Efficient Time Series Data Classification and Compression in Distributed Monitoring*

Sheng Di[1], Hai Jin[1], Shengli Li[1], Jing Tie[2], and Ling Chen[1]

[1] Cluster and Grid Computing Lab
Services Computing Technology and System Lab
Huazhong University of Science and Technology, Wuhan, 430074, China
[2] Department of Computer Science, University of Chicago
1100 E 58th Street Chicago, USA
hjin@hust.edu.cn

Abstract. As a key issue in distributed monitoring, time series data are a series of values collected in terms of sequential time stamps. Requesting them is one of the most frequent requests in a distributed monitoring system. However, the large scale of these data users request may not only cause heavy loads to the clients, but also cost long transmission time. In order to solve the problem, we design an efficient two-step method: first classify various sets of time series according to their sizes, and then compress the time series with relatively large size by appropriate compression algorithms. This two-step approach is able to reduce the users' response time after requesting the monitoring data, and the compression effects of the algorithms designed are satisfactory.

1 Introduction

Efficiently storing monitoring data is a crucial function to a distributed monitoring system, however, many current systems [1, 2, 3] have not considered it well enough, especially on how to classify several sets of time series and compress them appropriately. In practice, it is very important to design a way of reducing historical monitoring data before transmitting them in large-scale distributed systems. As a matter of fact, if there is not a processing module, the size of the transmission data may be so large that clients' memories can not afford such high costs (Some exceptions may be thrown out, such as "OutOfMemoryException" [4]), or the corresponding transmission time will be so long that users can not endure.

There are several candidate ways to solve the problem mentioned above. The first way is that we can make a buffer on the client device that retrieves historical monitoring data. The size of this buffer can be set as an appropriate value to avoid being out of memory. As soon as the buffer is full, the monitoring data retrieved will be displayed on GUI (*Graphic User Interface*) or stored into external storage. This way can certainly avoid being out of memory, but it unwillingly enhances the loads of client devices and cannot reduce transmission time. Moreover, it is against to the

* This paper is supported by National Science Foundation of China under grant 90412010, ChinaGrid project from Ministry of Education, and CNGI projects under grant CNGI-04-15-7A.

principle of designing a thin client instead of a fat one in the grid field [5]. The other way is to compress historical monitoring data before transmitting them. It can reduce not only the loads of clients but also the transmission time.

From clients' views, the historical monitoring data are displayed as several sets of time series to be compared. In our monitoring system, as a use case, each set of time series stands for the performance on some metric of one node inside a cluster. Thus, the compression issue could be transformed to how to compress time series. Moreover, it is apparent that as long as we choose appropriate compression algorithms, the fluctuation of time series compressed can still be displayed clearly on clients' computers.

This paper mainly studies a strategy in processing time series data. Not only do we develop a kind of classification algorithm to organize all the time series to be compared (such as, the monitoring data about all the nodes inside one cluster with some metrics), but we also study some algorithms used to compress a specific set of time series (i.e. to reduce dimensionality of time series).

In our practical distributed monitoring platform, CGSV [15], these approaches mentioned above are processed in a server called Archive [6] which mainly provides two services. One service is used to receive requests from *Target System Service* [15] (Target System Service is a web service deployed on the front node of a cluster and used to notify monitoring data to Archive) and execute corresponding operations (including "create table", "insert data", etc.), and the other one is used to publish interfaces for clients.

The rest of this paper is organized as follows: in section 2, we present the related work. In section 3, we introduce our classification algorithm (i.e. the preparation task of compressing). We give some detailed compression algorithms and relative analyses in section 4. We analyze the results of our practical testing in section 5 and conclude and present the future work in section 6.

2 Related Work

2.1 Storage Modules in Grid Monitoring System

The field on distributed monitoring system has a long history in literatures, but the issue about how to effectively process monitoring data is still underway.

- VisPerf [1]: VisPerf is a monitoring tool for grid computing. There is no storage module in it.
- MonALISA [2]: MonALISA has a special storage module used to compress monitoring data. As data are becoming older over time, the values stored will be compressed by evaluating the mean values on large time intervals. In our paper, we provide several better compression algorithms than this mean value algorithm.
- GridView [3]: GridView is a dynamic and visual grid monitoring system. It has a Grid Information Manager used to compress monitoring data periodically. Compared with it, our approach does not rewrite the original data so as to keep them lossless. The Archive module compresses the corresponding time series just as clients request them.

