CCF Transactions on High Performance Computing (2022) 4:321–338
https://doi.org/10.1007/s42514-022-00099-8

REGULAR PAPER

A tail-tolerant cloud storage scheduling based on precise periodicity detection

Yuxiao Han[1] · Jia Ma[1] · Fei Li[2] · Yubo Liu[1] · Nong Xiao[1] · Yutong Lu[1] · Zhiguang Chen[1]

Received: 30 November 2021 / Accepted: 8 April 2022 / Published online: 23 May 2022

Abstract
Cloud storage is a fundamental component of the cloud computing system, which significantly affects the overall performance and quality of service of the cloud. Cloud storage servers face the challenge of imbalanced workloads. According to our observations on the time series generated by cloud storage, we found that the imbalance workloads will dramatically increase the tail latency of data access in the multi-tenant scenario. The intuitive solution is to periodicity detect the imbalance storage nodes and re-balance the loads. However, there are four challenges to accurately detect load of storage in the cloud with multiple tenants since the load may change frequently in cloud. This paper proposes PrecisePeriod, a precise periodicity detection algorithm customized for multi-tenant cloud storage. It removes outliers through data preprocessing, employs the discrete wavelet transform to remove high-frequency noise while keeping frequency domain information, computes the candidate periodicity queue using the autocorrelation function, and determines precise period through periodicity verification. Then, we design a cloud storage load balancing scheduling strategy based on PrecisePeriod, and the evaluation shows that the PrecisePeriod scheduling significantly reduces tail latency while only bringing $1-2\%$ overhead.

Keywords Cloud storage · Time series · Periodicity detection · Scheduling · Tail latency

1 Introduction

Cloud computing has become a fast and effective solution with the explosive growth of data scale and applications. Modern tenant-oriented cloud services must meet strict Service Level Objectives (SLOs) (Sriraman et al. 2019), usually expressed as tail latency. Tail latency is generally caused by two reasons, one is made by garbage collection (Kim 2019) and memory management (Skarlatos et al. 2017) during system operation, and the other is caused by the high instantaneous load (Elyasi et al. 2017). To reduce costs, cloud service providers usually provide services to multiple tenants simultaneously, but the load superposition caused by multiple tenants poses a new challenge to tail-tolerant.

✉ Yuxiao Han
hanyx7@mail2.sysu.edu.cn

Jia Ma
majia5@mail2.sysu.edu.cn

Fei Li
renlei.lf@alibaba-inc.com

Yubo Liu
yubo.liu@nscc-gz.cn

Zhiguang Chen
zhiguang.chen@nscc-gz.cn

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China

[2] Alibaba Group, Beijing, China

To reduce the tail latency caused by load superposition, the scheduling method is usually used to re-balance the loads. Common resource scheduling methods include hash scheduling, greedy scheduling (Shi 2020), random strategy (Eltabakh 2011), heuristic strategy (Sellami et al. 2021), and so on. However, the previous scheduling methods only schedule according to the historical information, which can not ensure the stability of the results in the case of fewer scheduling times. The reason is the lack of prediction of future data, which is challenging to ensure the effectiveness of each schedule.

In the process of tenants using cloud computing services, a large number of time series are generated. We analyze the time series generated to better understand the challenge of load scheduling in the multi-tenant scenario. Because there are many periodic behaviors in tenants' activities, periodicity characteristics appear in the time series. Periodicity is a critical property of time series that has been applied to a variety of tasks, such as clustering (Vlachos et al. 2004; Kumar et al. 2002), classification (Vlachos et al. 2005), prediction (Theodosiou 2011), etc. Precise periodicity detection can better analyze the data and accurately predict the future situation, further used in scheduling.

However, performing precise periodicity detection in cloud storage is difficult due to the complexity and fluctuation of real-world time series in the multi-tenant scenario. The challenges are as follows: (1) Time series usually contain outliers, which may be errors in the data or in the statistical process and significantly impair the results of periodicity detection; (2) Time series quite often fluctuate due to noise; (3) Time series may be dynamic in nature and may change at any time in response to tenant behavior; (4) The majority of time series contain trend components, such as increasing or decreasing trend. Existing periodicity detection algorithms resort to four popular methods:

(1) Symbol periodicity detection method (Elfeky et al. 2005; Rasheed and Alhajj 2013): for the sequence composed of the finite symbol set, calculate the period of a single symbol or symbol set; the symbol periodicity detection method discretizes the data, converts the time series to a series made of finite symbol sets, and detects the periodicity of various modes, such as symbol periodicity, sequence periodicity, and segment periodicity. However, this method can only predict the range of future values but not the exact values themselves.
(2) Frequency domain method (Almasri et al. 2011): analyze the frequency of time series from the perspective of frequency domain and find out the dominant frequency as the period; the frequency domain method usually starts by transforming the time series from the time domain to the frequency domain using the discrete Fourier transform (DFT), then studies the spectrum structure and change law of the signal, and finds out the dominant frequency as the frequency corresponding to the periodicity. However, this method has the problem of spectrum leakage (Vlachos et al. 2005); when the frequency corresponding to the periodicity is not an integral multiple of the fundamental function, spectrum leakage occurs, resulting in interference across spectral lines and decreasing prediction accuracy.
(3) Time domain method (Wang et al. 2006): from the perspective of time domain, find the correlation distance between the time series and itself as the periodicity; which often employs the autocorrelation function (ACF) to determine the period of a time series by correlating it to itself. However, the ACF method tends to look for long periodicity and is seriously affected by outliers or trends, resulting in erroneous results.
(4) Frequency-time joint domain method (Toller et al. 2019): combine the frequency domain and time domain approaches to calculate the period; results from both frequency and time domains are mutually verified to obtain the final results. However, this method is limited by the defects of frequency and time domain methods and can not solve all the above problems. Generally speaking, the calculation results of the previous algorithms are not precise enough; we need to improve them and propose a precise periodicity prediction algorithm.

This paper proposes a precise periodicity detection algorithm called PrecisePeriod based on the frequency-time joint domain to predict the period and future time series. By observing and analyzing the time series, we present a preprocessing method for removing outliers from real datasets. We use discrete wavelet transform (DWT) to remove noise while retaining the original frequency domain information and calculating candidate periods through ACF. We design a periodicity verification method to select a period from the candidate periods and improve periodicity prediction precision by verifying the correctness and stability of the periodicity. In the process of periodicity detection, we first use a small part of data for calculation and then gradually increase the length of historical data to reduce the trend's impact and tolerate the periodicity's dynamic changes. Our PrecisePeriod algorithm shows better results on actual and synthetic datasets than several advanced periodicity detection methods. We have conducted extensive experiments on PrecisePeriod algorithm in Alibaba real datasets and found that 30% of the time series are periodic and can be accurately predicted.

We further apply PrecisePeriod to the load scheduling in multi-tenant cloud storage. The result is PreciseScheduling, a scheduling strategy that can accurately predict the access traffic and efficiently re-balance the loads, thus reducing the tail latency with low overhead. The evaluation shows that PreciseScheduling can ensure stability and achieve the best scheduling effect with fewer scheduling times.

The main contributions of this paper are as follows.

- We research the structure and characteristics of data access of multi-tenant cloud storage and analyze the challenges in load scheduling.
- We propose an efficient, precise periodicity detection algorithm, PrecisePeriod, to use periodicity verification with low overhead to ensure precise prediction periods and future time series.
- We apply PrecisePeriod to load scheduling in multi-tenant cloud storage and significantly reduce tail latency of data access in imbalance loads.

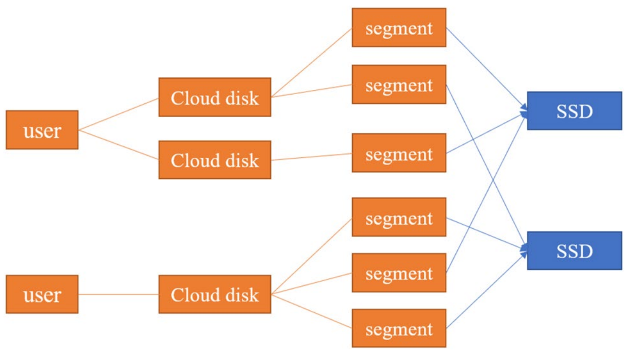

Fig. 1 Cloud environment storage system structure

The rest of the paper is organized as follows. Section 2 summarizes related works about periodicity detection and scheduling. Section 3 describes the design and implementation of PrecisePeriod. Section 4 describes the design and implementation of PreciseScheduling. In Sect. 5, we evaluate the effectiveness of PrecisePeriod and PreciseScheduling methods. Finally, Sect. 6 summarizes the work of this paper.

2 Background and related work

2.1 Tail latency in cloud storage

At present, cloud computing service providers usually use SSDs as storage nodes to provide services for multiple tenants simultaneously on the premise of meeting QoS requirements, as shown in Fig. 1. There are usually multiple tenants in the cloud computing scenario, and each tenant uses one or multiple cloud disks. According to the design of block storage, each cloud disk is usually divided into various 32GB segments and stored in SSDs. Recent research shows that SSDs do not consistently maintain their ideal performance in practical use (Yan et al. 2017). One of the reasons is that load superposition causes numerous requests in the SSD to queue up, resulting in high tail latency (Elyasi et al. 2017).

