} \\ &\Leftrightarrow M_n \geq \overline{M_{n-1}} * (1-\varepsilon). \end{aligned} \tag{9}$$

And similarly, equal sign from (9) occurs in any data points. The mean value of first k data points in segment is:

$$\begin{aligned} &\bar{M_k} = \overline{M_{k-1}} * \left(1 - \frac{\varepsilon}{k}\right) \\ &\Rightarrow M_k > M_1 * \left(1 - \frac{\varepsilon}{k}\right)^{k-1}. \end{aligned} \tag{10}$$

In order to figure out the allocation lower bound, the situation when efficiency become worst must be first determined. There are three different cases of memory consumption:

A: Memory is progressively increases within thresholds.

B: Memory is progressively decreases within thresholds.

C: Memory is randomly changes within thresholds.

From (7) and (8), suppose segment has kdata points, the efficiency of case A is:

$$\begin{aligned} &\frac{\bar{M}}{MAX} = \frac{M_1 * \Pi_{n=2}^{k}\left(1+\frac{\varepsilon}{n}\right)}{M_1 * \Pi_{n=2}^{k-1}\left(1+\frac{\varepsilon}{n}\right) * (1+\varepsilon)} \\ &\Leftrightarrow \frac{\bar{M}}{MAX} = \frac{1+\frac{\varepsilon}{k}}{1+\varepsilon}. \end{aligned} \tag{11}$$

From (9) and (10, suppose segment has k data points, the efficiency of case B is:

$$\begin{aligned} &\frac{\bar{M}}{MAX} = \frac{M_1 * \Pi_{n=2}^{k}\left(1-\frac{\varepsilon}{n}\right)}{M_1} \\ &\frac{\bar{M}}{MAX} = \Pi_{n=2}^{k}\left(1-\frac{\varepsilon}{n}\right). \end{aligned} \tag{12}$$

For all k > 2 and $\varepsilon \in (0,1]$, (11) > (12) by using investigating function approach. Therefore, the efficiency from case B is worse than case A.

$k = 2 \Rightarrow 1 + \frac{\varepsilon}{2} > \left(1 - \frac{\varepsilon}{2}\right)(1+\varepsilon)$

$k = 3 \Rightarrow 1 + \frac{\varepsilon}{3} > \left(1 - \frac{\varepsilon}{2}\right)\left(1 - \frac{\varepsilon}{3}\right)(1+\varepsilon)$

$k > 3 \Rightarrow \frac{1+\frac{\varepsilon}{k}}{1+\varepsilon} > \left(1 - \frac{\varepsilon}{2}\right)\left(1 - \frac{\varepsilon}{3}\right)\left(1 - \frac{\varepsilon}{4}\right)(1-\varepsilon) \Rightarrow$
$\frac{1+\frac{\varepsilon}{k}}{1+\varepsilon} > \Pi_{n=2}^{k}\left(1 - \frac{\varepsilon}{n}\right).$

The efficiency from case C is equal to the case when all data points in case C is sorted in gradually decreasing order. In this case, we can consider each data point is changed α time compared to mean value of previous data points. Additionally, the difference between MAX and MIN in case C is smaller than the difference in case B. Hence, we have:

$$\begin{Bmatrix} M_n = \overline{M_{n-1}} * (1-\alpha_n) \\ \alpha_n < \varepsilon, \forall n \end{Bmatrix} \tag{13}$$

From (13), following similar step from case B, the mean value of first k data points in segment is:

$$\bar{M_k} = M_1 * \Pi_{n=2}^{k}\left(1 - \frac{\alpha_n}{n}\right). \tag{14}$$

From (13) and (14), suppose segment has k data points, the efficiency of case C is:

$$\begin{aligned} &\frac{\bar{M}}{MAX} = \frac{M_1 * \Pi_{n=2}^{k}\left(1-\frac{\varepsilon}{n}\right)}{M_1} \\ &\Leftrightarrow \frac{\bar{M}}{MAX} = \Pi_{n=2}^{k}\left(1-\frac{\varepsilon}{n}\right) \end{aligned} \tag{15}$$