2.2 Compression Algorithms

In recent years, there has been an explosion of interests in mining time series databases. Several high level dimensionality reductions of time series have been proposed, including *Singular Value Decomposition* (SVD) [7], *Discrete Fourier Transform* (DFT) [8], *Discrete Wavelets Transform* (DWT) [9], *Symbolic Mappings* [10, 11, 12], *Piecewise Linear Representation* (PLR) [13], and *Piecewise Aggregate Approximation* (PAA) [14]. As a matter of fact, SVD, DFT and DWT are mainly designed to match patterns, so they are relatively intricate to be implemented to compress data. In this paper, we adopt the clearest approach, *Piecewise Linear Representation* (PLR), to compress the monitoring data. The goal of this approach is to index some special points in a time series and link them in a series. The most straightforward index algorithms used in PLR are *Random Walk Algorithm* (RWA) and *Mean Value Algorithm* (MVA). However, neither are they effective when encountering many white noises [16] (i.e. where each value x_t is completely independent to its neighbors x_{t-1}, x_{t+1}). In addition, although *Height Sorting Reduction* (HSR) algorithm and *Angle Sorting Reduction* (ASR) algorithm can adapt to white noises well, neither of them is suitable to the other cases without white noises. Facing this issue, we design two new algorithms (PFR and A-ISR) which are able to effectively index special points of time series in both cases (with white noises and without white noises). Finally, we test these two algorithms and prove that PLR can get a very high efficiency in compression.

3 Classification Algorithm of Historical Monitoring Data

In our monitoring platform (CGSV) [6], all the monitoring data are stored into the database deployed on the Archive. In order to reduce the complexity and keep database legible, all the time series data are saved together. We provide a use case: the monitoring performance data about all the nodes inside one cluster. In this case, Archive has to classify these time series out from the original data before compressing them.

3.1 Classification Algorithm

We assume the original data have been organized as *n* sets of time series shown in Fig.1. *S* is defined as the total remaining free space (i.e. the maximum number of monitoring data). With monitoring data being indexed over time, this free space will keep shrinking. The initial space is assumed to be S_0 (such as $S=S_0=10000$), and it means confining the maximum number of data transmitted at one time to *10000*. The main purposes of the classification algorithm are to justify which time series should be compressed and calculate the appropriate compression ratios for them.

It is obvious that $\overline{s}=S/n$ is the mean length of all the target time series. We define a rule as follows: If the length of some original time series is less than or equal to $\overline{s}$, it will not be compressed. Hence, these data will be indexed immediately. We call this kind of time series *Raw-Index Time Series*, while we call the time series which should be compressed *Compress-Index Time Series*. Apparently, if the number of *Raw-Index Time Series* is *m*, then the number of *Compress-Index Time Series* will be (*n-m*) and the remaining data space is: $S-\sum_m$(the size of the *Raw-Index Time Series*). After executing Formula (3) and Formula (4), one search round will be done, and then, the work flow will be recursively returned back to Formula (1). In each round, $\overline{S}$ may be greater than that of last round, so the length of a time series whose length was greater than that of $\overline{S}$ in previous rounds may become shorter than that of $\overline{S}$ in the current or future rounds.

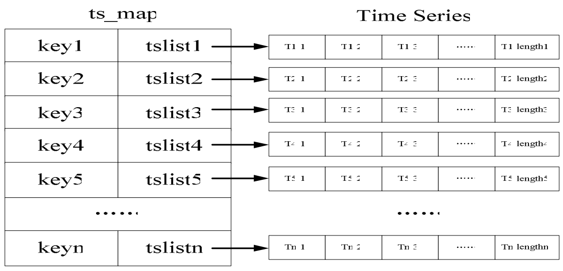


Fig. 1. Time Series of Ts_map

$$\overline{S} = \frac{S}{n} \tag{1}$$

$$l\begin{cases} \le \overline{S} & \text{Raw-Index Time Series} \\ > \overline{S} & \text{Compress-Index Time Series} \end{cases} \tag{2}$$

$$S \leftarrow S - \sum_m (\text{the size of Raw-Index Time Series}) \tag{3}$$

$$n \leftarrow (n - m) \tag{4}$$

3.2 Pseudo-code of Classification Algorithm

All the variables used in the classification algorithm are listed in Table 1. The pseudo-code of classification algorithm is shown in Fig.2.