The reason can be analyzed from queuing theory (Cooper 1981). According to queuing theory, tail latency D is mainly affected by unit time SSD read/write speed S and unit time read/write operation arrival speed A. The calculation formula of tail latency is as follows:

$$D = \frac{1}{S - A}. \tag{1}$$

When S is fixed, D is positively correlated with A. With the growth of A, the growth rate of D increases significantly. We also use the simulated SSD with the configuration of Table 3 to simulate the change of tail latency D under different read-write operation speeds A when unit time SSD read/write speed S remains unchanged, as shown in Fig. 2. We statistic

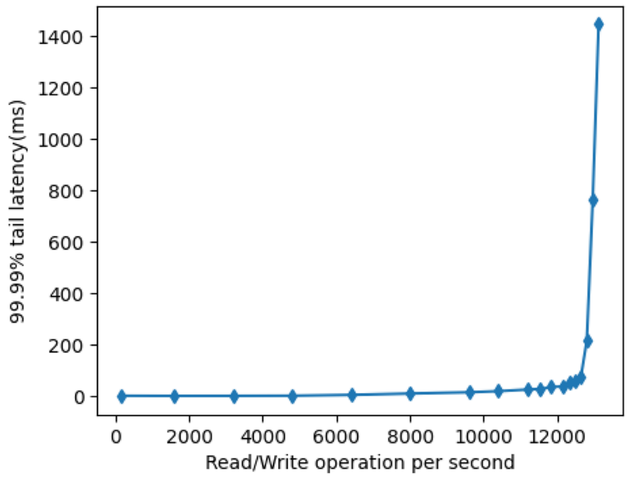

Fig. 2 Relationship between tail latency and read/write operation, 64 consecutive sectors per random request

the tail latency changes under different A, and each operation reads/writes 64 consecutive sectors.

We can see that when A is less than 12000, the change of D is not apparent. When A is greater than 12000, the waiting queue length of read-write requests in SSD increases significantly, causing the tail latency to grow exponentially.

Cloud computing service providers need to meet strict QoS requirements, and tail latency is a problem that must be solved(Didona and Zwaenepoel 2019). The simplest way to reduce the tail latency is to improve the performance of the equipment, increase the service speed and keep the average load at a low level to avoid the tail latency caused by the high instantaneous load. However, this method undoubtedly increases additional costs and wastes resources. Because most SSDs do not reach peak load, and only a few SSDs have high instantaneous load caused by load superposition. Avoiding load superposition through scheduling is an economical and reasonable solution to solve the tail latency.

2.2 Scheduling algorithm

In the aspect of data scheduling strategy, the existing research mainly focuses on the placement of replicas and blocks in distributed storage systems. Data scheduling can be separated into two types based on frequency: dynamic data scheduling and static data scheduling. Dynamic data scheduling strategy is to schedule the existing data after each data request. When the amount of data is small, dynamic data scheduling is effective. However, the system overhead caused by frequent data scheduling is enormous for the massive amount of data in a distributed storage system. Static data scheduling strategy assumes that the data access is constant for some time. The proposed data scheduling strategy can remain effective for some time. Data migration can realize the overall adjustment when the data layout is obviously inappropriate.

The static data scheduling method is usually used to reduce peak traffic to avoid the high instantaneous traffic caused by the multi-tenant load superposition.

The commonly used method is based on the greedy (Shi 2020) or random strategy (Eltabakh 2011) scheduling method, based on historical information to avoid possible peak traffic in the future. However, the problem with these methods is the lack of accurate prediction of future traffic, which leads to unstable scheduling effects.

SP (Lee et al. 2000) and PB (Madathil et al. 2008) strategy can minimize the variance of service time and realize load balancing of distributed storage clusters. However, these two strategies assume that data access is independent, which is unreasonable because collaborative access is common in data-intensive applications. Hence, the superposition of data access traffic often occurs, which can not ignore in data layout.

DRAW (Wang et al. 2014) and CLUST (Vengadeswaran and Balasundaram 2020) strategy analyzes the characteristics of a single dataset and then puts forward the scheduling strategy according to the features. However, the shortcomings of these two algorithms are apparent: they are only applicable to specific datasets, and the scalability of the method is poor.

HDFS default data scheduling strategy can only ensure the storage balance of each node. Still, it cannot guarantee the I/O load balance, resulting in a series of periodic data blocks that may be placed on the same node, resulting in load superposition.

By analyzing the actual workload, we find that there is a lot of periodic behavior. It is possible to avoid peak overlap with fewer scheduling times if we can precisely detect periodicity and future peaks, thereby reducing tail latency.

2.3 Periodicity prediction

In different works of literature, periodicity prediction is also called seasonal length estimation (Toller et al. 2019) or segment periodicity detection (Rasheed and Alhajj 2013), which all represent the same meaning; we use periodicity detection in this paper. To calculate the periodicity of time series, many different methods have been proposed. Most of these methods can be divided into four categories: (1) Symbol periodicity detection method; (2) Frequency domain method; (3) Time domain method; (4) Frequency-time joint domain method.

For symbol periodicity detection, a suffix-tree-based algorithm (Rasheed and Alhajj 2013) and a convolution-based algorithm (Elfeky et al. 2005) are proposed. However, these methods only perform well in the synthetic dataset and are seriously affected by outliers and noise. At the same time, these methods can not accurately predict specific values, which is not conducive to subsequent scheduling.

Frequency domain method usually uses DFT to calculate the periodogram of time series. However, when the periodicity is long, the calculation result of the periodogram is inaccurate. When the selection of the basis function is incorrect, there is the problem of spectrum leakage (Vlachos et al. 2005). To improve the effectiveness of the periodogram, (Almasri et al. 2011) proposes the use of discrete wavelet transform (DWT) to replace DFT, but the result of the periodogram is still inaccurate. (Mezic and Surana 2016; Lange et al. 2021) proposes the spectral analysis method, but the calculation results are unstable and will be affected by outliers and trends.

Time domain method uses ACF to calculate the correlation between the time series and itself (Tian 1988). In short, if the period of the time series is T, the result of the backward shift length T of the time series is highly correlated with itself, and it appears as a local peak in the calculation result of ACF. Based on the characteristics of ACF, autocorrelation peak is proposed as periodicity prediction (Wang et al. 2006) . However, ACF tends to look for long periods. When the period of the time series is T, there will be multiple local peaks in the ACF calculation results, such as T, 2T, which will affect the results of period detection. An autocorrelation zero distance method is proposed to solve this problem of multiple local peaks (Toller et al. 2019) . However, it can not solve the problem of ACF itself and still produces wrong results due to outliers and noise.

Frequency-time joint domain uses both frequency domain and time domain information. (Toller et al. 2019) proposes the SAZED method, which uses time domain and frequency domain information to calculate the candidate period and uses different properties of ACF to verify the periodicity. However, the disadvantage of this method is that the precision of periodicity detection results is not high and will get a lot of inaccurate results. (Mitsa 2010; Vlachos et al. 2005) put forward the AUTOPERIOD method, which uses a periodogram to calculate the candidate period in frequency domain and ACF to identify the accurate periodicity in time domain. However, this method is used for multiple periodicity detection. We only need to find the most critical periodicity and predict the future time series.

Although there are many periodicity detection methods, the precision of their final results is not high, and it is difficult to be used for accurate scheduling due to the influence of outliers, noise, trends, and other factors. We want to find a method that can precisely detect the periodicity and use the predicted data in scheduling.

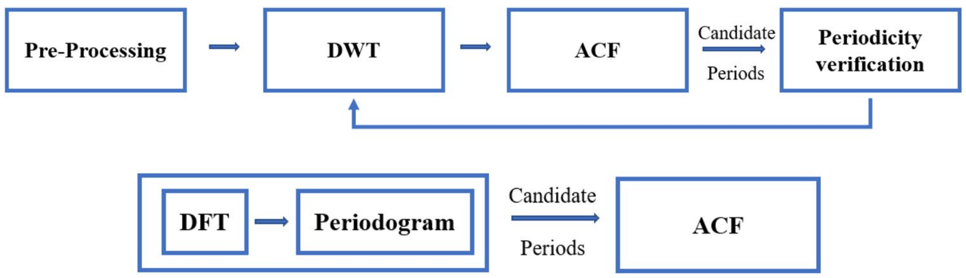

Fig. 3 Overview of periodic detection algorithm flow. The upper plot shows PrecisePeriod algorithm, and the lower plot shows other frequency-time joint domain algorithms. *DWT* discrete wavelet transform, *ACF* autocorrelation function, *DFT* discrete Fourier transform

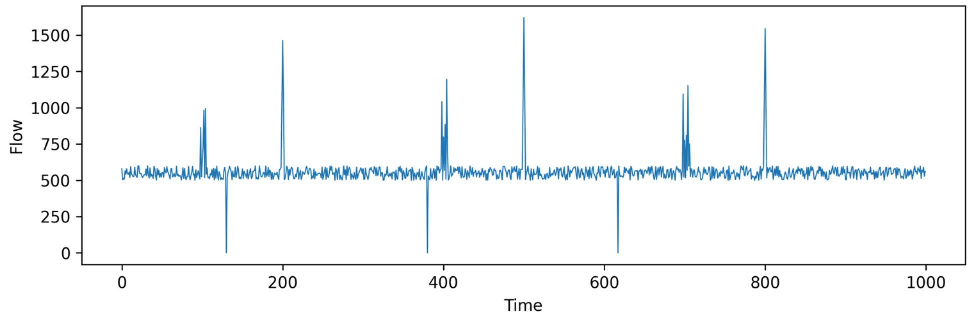

Fig. 4 Original time series from data capture

3 PrecisePeriod design

3.1 Method overview

In this section, we will discuss our proposed PrecisePeriod algorithm. We define the periodicity model of time series as the following formula:

$$y_t = s_t + n_t + a_t, \quad t = 0, 1, \dots, N-1, \tag{2}$$

where y_t represents the time series at time t, s_t represents the periodic sequence with period T at time t,n_t represents the noise at time t, and a_t represents the outliers at time t. The PrecisePeriod proposed in this paper is to precisely identify the periodicity's length after removing the noise.