The efficiency in case B is also worse then case C because of (12), (13) and (15):

$$\begin{aligned} &1 - \frac{\alpha_n}{n} > 1 - \frac{\varepsilon}{n}\ \forall n \\ &\Leftrightarrow \Pi_{n=2}^{k}\left(1 - \frac{\alpha_n}{n}\right) > \Pi_{n=2}^{k}\left(1 - \frac{\varepsilon}{n}\right). \end{aligned} \tag{16}$$

Thus it is confident to say the worst efficiency belongs to case C when memory consumption is progressively decreased. The boundary of segment efficiency is:

$$\Pi_{n=2}^{k}\left(1 - \frac{\varepsilon}{n}\right) < Efficiency < 1$$

When k→+∞, lower bound will go toward 0 but in face, kis always a limited number and my depend on program type, program execution time, sensor collecting interval, etc. Table 1 showed the lower bound of different $\varepsilon$with different k. Decreasing ε can potentially lead to decrease of knumber in all segments, the smaller ε value, the more efficiency could be guaranteed.

RESULT AND EVALUATION

MSDF proposes a way to store monitoring memory data and retain only necessary information in order to save storage space. In case of memory allocation problem, ε value indicates the trade off between allocation efficiency and storage saving. We define some metrics to clarify MSDF efficiency and the trade off between these two factors with different εvalue. Efficiency score showed the efficiency of memory allocation could potentially achieve with segment information corresponding to the εvalue. Number of blocks indicates the disk block used by storage to save monitoring data, suppose the field size of all type of monitoring data in database is all equal to exactly on

Table 1: Efficiency lower bound of allocation strategy corresponding to different ε and different k.

ε	1	0.8	0.5	0.35	0.2	0.1
K = 1000	0	0	0.04	0.1	0.27	0.52
K = 10000	0	0	0.01	0.04	0.17	0.41

block. And finally, storage metrics compares the storage space of MSDF approach to normal approaches such as Zabbix or Prometheus.

As mentioned before, to our knowledge, there have not been any works which is similar to ours. MSDF is evaluated on four different computing programs from civil engineering research at our SuperNode-XP system with 10 seconds collector sensor interval. Each program used VASP library and executed in total 196.6 hours on computing node with 48 cores, 96 threads and 256 GB memory configuration. And normal approaches which store all monitoring data is set as the baseline for comparison purpose.

Efficient scores from Table 2 confirmed the validity of Table 1 where efficient scores are all greater than the lower bound in the same ε value. When the threshold value is decreased, the efficiency increased but number of blocks, i.e storage size also increased. Because lower threshold means more strict in allowing value change to be happen, and hence will be split to more segments which cost more storage size. But the fluctuation of data points in each segment will become stable thus increasing the overall efficiency score.

When ε goes toward 0, efficiency goes toward 1 and number of records reaches total raw records in which mean $=$ max $=$ min $=$ value. Based on the efficiency score, storage saving and the statistics value at each segment, ε value can be reconfigured to find the best trade off between efficiency and storage saving. Different type of applications may yield more or less optimistic result, however with VASP programs above, MSDF is able to save 99% storage when allocation effectiveness reach more than 80%.

DISCUSSION

Allocation strategy from Section 4 suggest that at each segment, program should be allocated to the maximum memory usage of that segment. In fact, at each segment, resources must be allocated before, but the maximum value can not be found until reaching the end of that segment. Fortunately, applications in computing system usually belong to parameter-sweep class [9], i.e program executes each time in the same behavior but with different input. Thus applications in computing system can be assumed that memory or resources between its different executions do not vary much, so it is feasible to apply history information including segment and segment maximum to the next execution.