4 Compression Algorithms

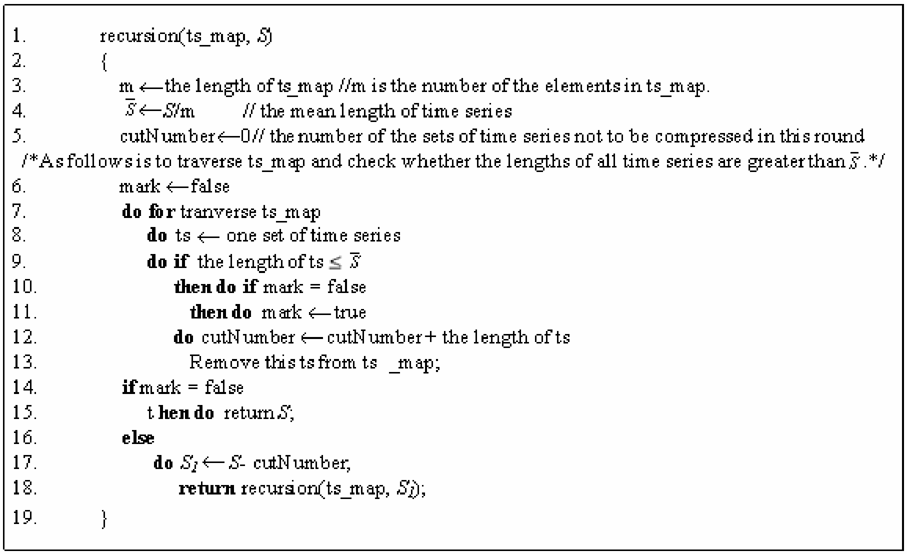
4.1 Characteristics of Monitoring Data

Monitoring data own a number of characteristics. We study them carefully in our distributed monitoring platform: CGSV (*ChinaGrid SuperVision*) [6, 15]. In this system, all the sensors deployed on monitored nodes are used to retrieve monitoring data every β seconds (β can be modified by users). In other words, as soon as the value of a metric is changed over a revisable threshold (its default value is 0.1 in CGSV), the node's sensor will receive a corresponding notification and forward it upwards. The characteristics of monitoring data are as follows:

- Small amplitude fluctuations may inevitably exist in high frequencies.
- It is possible to generate abrupt fluctuations at some time stamps in a time series.
- The randomicity of monitoring time series data is very high.
- There are huge discrepancies among the monitoring data belonging to different metrics.

Table 1. Variables in the Pseudo-code of Classification Algorithm

Variable	Explanation
ts_map	A map that consists of time series (such as Fig.1)
ts	One element in *ts_map* (i.e. a set of time series)
cutNumber	The number of the sets of time series not to be compressed in this round
mark	It is used to mark if there is at least one set of time series which need not be compressed in a round



```
recursion(ts_map, $S$)
{
  m ← the length of ts_map //m is the number of the elements in ts_map.
  $\bar{S}$ ← $S$/m      // the mean length of time series
  cutNumber ← 0// the number of the sets of time series not to be compressed in this round
/*As follows is to traverse ts_map and check whether the lengths of all time series are greater than $\bar{S}$.*/
  mark ← false
  do for tranverse ts_map
     do ts ← one set of time series
     do if the length of ts ≤ $\bar{S}$
        then do if mark = false
           then do mark ← true
        do cutNumber ← cutNumber + the length of ts
           Remove this ts from ts _map;
  if mark = false
     then do return $S$;
  else
     do $S_1$ ← $S$- cutNumber;
        return recursion(ts_map, $S_1$);
}
```

Fig. 2. The Pseudo-code of Classification Algorithm

The area magnified in Fig.3 shows the small amplitude fluctuations. In practice, when the metric is set to be CPU_Idle, most of monitoring data may often fluctuate from 99.8 to 100.0. Fig.3 also shows the abrupt fluctuations and the randomicity of monitoring data. Fig 4 shows the monitoring data of free memory. Plus the different units of diverse metrics or other changeful cases, the fluctuations of different time series may be extremely disparate.