The algorithm mainly includes the following four parts, as shown in upper part of Fig. 3 : (1) Data preprocessing to remove outliers in time series; (2) Employ discrete wavelet transform (DWT) to decompose and reconstruct time series to remove high-frequency noise; (3) Use autocorrelation function (ACF) to detect periodicity; (4) Compare the predicted data with the actual data to check the accuracy and stability of periodicity.

Compared with other periodicity detection methods, our PrecisePeriod mainly adds two steps: pre-processing and periodicity verification, of which periodicity verification plays the most critical role in improving accuracy. The calculation steps of AUTOPERIOD (Vlachos et al. 2005) are shown in the lower part of Fig. 3. Although both frequency domain and time domain information are used simultaneously, the accuracy of the calculation results is not high due to the influence of outliers and trends. Subsequent people also proposed pre-processing methods, but only pre-processing alone cannot guarantee the precision of the results. We creatively put forward the process of periodicity verification, which ensures the precision of the results by verifying the correctness and stability of the period many times.

3.2 Data preprocessing

In the real world time series, the observed results usually contain various errors, and the time series we use will inevitably produce errors in the capture process. Excluding the random influence of accidental error, the systematic error in the acquisition process is an important part that can not be ignored. For example, when the network is under high load, the data capture program will temporarily stop to ensure the transmission of tenant data. In that case, the output result will produce the phenomenon of log loss, which will be reflected in the captured time series as temporary data loss. When there is usually traffic in the data, a temporary data loss will produce a significant minimum, as shown in Fig. 4, resulting in abnormal data fluctuation. This fluctuation will bias the calculation results of ACF, resulting in misleading periodic information. Therefore, the outliers caused by this systematic error need to be removed by data preprocessing.

Due to the load superimposition of multi-tenant on the cloud disk, the instantaneous traffic peak may occur in the traffic, as shown in Fig. 4. For example, the tenants will back up the recently generated data to the cloud storage every other period or summarize the newly developed data for data analysis, so this instantaneous traffic peak needs to be reserved. However, it may also produce continuous peaks, which is not conducive to ACF calculation and will

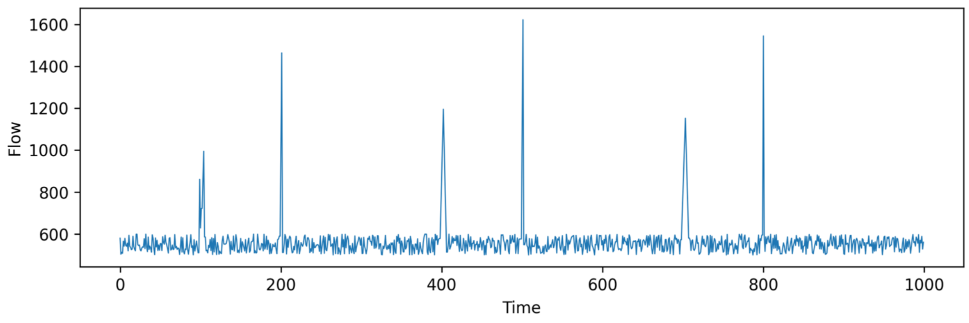

Fig. 5 Time series after preprocessing

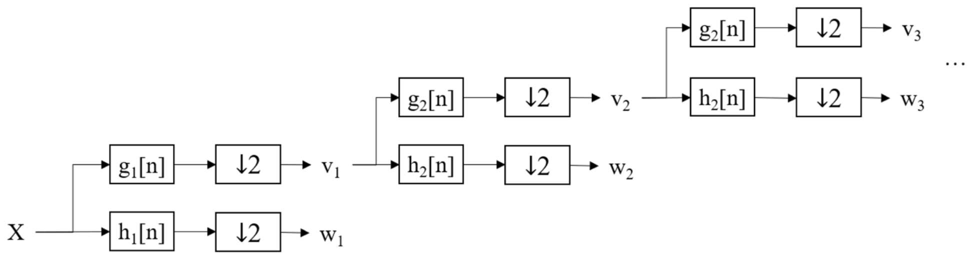

Fig. 6 Principle of DWT decomposition

lead to errors in periodicity detection. To avoid this situation, we need to smooth the time series while retaining the change and fluctuation of flow to minimize the occurrence of glitches in the oscillation.

The method used in this paper is to form a new series based on the change of data, detect the data through Tukey's test (Tukey 1977) method, and different types of outliers are handled differently. Tukey's test method calculates the upper quartile Q_3 and lower quartile Q_1 of the data and then we can define the outliers as any value outside this range according to the coefficient k : $[Q_3 + k(Q_3 - Q_1), Q_1 - k(Q_3 - Q_1)]$. This paper uses Tukey's test method to obtain the result $k = 1.5$ to find the outliers in the time series.

The change series of data is used to generate outliers position series in order to distinguish different types of outliers. For the minimum value caused by systematic error, the outliers position series is a continuous sequence of length 2. The first value is less than the minimum range, and the second is greater than the maximum range. We use the average value of data nearby to replace the outliers. For other types of outliers, the homogenization method is adopted. The original outliers series is replaced by the uniformly spaced increase and then decrease sequence to smooth the time series while retaining the change message of flow. Through data preprocessing, we can get the time series corresponding to Fig. 5.

3.3 DWT denoise

We use discrete wavelet transform (DWT) to denoise the input time series at different levels (Cai and Harrington 1998), convenient for periodic detection. The reason why DWT is used instead of discrete Fourier transform (DFT) is that DWT has the following advantages: (1) DWT can well retain the peak and abrupt part of the valuable signal required in the original signal, which has higher robustness, but DFT can easily lead to the distortion of the original signal in the recovery process; (2) DWT has good time-frequency localization characteristics; (3) DWT can well characterize the non-stationary characteristics of signals.

The process of DWT denoise includes three steps: (1) Decompose the time series into wavelet coefficients (from the high-pass filter) and scaling coefficients (from the low-pass); (2) Zero filling or compression of the signal according to some criteria; (3) The compressed coefficients are inversely transformed into the original domain to obtain the denoised time series.

We use the common DWT based on Daubechies wavelet (Daubechies 1992; Percival and Walden 2000) to denoise the time series. When calculating DWT for time series $X = (X_0, X_1, ..., X_{N-1})$ of length N, j layer wavelet coefficients w_j are given by

$$w_{j,k} = \sum_{l=0}^{L-1} h_l w_{j-1,2k+1-l}, \quad k = 0, 1, ..., N_j - 1, \tag{3}$$

and j layer scaling coefficients v_j are given by

$$v_{j,k} = \sum_{l=0}^{L-1} g_l v_{j-1,2k+1-l}, \quad k = 0, 1, ..., N_j - 1, \tag{4}$$

where $N_j = N2^{-j}$ and $v_0 = X$, as illustrated in Fig. 6.

For time series, the low-frequency component is vital. It often contains the characteristics of time series, while the high-frequency component corresponds to the details or differences of time series. For periodicity detection tasks, although high-frequency components have a small number of periodic information, most of them are noise, which means removing high-frequency parts can increase the accuracy of ACF detection. Therefore, we use the zero filling method to remove the wavelet coefficients calculated by each layer and only use the scaling coefficients for wavelet reconstruction to obtain different levels of denoised time series.

$$ACF_k = \frac{1}{(N-k)\sigma^2}\sum_{t=1}^{N-k}(X_t-\mu)(X_{t+k}-\mu),\quad k = 0, 1, ..., N-1, \tag{5}$$

where μ is the mean of the time series, and σ^2 is the variance of the time series.

After obtaining the calculation results of ACF, we select the maximum value in the local peak as the period obtained by ACF calculation. For different levels of denoised time series, we calculate different ACF results and get the detection values of multiple periods. When there are repeated values in periods, we take the modal number as the detection results of the period. When periods present a polarization distribution, there is an error in the DWT process, which will lead the ACF to predict long periods and short periods, respectively. We take the two levels of the detection result as the period. When there are outliers in periods, we remove the outliers and take the mean value of other detection results as the period.

Algorithm 1 PrecisePeriod

```
Input: time series: $X = (X_0, X_1, ..., X_{N-1})$, PCC threshold: $PCC\_std$, Fitting length per increase: $fit\_range$
Output: periodicity: $T$
1: $fit\_length = fit\_range$
2: $RemoveOutliers(X)$
3: while $True$ do
4:     if $Predicted\_data\_length = N$ and $Predicted\_data\_PCC > PCC\_std$ then /* Predict periodicity success */
5:         return T
6:     end if
7:     if $Predicted\_data\_PCC > PCC\_std$ then
8:         $Predicted\_data,\ Predicted\_data\_PCC = Prediction(X,\ T)$ /* Predict data according to periodicity T */
9:     else /* Predict periodicity */
10:        if $fit\_length > Predicted\_data\_length$ then /* Predict period using partial time series from scratch */
11:            $DWTDenoise(X[0 : fit\_length])$ /* Using DWT remove high-frequency noise */
12:            $T\_list = ACFPredictPeriodicity(X[0 : fit\_length])$ /* Using ACF to predict periodicity of time series */
13:            $PCC,\ T = fit(X[0 : fit\_length],\ T\_list,\ PCC\_std)$ /* Verify multiple periodicities to find the most suitable periodicity */
14:        else /* Use the time series near the current prediction result to predict periodicity */
15:            $DWTDenoise(X[Predicted\ data\ length - fit\_length : Predicted\ data\ length])$
16:            $T\_list = ACFPredictPeriodicity(X[Predicted\ data\ length - fit\_length : Predicted\ data\ length])$
17:            $PCC,\ T = fit(X[Predicted\ data\ length - fit\_length : Predicted\ data\ length],\ T\_list,\ PCC\_std)$
18:        end if
19:        $fit\_length += fit\_range$
20:        if $fit\_length > N$ and $PCC < PCC\_std$ then /* Predict periodicity fail */
21:            return $T = 0$
22:        end if
23:    end if
24: end while
```

3.4 ACF periodicity detection

After obtaining different levels of denoised time series through DWT, we use autocorrelation function (ACF) (Brockwell 2009; Box 2015) to calculate the period of each denoised time series. ACF can provide us with the autocorrelation value of any sequence with lag values. Intuitively, it describes the degree of correlation between the current value of the time series and its future value. Time series contains trends, seasonality, periodicity, residuals, and other components. ACF will consider all these components when looking for correlation, including direct and indirect correlation information.