The core idea of MSDF is to group together any continuous and stable data points. MSDF accepts new coming data changing below certain threshold compared to previous data points. In case of applications which memory usage is gradually increased or decreased within the allowable threshold, MSDF eventually will have only one segment with low efficiency allocation. As a consequence, MSDF should not be applied in applications with resource usage gradually changed behavior.

Additionally, since the final data saved in storage of each application executions is only segment information, These value can be directly visualized as shown in Figure 3 without being recomputed. The more closer between line and upper rectangle boundary compared to lower boundary indicated the more efficiency of allocation in the corresponding segment. Moreover, by visualizing different application executions and stacking these graphs together, system-operating questions such as whether these applications are able to executed simultaneously can be easily answered. In general MSDF can be utilized to use in scheduling problem as well.

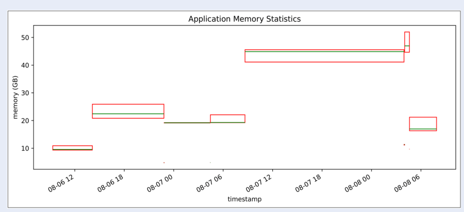

Figure 3: Statistics visualization corresponding to Figure 2. In each segment, the line represented mean value, rectangle represented min and max boundary. Lines without rectangle boundary represents the mean=min=max situation.

CONCLUSION

In this paper, we first introduced our monitoring framework architecture and briefly detailed its components. To sum up, monitoring framework can collect metrics at application level, utilize Apache

Table 2: Efficiency and records stored in storage associate with different thresholds.

Threshold	Efficiency	Number of Blocks	Storage Saving
1	0.04	60	99.91%
0.8	0.28	204	99.71%
0.5	0.67	400	99.43%
0.35	0.79	476	99.32%
0.2	0.85	660	99.06%
0.1	0.91	1136	98.39%
Normal Approaches	~1	70693	100.00%

Kafka as a data broker to organize system hierarchy structure under different topic and also leverage Kafka streaming processing ability to perform online-analysis. As a use case, we demonstrated how to apply online-analysis in monitoring memory for allocation problem through MSDF. MSDF showed the trade off between allocation efficiency and storage saving based on the threshold εvalue. In conclusion, applying MSDF in monitoring and analyzing memory usage of computing application can save a huge storage capacity while still ensure allocation efficiency.

In future, since setting threshold in MSDF can affect both allocation efficiency and storage saving, we aim to fine tuning MSDF ε value in order to get the best trade off between the two factors, and in advanced providing MSDF ability to auto update that threshold value at application runtime. Moreover, also in memory monitoring area, we planned to adapt MSDF to solve application scheduling problem as well.

ACKNOWLEDGMENT

This research was conducted within 58/20-DTDL.CN-DP Smart Village project sponsored by the Ministry of Science and Technology of Vietnam.

We gratefully acknowledge the valuable support and resources provided by HPC Lab at HCMUT, VNU-HCM.

CONFLICT OF INTEREST

The authors confirm that there is not any conflict of interest related to the content reported in this paper.

AUTHORS CONTRIBUTION

La Quoc Nhut Huan: first author, writing & editing, investigation, formal analysis, provide solution.

Nguyen Manh Thin: supervision, validation, function testing, resources providing.

Nguyen Quang Hung: solution advising, reviewing, methodology.

Nguyen Le Duy Lai: solution advising, reviewing, methodology.

Thoai Nam: funding acquisition, supervision, conceptualization, instruction.