When we have time series $X = (X_0, X_1, ..., X_{N-1})$ of length N, ACF_k is given by

3.5 Periodicity verification

On the one hand, the ACF tends to find long periods and is prone to outliers and noise; on the other hand, the denoised time series will also produce errors due to errors in the DWT process or outliers in time series. Therefore, we need to verify the accuracy and stability of ACF results. For detection accuracy, we use periodicity to predict the future time series.

If the prediction results are close to the actual time series, the periodicity detection results are correct. For detection stability, we use periodicity to predict longer-term future time series. When the prediction results are close to the actual time series for a long time, it indicates that the periodicity can exist stably for a long time and is relatively stable.

For the time series, we take a small part of the time series to predict periodicity, then decide whether to use more time series to detect the periodicity or verify its stability based on the prediction results.

Take out part of the time series and predict the periodicity through DWT and ACF. ACF may give detection results of multiple periodicities. We need to verify each detection result in the time series to find the most suitable periodicity. When we have obtained the detection value of the periodicity, the prediction of future time series is obtained by weighting the historical time series. For time series $X = (X_0, X_1, ..., X_{N-1})$ of length N, after the periodicity T is obtained, the prediction result of time t is $X_t = (X_{t-T} + X_{t-2T} + X_{t-3T})/3$. When the historical time series length is less than 3 times of periodicities, use the actual time series as the prediction result. Until there are enough historical time series to verify the accuracy of the periodicity.

After multiple periodicities are obtained through different levels of denoised time series, the Pearson correlation coefficient (PCC) (Berthold and Höppner 2016) between the prediction results and the actual time series is calculated to represent the correlation. The closer the calculation results are to 1, the stronger the correlation is. The periodicity corresponding to the maximum PCC is taken as the detection periodicity. If the result of PCC is higher than the predefined threshold, it indicates that the prediction result has a strong correlation with the actual time series which indicates that the detection of the periodicity is correct. So the periodicity can be used to predict longer-term time series to verify the stability of the periodicity. On the contrary, if the result of PCC is lower than the predefined threshold, it indicates a specific difference between the predicted result and the actual time series which means the actual time series cannot be represented by the prediction result. Therefore, it is necessary to add historical time series and predict the periodicity again.

The accuracy of periodicity prediction will be low if the PCC threshold is set too high. The precision of periodic detection will be low if the PCC threshold is set too low. According to the experimental results, the prediction effect is the best when the PCC threshold is 0.7. If the prediction fails many times, we will increase the growth rate of the length of historical time series to speed up the prediction periodicity.

Due to the application change of the tenants on the cloud disk, periodicity will be changed simultaneously. For example, the tenant switches from backing up every 1 hour to backing up every 2 hours. Although the periodicity has been successfully predicted in the detection process, the detection results over time are imprecise. We consider using the historical time series near the current detection stage to predict the periodicity again in order to adapt to the change of the periodicity. This helps to exclude some old historical time series and predict the latest periodicity to avoid the impact of the periodicity contained in the old historical time series. Then we use the method mentioned before to verify the accuracy and stability of the periodicity after detection. The complete algorithm is described in Algorithm 1.

Algorithm 2 PreciseScheduling

```
Input: time series: $Y = (X^0, X^1, ..., X^{M-1})$, PCC threshold: $PCC\_std$ , scheduling times: $sch\_times$
Output: PreciseScheduling result: $move\_list$
1: for $i \in \{0, 1, ..., M\}$ do
2:     $T = PrecisionPeriodicityPrediction(X^i,\ PCC\_std,\ X^i\_length/8)$
3:     if $T \neq 0$ then /* Predict periodicity success */
4:         $Y\_Predicted.add(Prediction(X,\ T))$
5:     end if
6: end for
7: Generate $SSD\_Predicted$ from $Y\_Predicted$ /* Get the prediction results of each SSD */
8: for $i \in \{0, 1, ..., sch\_times\}$ do
9:     Find $source\_SSD$ from $SSD\_Predicted$ /* peak traffic SSD */
10:    Find $segment$ /* the most contribute to peak traffic SSD */
11:    Find $target\_SSD$ /* the lowest peak traffic SSD after adding segment */
12:    $move(segment,\ source\_SSD,\ target\_SSD)$ /* move segment in $SSD\_Predicted$ */
13:    $move\_list.add(segment,\ source\_SSD,\ target\_SSD)$
14: end for
15: return $move\_list$
```

4 PreciseScheduling based on PrecisePeriod

This paper aims to avoid the tail latency caused by load superposition of multiple segments on SSDs for the current cloud computing storage structure, as shown in Fig. 1. Specifically, we want to predict future load superposition, determine which segment causes it, and schedule the segment to another SSD. It is possible to minimize high instantaneous load induced by load superposition, thereby reducing tail latency. Specifically, PreciseScheduling is divided into two distinct stages: (1) Using the results of PrecisePeriod, forecast future traffic of SSD and segment; (2) Identify the SSD with the highest peak traffic and the segment that causes it, then schedule it to another SSD to lower peak traffic.

For the existing segment time series, we select the 20% segment with the highest traffic according to the Pareto principle and use PrecisePeriod to predict the periodicity of the segment. For unpredictable parts, the traffic situation of this part in the future is unknown, and the results obtained by scheduling them are unstable, which may be better or worse. To consider the robustness of PreciseScheduling, we assume that the traffic of unpredictable segments conforms to a uniform distribution, which is distributed to each SSD by a hash method. This part of the segments is not considered in the subsequent scheduling process.

For predictable segments, we predict the read traffic and write traffic of the segment, respectively, in the prediction process. We use the periodicity of read/write traffic to predict write/read traffic if only read/write traffic can be forecasted. The sum of the predicted values of read and write traffic is used as the predicted traffic of segment for scheduling because there is no significant difference in read-write latency for SSDs. Then we predict the traffic in the next 6 h and combine the traffic according to the SSD where it is located to forecast the future traffic of SSD. Six hours are reserved for scheduling because every schedule needs to move 32 GB of data on SSD, so the scheduling interval cannot be too short. At the same time, the periodicity is relatively stable in the short term but unstable in the long term, so it needs to be predicted again at regular intervals. In general, we choose 6 hours as the scheduling interval.

After the prediction results of SSD and segment are calculated, we use the greedy idea to schedule them. Firstly, find the SSD with the most enormous peak traffic in the next 6 h, and find the segment with the most significant impact on the peak traffic in the SSD as the scheduling source. The peak traffic here refers to the instantaneous traffic rather than the total traffic. The primary reason for tail latency is an overwhelming immediate load created by instantaneous traffic. More specifically, we divide 6 h into 10 min intervals to find the SSD with the highest total flow in the gap and the segment causing the instantaneous flow. Then, the traffic information of the segment is added to the other SSDs to find the SSD with the lowest peak traffic as the scheduling target. If the peak traffic of the scheduled target SSD is higher than that of the original SSD, it will not be scheduled, and the segment will be added to the non-scheduling queue to find a new segment for scheduling.

The number of scheduling times is determined by the additional overhead caused by scheduling. The scheduling overhead is due to the extra traffic generated by scheduling. The scheduling is in the segment unit for the cloud computing storage structure and moves 32GB of data each time. We need to ensure that the additional overhead generated by PreciseScheduling is within an acceptable range. For example, the scheduling overhead caused does not exceed 5% of the original traffic. Therefore, we can calculate the maximum number of scheduling times according to the scheduling overhead and the size of the actual traffic. The complete algorithm is described in Algorithm 2.

5 Experiments and discussions

5.1 PrecisePeriod performance

Our experiments evaluate the performance of PrecisePeriod with other advanced periodicity detection algorithms on public real-world and synthetic datasets, respectively. We also study how each component we proposed contributes to PrecisePeriod.

5.1.1 Datasets

Public datastes. We use CRAN time series data as public datasets (Toller et al. 2019) to facilitate the comparison with other periodicity detection algorithms. It contains time series in various fields, such as economic indicators and environmental measurement, etc. It has 82 time series with a length range from 16 to 3024 and period length ranges from 2 to 52.

Synthetic datastes. We generate a synthetic time series with a length of 1000 with complex patterns, including one periodic component, multiple outliers, variation trend, and noise. Specifically, we first create a sine wave with a period between 50 and 200 whose amplitude changes randomly. The amplitude change is to better simulate the actual situation. There are fluctuations in the actual data, and there will be no perfect periodicity. Next, we generate outliers whose proportion is randomly generated from 0 to 0.05 and add them to the sine wave. Then Gaussian noise is added to the time series, the noise variance is between 0 and 0.5. Finally, add a random linear trend, linear increase, or linear decrease. In the synthetic data, we generate 1000 time series; one of the synthetic time series is shown in the Fig.7.

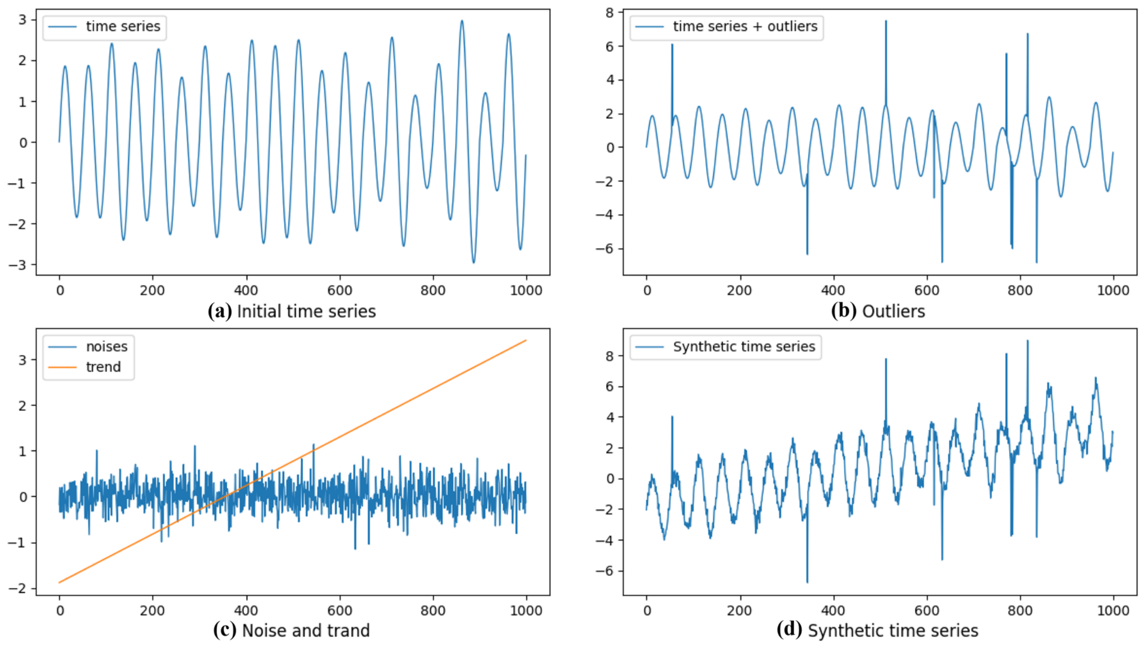

Fig. 7 The generation of synthetic data. **a** Create a time series consisting of sine waves with randomly varying amplitude; **b** Add random outliers; **c** Generate random Gaussian noise and trend; **d** Add noise and trend to generate synthetic time series

Table 1 Accuracy and F1 score comparisons of different periodicity detection algorithms on CRAN dataset and synthetic dataset

Algorithm	CRAN dataset				Synthetic dataset			
	±0%		±2%		±0%		±2%	
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
FindFrequency	0.439	0.486	0.439	0.486	0.097	0.097	0.359	0.361
SeasonLength	0.146	0.150	0.146	0.150	0.011	0.011	0.037	0.037
Sazed$_{opt}$	**0.524**	0.524	**0.537**	0.537	0.254	0.254	0.584	0.584
Sazed$_{maj}$	0.439	0.444	0.488	0.494	0.144	0.144	0.430	0.430
PrecisePeriod	0.500	**0.656**	0.512	**0.672**	**0.303**	**0.304**	**0.743**	**0.746**

Bold values indicate better results than other methods

5.1.2 Comparison with existing algorithms

We compare PrecisePeriod with the other four algorithms: (1) findFrequency (Hyndman and Athanasopoulos 2018); (2) seasonLength (Toller and Kern 2019); (3) SAZED$_{maj}$; and (4) SAZED$_{opt}$ (Toller et al. 2019). We compare the accuracy and F1 value of several algorithms at different datasets and levels. We first evaluate whether the periodicity detection results match the actual periodicity entirely, and the results are shown as ±0% in Table 1. Then evaluate the periodicity detection results within the actual periodicity's ±2% tolerance interval. The detection periodicity in this interval can also approximately predict future data.

Table 1 shows the accuracy and F1 score of different algorithms in two data sets. In CRAN datasets, the reason for the low detection accuracy and F1 score of the seasonLength algorithm is the algorithm overfits a specific dataset, resulting in inferior performance in other datasets. The accuracy of findFrequency and SAZED algorithms is relatively close to the F1 score, indicating that these two algorithms predict more results as much as possible to improve the detection accuracy, resulting in low precision and recall. In contrast, our proposed PrecisePeriod algorithm can improve the detection accuracy as much as possible through the periodic verification method and ensure high detection precision and F1 score simultaneously. In the synthetic dataset, the detection effect of the findFrequency algorithm decreases significantly. The reason is that findFrequency uses spectral decomposition and autoregressive methods to predict periodicity, which is seriously affected by outliers, resulting in poor detection effect. The SAZED method still guarantees a certain degree of accuracy and F1 score, but it is lower than

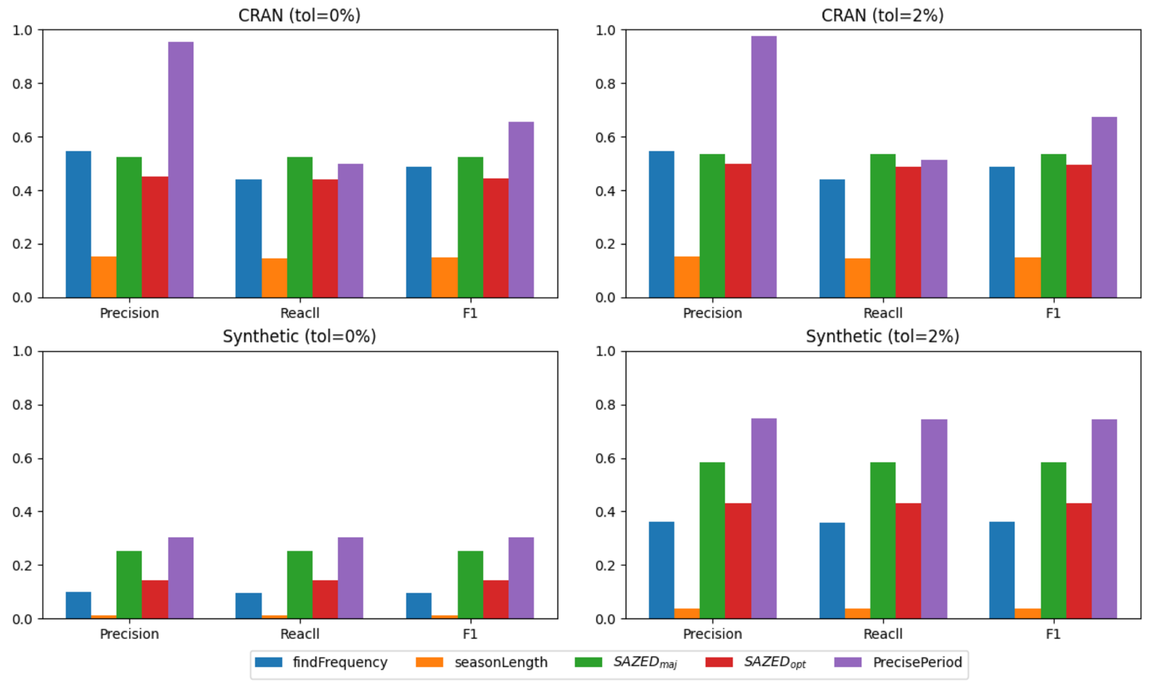

Fig. 8 Precision, recall, and F1 of different periodicity detection algorithms on CRAN dataset and synthetic dataset

Table 2 Ablation experiment of the PrecisionPeriodicity on CRAN dataset and synthetic dataset

Algorithm	CRAN dataset				Synthetic dataset			
	±0%		±2%		±0%		±2%	
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
No preprocessing	0.476	0.619	0.488	0.635	0.276	0.279	0.711	0.719
Single periodicity verification	0.341	0.500	0.341	0.500	0.292	**0.328**	0.636	0.714
PrecisePeriod	**0.500**	**0.656**	**0.512**	**0.672**	**0.303**	0.304	**0.743**	**0.746**

Bold values indicate better results than other methods

our proposed algorithm. Overall, our proposed algorithm achieves the best performance in accuracy and F1 score.

To further analyze the performance of various algorithms, we plot the precision, recall, and F1 score of different algorithms in CRAN dataset and synthetic dataset in Fig. 8. It can be observed from the figure that our proposed PrecisePeriod algorithm achieves the best effect in the three indicators. Especially in CRAN dataset, our proposed algorithm achieves a precision rate of nearly 1.0. It achieves the goal of precise periodicity detection, which is helpful to schedule using PrecisePeriod results.

5.1.3 Ablation experiment

To further understand the contribution of each part of our PrecisePeriod algorithm, we compare the performance of the PrecisePeriod algorithm with the following two versions: (1) No Preprocessing: this version has no preprocessing process, and the rest are the same as PrecisePeriod algorithm; (2) Single Periodicity Verification: this version modifies the periodicity verification process. Because our algorithm must select the final result from the candidate periods through periodicity verification, the periodicity verification part cannot be removed entirely. This version inputs all time series into the calculation forecast period at one time and uses single times periodicity verification to select the final period. It lacks the process of predicting and verifying future data with the period and the rest are the same as PrecisePeriod algorithm.