REFERENCES

1. Choenni S, Bakker R, Blok HE, de Laat R. Supporting technologies for knowledge management. In: Knowledge Management and Management Learning: Extending the Horizons of Knowledge-based Management. Springer; 2005. p. 89-112;Available from: https://doi.org/10.1007/0-387-25846-9_6.
2. Shoro AG, Soomro TR. Big data analysis: Ap spark perspective. Global Journal of Computer Science and Technology: Csoftware & Data Engineering. 2015;15(1):7-14;.
3. Javed MH, Lu X, Panda DK. Characterization of big data stream processing pipeline: a case study using flink and kafka. In: Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. 2017. p. 1-10;Available from: https://doi.org/10.1145/3148055.3148068.
4. Beneventi F, Bartolini A, Cavazzoni C, Benini L. Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools. In: Design, Automation & Test in EuropeConference & Exhibition (DATE), 2017. IEEE; 2017. p. 1038-1043;Available from: https://doi.org/10.23919/DATE.2017.7927143.
5. Weninger M, Lengauer P, Mössenböck H. User-centered offline analysis of memory monitoring data. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. 2017. p. 357-360;Available from: https://doi.org/10.1145/3030207.3030236.
6. Weninger M, Mössenböck H. User-defined classification and multi-level grouping of objects in memory monitoring. In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering. 2018. p. 115-126;Available from: https://doi.org/10.1145/3184407.3184412.
7. Matias R, Costa BE, Macedo A. Monitoring memory-related software aging: An exploratory study. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops. IEEE; 2012. p. 247-252;Available from: https://doi.org/10.1109/ISSREW.2012.90.
8. Li Z, Guo L, Cheng J, Chen Q, He B, Guo M. The serverless computing survey: A technical primer for design architecture. ACM Computing Surveys (CSUR). 2022;54(10s):1-34;Available from: https://doi.org/10.1145/3508360.
9. Chirigati F, Silva V, Ogasawara E, de Oliveira D, Dias J, Porto F, Valduriez P, Mattoso M. Evaluating parameter sweep workflows in high performance computing. In: Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies. 2012. p. 1-10;Available from: https://doi.org/10.1145/2443416.2443418.

Tạp chí Phát triển Khoa học và Công nghệ – Engineering and Technology 2024, ():1-1

Open Access Full Text Article

Bài nghiên cứu

MSDF: Định dạng thống kê dữ liệu cho bộ nhớ ứng dụng trong giám sát hệ thống

La Quốc Nhựt Huân[1,2,3,*], Nguyễn Mạnh Thìn[1,3], Nguyễn Lê Duy Lai[1,3], Nguyễn Quang Hùng[1,3], Thoại Nam[1,2,3]

Use your smartphone to scan this QR code and download this article

TÓM TẮT

Hệ thống máy tính hiệu năng cao (HPC) hoặc hệ thống tính toán có sự khác biệt nhất định với hệ thống dịch vụ thông thường. Nhìn chung, hệ thống dịch vụ chỉ chạy một số ứng dụng cụ thể, ví dụ như máy chủ web hoặc máy chủ mail và phục vụ cùng lúc nhiều người dùng nhất có thể trong khi với hệ thống tính toán, người dùng trong hệ thống có quyền chạy các ứng dụng của riêng họ và hoàn toàn cô lập với người dùng khác. Kỹ thuật giám sát là chìa khóa để đảm bảo hiệu quả sử dụng hệ thống và sự hài lòng của người dùng, bằng cách kết hợp kỹ thuật giám sát cùng với phân tích dữ liệu, quản trị viên có thể giải quyết một số bài toán vận hành cụ thể như phân bổ tài nguyên, lập lịch ứng dụng, phát hiện bất thường, v.v. Khác với hệ thống dịch vụ trong khi các quản trị viên thường sẽ giám sát những thông tin tổng quát của hệ thống trong khi với hệ thống tính toán sẽ cần giám sát thông tin của từng ứng dụng khởi chạy bởi từng người dùng. Do hệ thống tính toán thường đồng thời thực thi rất nhiều ứng dụng nên việc giám sát bằng các phương pháp truyền thống sẽ tiêu tốn một lượng lớn dung lượng lưu trữ và khiến ta chi trả nhiều phí hơn nếu hệ thống được triển khai trên môi trường điện toán đám mây.