Table 2 shows the detection accuracy and F1 score of different algorithm versions in two datasets. Skipping the preprocessing process will cause the outliers in time series cannot be removed, thus affecting the calculation of autocorrelation function and resulting in the decline of accuracy and F1 score of periodicity detection in various cases. After modifying the periodicity verification process, directly using

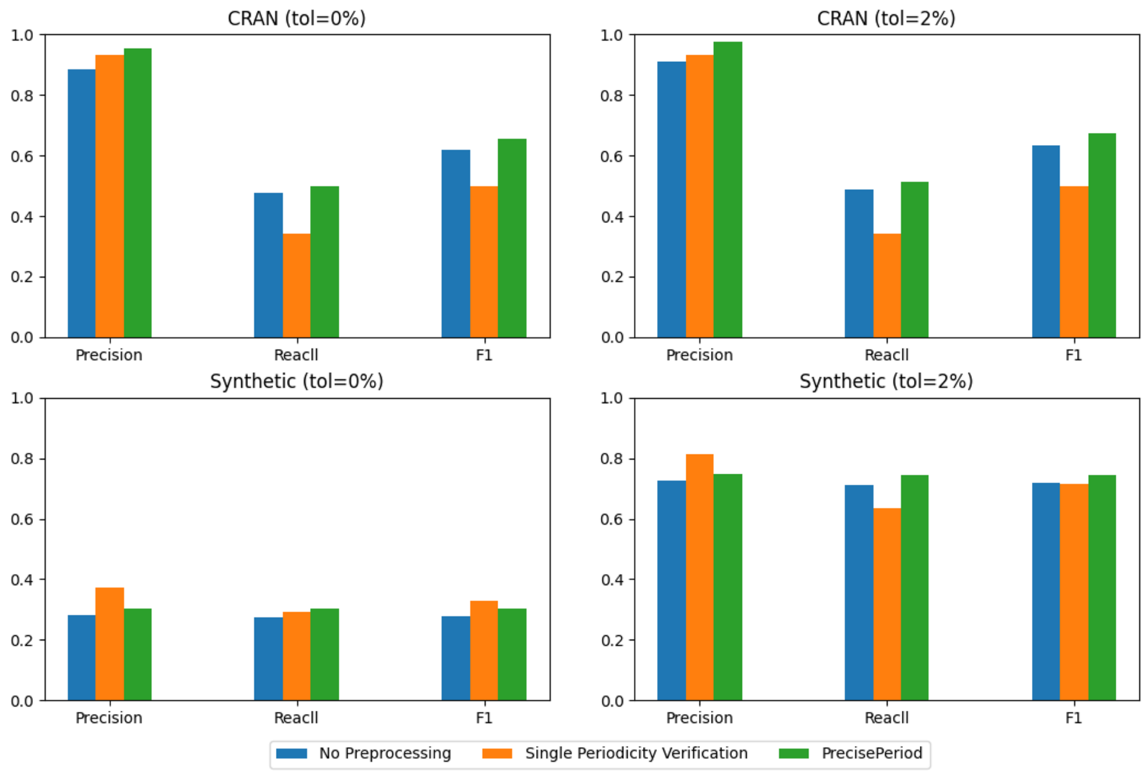

Fig. 9 Precision, recall, and F1 of ablation experiment on CRAN dataset and synthetic dataset

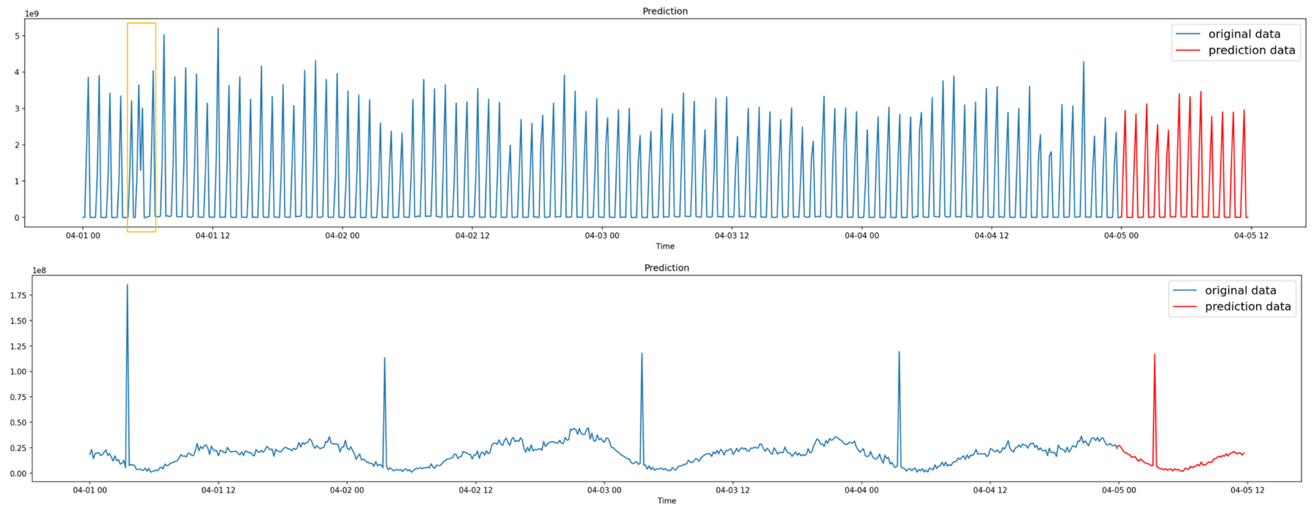

Fig. 10 Actual time series: original data and prediction results of single time series

the entire time series to predict the periodicity will lack the details analysis and lead to missing the excavation of short periodicity and preferring to find long periodicity. At the same time, this method is affected by the trend in the time series. An increasing or decreasing trend in time series will affect the periodicity detection, resulting in the decline of detection accuracy.

To further analyze the impact of each part on the performance of the algorithm, we plot the precision, recall, and F1 values of different versions of the algorithm in CRAN dataset and synthetic dataset in Fig. 9. It can be observed from the figure that the values of three indicators can be increased at the same time through preprocessing. The detection

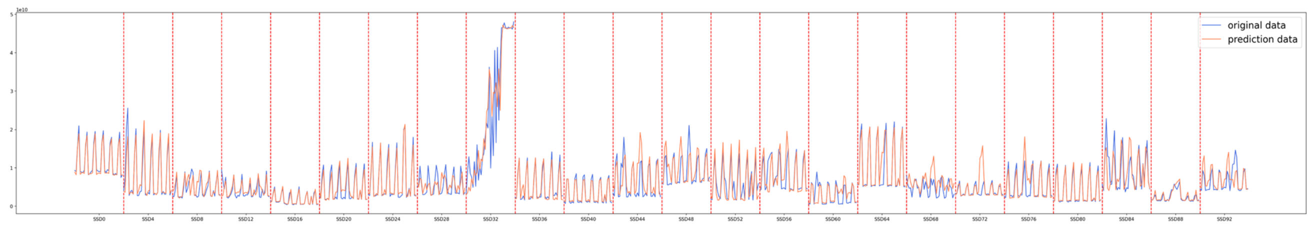

Fig. 11 Actual time series: original data and prediction results of part of SSD

Table 3 Configuration of the simulated SSD

SSD Organization	Host interface: PCIe 3.0 (NVMe 1.2) User capacity: 480GB 8 channels, 4 chips per channel QueueFetchSize = 512
Flash communication interface	ONFI 3.1 (NV-DDR2) Width: 8bit, Rate: 333 MT/s
Flash microarchitecture	8 KiB page, 448 B metadata per page, 256 pages per block, 2048 blocks per plane, 2 places per die
Flash access parmeters	Read latency: 75 s, Program latebcy: 750 Us, Erase latency: 3.8 ms

results can be more stable through periodicity verification, and periodicity detection accuracy can be improved.

5.1.4 Experiment on real datasets

We extensively evaluate our algorithm on Alibaba real datasets. We discovered that about 30% of the time series in the actual cloud computing load are periodic. We can accurately anticipate future time series based on the periodicity, as illustrated in Fig. 10. The figure shows two time series from Alibaba with apparent periodicity and relative stability. The blue line represents the original time series, while the red line represents the predicted result. Our method successfully finds the periodicity of time series and predicts the future data according to the original data. In the initial part of the first time series, we find that it has a small periodic change, which is only a mild change, and there are more complex situations in the actual time series. Our algorithm can tolerate this change through periodicity verification and can still predict the correct periodicity.

To verify the accuracy of the prediction results more broadly, we summarized all the results to achieve the overall traffic prediction for different SSDs, as shown in Fig. 11. The figure shows the predicted outcome of some SSDs, which has the expected time series and actual time series of the expected success part only. We add up the time series for predicting success as SSD whole traffic predict, and comparing with existing time series of traffic. In the figure, the blue line represents the actual traffic of SSD, the orange line represents the predicted traffic of SSD, and the red line is the division line of different SSDs. The figure shows that our prediction results are close to the actual traffic. Accurate prediction of future time series can be realized through periodicity, and the prediction results can be used for PreciseScheduling.

5.2 PreciseScheduling performance

Our experiment uses Alibaba real datasets to simulate the scheduling effect on the MQsim simulator.

5.2.1 Experimental setup

Simulation. To evaluate the scheduling effect, we use MQsim (Tavakkol et al. 2018), an open-source MQ-SSD simulator, which can accurately simulate modern SSD based on NVME. Table 3 shows the configuration of the MQ-SSD system we modeled.

Dataset. We use an actual cloud computing load. The dataset contains more than 40000 different segments and is stored in 96 separate SSDs. Overall, the average value of the total read-write traffic is 2.5GB/s, and the peak value of the total read-write traffic is 6.5GB/s. In our simulated SSD environment, the average read-write latency of SSDs is 127us, and the average 99.99% read-write tail latency of SSDs is 1.019ms. However, for a single SSD, the 99.99%

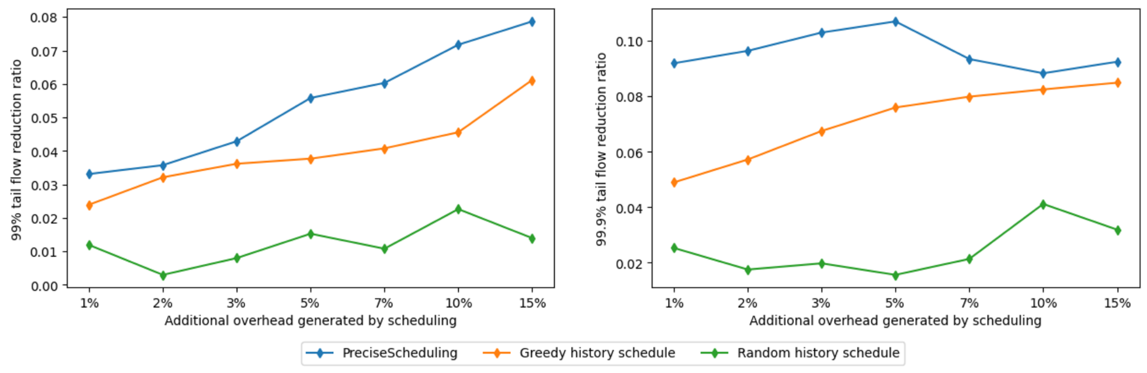

Fig. 12 Tail flow reduction of different scheduling algorithms

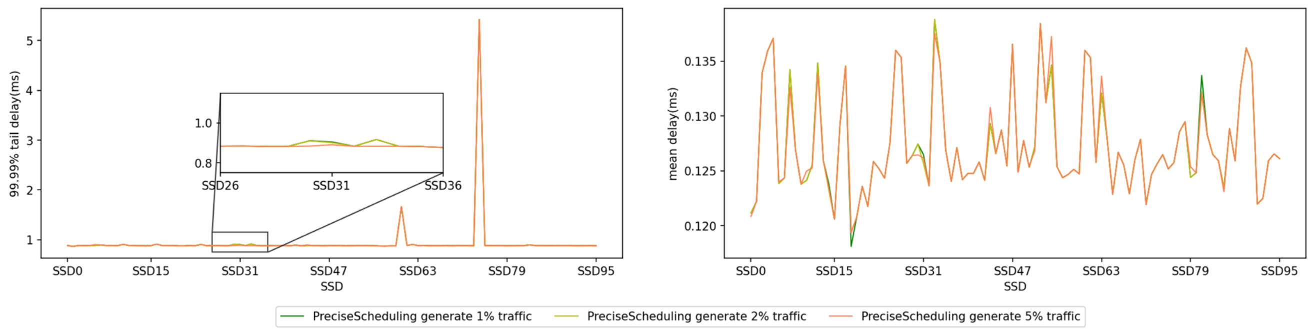

Fig. 13 PreciseScheduling effect under various scheduling overhead. The left plot shows the 99.99% tail latency. The right plot shows the average latency

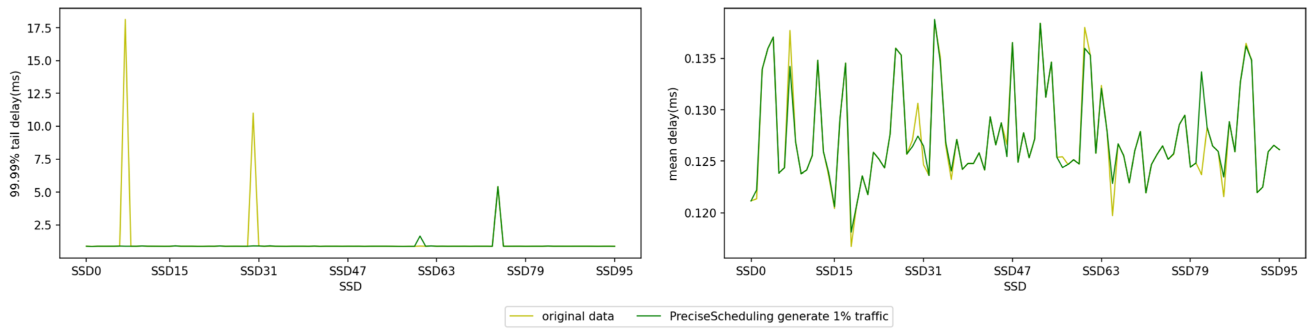

Fig. 14 PreciseScheduling effect. The left plot shows PreciseScheduling lowers the tail latency. The right plot shows it doesn't increase the average latency

read-write tail latency may be up to ten milliseconds. Our proposed PreciseScheduling algorithm can effectively reduce the peak of tail latency and ensure that all SSDs do not produce too significant tail latency.

5.2.2 Comparison with existing algorithms

We first spread all segments into different SSDs by hashing and then reduce the tail latency by scheduling. We compare the proposed PreciseScheduling algorithm with the other two methods: (1) Greedy scheduling based on historical information, similar to our idea. The difference is that this method selects the scheduled segments from all segments according

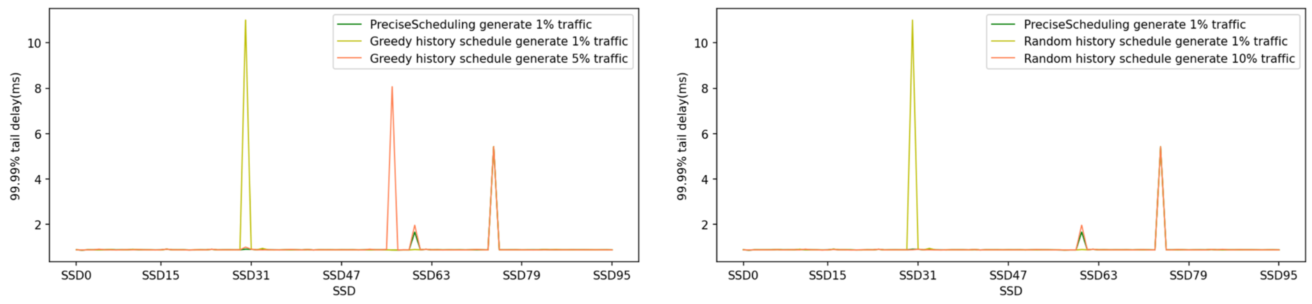

Fig. 15 The comparison of PreciseScheduling with two different scheduling algorithms

to the historical data, but our process only considers the predicted successful segments; (2) Random scheduling based on historical information, similar to greedy scheduling based on historical information. The difference is that this method randomly selects the scheduling target SSD, and greedy scheduling selects the SSD that can minimize the peak traffic.

Firstly, we compare the decline degree of the tail flow of different scheduling methods under various scheduling times. As shown in Fig. 12, PreciseScheduling consistently achieves the best effect. It can be seen from the figure on the left, as the number of scheduling times increases, the additional overhead caused by scheduling increases, and the degree of 99% tail flow decline is also increasing. As can be observed from the figure on the right, PreciseScheduling algorithm performed well in the situation of a few scheduling times for 99.9% tail flow. Although it can further reduce the 99.9% tail flow by increasing the scheduling times, the degree of decline is insignificant. The most severe impact on tail latency is usually the peak traffic, so PreciseScheduling can achieve good results with little scheduling overhead. The other two scheduling algorithms need more scheduling overhead to reduce the peak traffic and reduce the tail latency effectively.

Next, we experiment with the scheduling results in the simulated SSD environment, simulate the scheduling for 6 hours, and observe the change of tail latency. The dataset we use contains more than 40000 segments, and we move 32GB of data for each schedule. According to the calculation results of data traffic, generating an additional 1% scheduling overhead means scheduling 17 segments every 6 hours.

Firstly, we test the effect of PreciseScheduling under different scheduling overhead, as shown in Fig. 13. The left figure shows 99.99% tail latency under various scheduling overheads, and the right figure represents the average latency. There is still a tail latency peak because the traffic of a single segment is too high, which can not be solved by scheduling. The effect is the same as that in Fig. 12. After increasing the scheduling overhead, the change degree of 99.9% tail flow is small, and there is no discernible variation in 99.99% tail delay. Therefore, we use 1% extra scheduling overhead to compare with other methods.

Secondly, we analyze the effect of PreciseScheduling, as shown in Fig. 14. The left figure shows the 99.99% tail delay before and after each SSD scheduling. The scheduling algorithm proposed reduces the tail delay when generating an additional 1% scheduling overhead. PreciseScheduling solves the problem of the high tail latency of a single SSD. We schedule the load to different SSDs to avoid the tail latency caused by load superposition. The figure on the right shows that PreciseScheduling will not increase the average latency while reducing the tail latency.

Finally, we compare PreciseScheh the other scduling algorithm witheduling methods, as shown in Fig. 15. The left figure shows PreciseScheduling and greedy scheduling based on historical information. When the scheduling times are the same, greedy scheduling based on historical information can not well reduce the tail latency of all SSDs, and there is still a high tail latency in one SSD. Even if it generates five times the scheduling overhead of PreciseScheduling, the scheduling effect is still not good. The figure on the right shows PreciseScheduling and random scheduling based on historical information. The effect of random scheduling is worse than PreciseScheduling. Only when scheduling generates an additional 10% scheduling overhead can it achieve similar results to PreciseScheduling.

We conduct experiments with PreciseScheduling on more extensive datasets. The scheduling effect varies according to the different of data. In aggregate, our algorithm can considerably reduce the tail latency by generating an additional $1-2\%$ of the scheduling overhead while the SSD with the highest 99.99% tail latency in the storage node can minimize tail latency by $50-90\%$.