Bài viết này tập trung vào việc phân tích dữ liệu sử dụng bộ nhớ của chương trình tính toán nhằm giải quyết bài toán phân bổ tài nguyên cho lần khởi chạy tiếp theo của ứng dụng đó. Khác với các phương pháp truyền thống trong đó tất cả dữ liệu được giám sát thu thập sẽ được lưu trữ trong cơ sở dữ liệu trước khi phân tích, chúng tôi sử dụng các phương pháp phân tích trực tuyến trong đó mọi dữ liệu mới sẽ được thu thập, xử lý, lưu trữ trong bộ nhớ đệm để chuyển đổi thành thông tin hữu ích và chỉ cho phép dữ liệu cần thiết được ghi xuống đĩa cứng. Chúng tôi đề xuất Định Dạng Thống Kê Dữ Liệu Cho Bộ Nhớ (MSDF), một kỹ thuật xử lý trực tuyến được sử dụng trong giám sát bộ nhớ sử dụng của ứng dụng nhằm tiết kiệm dung lượng lưu trữ trong đĩa cứng trong khi vẫn lưu giữ đủ thông tin để giải quyết bài toán phân bổ tài nguyên cho ứng dụng. MSDF có thể giúp tiết kiệm hơn 95% dung lượng lưu trữ trong khi luôn đảm bảo hiệu quả phân bổ tài nguyên tùy thuộc vào tham số ε và MSDF có thể được mở rộng để giải quyết thêm nhiều bài toán vận hành khác hoặc tinh chỉnh để thích ứng trong việc giám sát và phân tích các thông số khác của ứng dụng.

Từ khoá: giám sát hệ thống, giám sát bộ nhớ, xử lý dòng dữ liệu, phân tích trực tuyến (online), phân bổ tài nguyên bộ nhớ

[1]*Phòng thí nghiệm tính toán hiệu năng cao, khoa Khoa học và Kỹ thuật máy tính (HPC Lab), Trường Đại học Bách Khoa Thành Phố Hồ Chí Minh (HCMUT), 268 Lý Thường Kiệt, Quận 10, Thành Phố Hồ Chí Minh, Việt Nam,*

[2]*Viện Khoa học và Công nghệ tiến tiến liên ngành (iST), Trường Đại học Bách Khoa Thành Phố Hồ Chí Minh (HCMUT), 268 Lý Thường Kiệt, Quận 10, Thành Phố Hồ Chí Minh, Việt Nam,*

[3]*Đại học Quốc gia Thành Phố Hồ Chí Minh (VNU-HCM), Phường Linh Trung, Thành Phố Thủ Đức, Thành Phố Hồ Chí Minh, Việt Nam.*

Liên hệ

La Quốc Nhựt Huân, Phòng thí nghiêm tính toán hiệu năng cao, khoa Khoa học và Kỹ thuật máy tính (HPC Lab), Trường Đại học Bách Khoa Thành Phố Hồ Chí Minh (HCMUT), 268 Lý Thường Kiệt, Quận 10, Thành Phố Hồ Chí Minh, Việt Nam,

Viện Khoa học và Công nghệ tiến tiến liên ngành (iST), Trường Đại học Bách Khoa Thành Phố Hồ Chí Minh (HCMUT), 268 Lý Thường Kiệt, Quận 10, Thành Phố Hồ Chí Minh, Việt Nam,

Đại học Quốc gia Thành Phố Hồ Chí Minh (VNU-HCM), Phường Linh Trung, Thành Phố Thủ Đức, Thành Phố Hồ Chí Minh, Việt Nam.

Email: huan@hcmut.edu.vn.

Trích dẫn bài báo này: Huân L Q N, Thìn N M, Lai N L D, Hùng N Q, Nam T. **MSDF: Định dạng thống kê dữ liệu cho bộ nhớ ứng dụng trong giám sát hệ thống**. *Sci. Tech. Dev. J. - Eng. Tech.* 2024; ():1-1.