6 Conclusion

We proposes PrecisePeriod in this paper, a precise periodicity detection algorithm. It achieves precise periodicity detection by four steps, removes outliers through data preprocessing, employs discrete wavelet transform to remove high-frequency noise, computes candidate periodicity queue by autocorrelation function, and determines precise period through periodicity verification. Experiment on actual and synthetic datasets shows that PrecisePeriod algorithm is better than several state-of-art periodicity detection methods. We found that 30% of the time series in real-world datasets from Alibaba are periodic through PrecisePeriod algorithm. In the end, we introduce a cloud storage load balancing scheduling method based on PrecisePeriod, and the evaluation reveals that PrecisePeriod scheduling significantly reduces tail latency while incurring just $1-2\%$ overhead.

Acknowledgements We thank Alibaba for the data provided for this paper experiment and the reviewers for their valuable comments on this article. This work is supported by The National Key Research and Development Program of China (2019YFB1804502), NSFC: 61872392, NSFC: 61832020, the Major Program of Guangdong Basic and Applied Research: 2019B030302002, the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant No. 2016ZT06D211, Guangdong Natural Science Foundation (2018B030312002).

Declarations

Conflict of interest The authors declare no conflict of interest, financial or otherwise. On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

Almasri, A.: A new approach for testing periodicity. Commun. Statist. **40**(7), 1196–1217 (2011) https://doi.org/10.1080/03610920903564743

Berthold, MR., Höppner, F.: On clustering time series using euclidean distance and pearson correlation. In: arXiv preprint arXiv:1601.02213 (2016)

Box, G.E.P.: Time series analysis: forecasting and control, John Wiley & Sons, Hoboken (2015)

Brockwell, P.J., Davis, R.A. (eds.): Introduction to time series and forecasting. Springer, New York (2002)

Cai, C., Harrington, P.d.B.: Different discrete wavelet transforms applied to denoising analytical data. J. Chem. Inform. Model. **38**(6), 1161–1170. 10.1021/ci980210j (1998)

Cooper, RB.: Queueing theory. In: Pro-ceedings of the ACM '81 Conference. New York, NY, USA: Association for Computing Machinery, 119–122. 10.1145/800175 (1981)

Daubechies, I.: Ten lectures on wavelets. SIAM (1992)

Didona, D., Zwaenepoel, W.: Sizeaware sharding for improving tail latencies in inmemory key-value stores. In: 16th fUSENIXg Symposium on Networked Systems Design and Implementation (fNSDIg 19), pp. 79–94 (2019)

Elfeky, M.G., Aref, W.G., Elmagarmid, A.K.: Periodicity detection in time series databases. IEEE Transact.Knowledge Data Eng. **17**(7), 875–887 (2005)

Eltabakh, M.Y., et al.: CoHadoop: exible data placement and its exploitation in Hadoop. In: Proceedings of the VLDB Endowment **4**(9), 575–585 (2011)

Elyasi, N., et al.: (2017). Exploiting intra-request slack to improve SSD performance. In: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 375–388

Hyndman, R.J., Athanasopoulos, G.: Forecasting: principles and practice. OTexts (2018)

Kim, J., et al.: Alleviating garbage collection interference through spatial separation in all ash arrays. In: 2019 fUSENIXg Annual Technical Conference (fUSENIXg fATCg 19), pp. 799-812 (2019)

Kumar, M., Patel, N.R., Woo, J.: Clustering seasonality patterns in the presence of errors. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 557–563 (2002)

Lange, H., Brunton, S.L., Kutz, J.N.: From Fourier to Koopman: spectral methods for long-term time series prediction. J. Mach. Learn. Res. **22**(41), 1–38 (2021)

Lee, L-W., Scheuermann, P., Vingralek, R.: File assignment in parallel I/O systems with minimal variance of service time. IEEE Transact. Comput. **49**(2), 127–140 (2000)

Madathil, D.K., et al.: A static data placement strategy towards perfect load-balancing for distributed storage clusters. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, pp. 1–8 (2008)

Mezic, I., Surana, A.: Koopman mode decomposition for periodic/quasi-periodic time dependence. IFAC-PapersOnLine **49**(18), 690–697 (2016)

Mitsa, T.: Temporal data mining. CRC Press, Boca Raton (2010)

Percival, D.B., Walden, A.T.: Wavelet methods for time series analysis. Vol. 4. Cambridge university press, Cambridge (2000)

Rasheed, F., Alhajj, R.: A framework for periodic outlier pattern detection in time-series sequences. IEEE Transact. Cybern. **44**(5), 569–582 (2013)

Sellami, M., et al.: Clustering-based data placement in cloud computing: a predictive approach. Cluster Comput., pp. 1–26. https://doi.org/10.1007/s10586-021-03332-1(2021)

Shi, Y., et al.: AdaptScale: an adaptive data scaling controller for improving the multiple performance requirements in Clouds. Future Gener. Comput. Syst. **105**, 814–823 (2020). https://doi.org/10.1016/j.future.2017.08.034

Skarlatos, D., Kim, N.S., Torrellas, J.: Pageforge: a near-memory content-aware page-merging architecture. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-50 '17. Association for Computing Machinery, 302-314. https://doi.org/10.1145/3123939.3124540(2017)

Sriraman, A., Dhanotia, A., Wenisch, T.F.: SoftSKU: optimizing server architectures for microservice diversity @scale. In: Proceedings of the 46th International Symposium on Computer Architecture. ISCA '19. New York, NY, USA: Association for Computing Machinery, 513–526. isbn: 9781450366694. https://doi.org/10.1145/3307650.3322227(2019)

Tavakkol, A., et al.: Mqsim: a framework for enabling realistic studies of modern multi-queue SSD devices. In: 16th fUSENIXg Conference on File and Storage Technologies (fFASTg 18), pp. 49–66. urlhttps://www.usenix.org/conference/fast18/presentation/tavakkol (2018)

Theodosiou, M.: Forecasting monthly and quarterly time series using STL decomposition. Int J. Forecast. **27**(4), 1178–1195 (2011)

Tian, C.J.: A Limiting property of sample autocovariances of periodically correlated processes with application to period determination. J. Time Series Anal. **9**(4), 411–417. https://doi.org/10.1111/j.1467-9892.1988.tb00480.x(1988)

Toller, M., Kern, R.: Robust parameter-free season length detection in time series. In: arXiv preprint arXiv:1911.06015 (2019)

Toller, M., Santos, T., Kern R.: SAZED: parameter-free domain-agnostic season length estimation in time series data. Data Mining Knowledge Discovery **33**(6), 1775–1798 (2019)

Tukey, J.W., et al.: Exploratory data analysis, vol. 2. Reading Mass (1977)

Vengadeswaran, S., Balasundaram, S.R.: Clust: grouping aware data placement for improving the performance of large-scale data management system. In: Proceedings of the 7th ACM IKDD CoDS and 25th COMAD, pp. 1–9 (2020)

Vlachos, M., Yu, P., Castelli, V.: On periodicity detection and structural periodic similarity. In: Proceedings of the 2005 SIAM international conference on data mining. SIAM, pp. 449–460 (2005)

Vlachos, M., et al.: Identifying similarities, periodicities and bursts for online search queries. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 131–142 (2004)

Wang, J., Chen, T., Huang, B.: Cyclo-period estimation for discrete-time cyclo-stationary signals. IEEE Transact. Signal Proces. **54**(1), 83–94. https://doi.org/10.1109/TSP.2005.859237(2006)

Wang, J., Shang, P., Yin, J.: Draw: a new data-grouping-aware data placement scheme for data intensive applications with interest locality. Cloud Comput. Data-Intensive Appl. Springer, pp. 149–174 (2014)

Yan, S., et al.: Tiny-tail ash: near-perfect elimination of garbage collection tail latencies in NAND SSDs. ACM Transact. Storage (TOS) **13**(3), 1–26 (2017)

Yuxiao Han is currently working toward an M.Sc. degree in the School of Computer Science and Engineering from Sun Yat-sen University, China. His main research interests focus on resource scheduling in cloud storage.

Jia Ma is currently working toward the M.Eng. degree in the School of Computer Science and Engineering from Sun Yat-sen University, Guangzhou. His current research areas include resource management in cloud storage.

Fei Li graduated from the State Key Laboratory of parallel and distributed computing at the University of national defense science and technology with a computer science and technology doctor. He is a technical expert in Alibaba cloud intelligent storage department, a distinguished researcher of Shenzhen University, a member of CCF YOCSEF of China computer society, a member of CSIG visual perception special committee of China image graphics society, communication member of China artificial intelligence society and CAAI machine learning professional committee, engaged in research in the intelligent application field.

Yubo Liu received his Ph.D. degree in computer science and technology from Sun Yat-sen University. His research direction is storage system and high performance computing, including but not limited to distributed/local file system, object storage system, persistent memory, and consistency.

Nong Xiao received his B.S. and Ph.D. degrees in computer science and technology from National University of Defense Technology, Changsha. He is a professor at Sun Yat-sen University, Guangzhou, and the deputy director of National Supercomputer Center in Guangzhou. His current research interest includes cloud computing and cloud storage systems, memory computing, high-performance big data computing systems, parallel algorithms for big data analysis and processing, large-scale solid-state storage, and new data-intensive computing systems.

Yutong Lu received her B.S., M.S., and Ph.D. degrees in computer science and technology from National University of Defense Technology, Changsha. She is a professor at Sun Yat-sen University, Guangzhou, and the director of National Supercomputer Center in Guangzhou. Her current research interest includes large-scale storage system, high-performance computing, and computer architecture.

Zhiguang Chen received his B.S. degree from Harbin Institute of Technology, Harbin, and his M.S. and Ph.D. degrees in computer science and technology from National University of Defense Technology, Changsha. He is an associate professor at Sun Yat-sen University, Guangzhou. His current research interest includes distributed file system, network storage, and solid-state storage system.