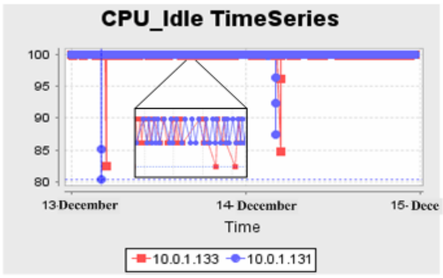


Fig. 3. Small Amplitude Fluctuation and Abrupt Fluctuations at Time Stamp

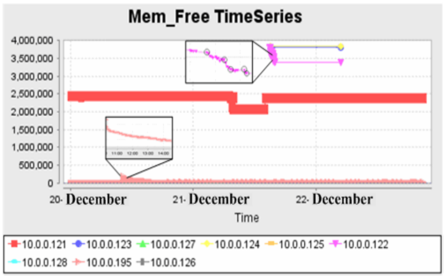


Fig. 4. Monitoring Data of Some Free Memory

There have been many existing outstanding solutions to compress large-scale time series data [7, 8, 9, 10, 11, 12, 14]. However, all of them are relatively complicated to implement. For instance, as for DFT [8], programmers have to implement the intricate DFT algorithm, choose appropriate coefficients, and implement the R*-Tree algorithm [17].

In order to adapt to the features of our monitoring data mentioned above and also facilitate the implementation, we adopt PLR [13] and design a couple of new algorithms to index special points.

4.2 Time Series Index Algorithms in PLR

In order to facilitate the implementation of compression, we adopt the PLR method and put forward two new index algorithms in it. Let a set of values as $\{x_0, x_1, \ldots, x_n\}$ and the corresponding time stamps as $\{t_0, t_1, \ldots, t_n\}$. We assume the time series with n original points will be extracted into a sequence with $\overline{s}$ ($\overline{s}$ is the $\overline{s}$ after executing classification algorithm) special points and the points (t_0, x_0) and (t_n, x_n) are always being indexed.

Peak Focus Reduction (PFR)

The focus of this algorithm is all the peaks of a time series, shown in Fig.5. First, sort the points in the given time series according to d_i calculated in Formula (5) by descend.

$$d_i = x_i - \overline{v}, \overline{v} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5}$$

And then, pick up these points in the time series by descend one by one. As for each point selected, estimate its neighbor points in time series toward two sides (left and right). Once the warp between its value and that of its neighbor points surpasses a threshold, this neighbor point will be indexed and this estimation section will be ended. Otherwise, estimate its neighbor's neighbor point analogically. The whole algorithm will not be done until the number of points indexed is saturated (i.e. up to $\overline{s}$ points).

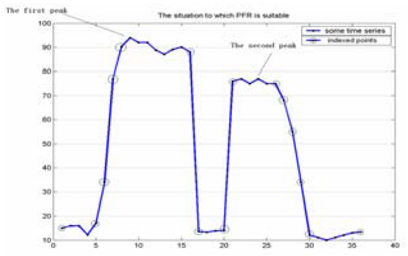


Fig. 5. The Special Points Indexed with PFR Algorithm

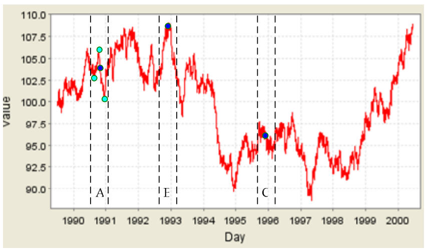


Fig. 6. The Special Points Indexed with A-ISR Algorithm

Adaptable Interzone Sampling Reduction (A-ISR)

Define *K* as an adaptable coefficient determining the value of the threshold *TH* and we take *K=10* in our practical test. In Formula (6), the variables *max* and *min* denote the maximal value and the minimal value, respectively, during the whole time series.

$$TH = \frac{1}{K}\,|\max - \min| \tag{6}$$

As long as Condition (7) is met, the corresponding fluctuation will be viewed as distinct. In contrast, if it is not met, the fluctuation will be ignored (i.e. we treat this tiny fluctuation as no fluctuation).

$$\Delta x > TH, \qquad \Delta x = |x_{i+1} - x_i| \tag{7}$$

We use a variable *p* to mark the current trend of the segment of time series.

$$p = \begin{cases} 1 & \text{The current value is larger than the last one} \\ 0 & \text{The two values' discrepance should be ignored} \\ \text{-1} & \text{The current value is smaller than the last one} \end{cases} \tag{8}$$

We use n_{turn} to record the number of the turning points. By the end of the algorithm, n_{turn} will be checked. n_{turn}=0 means there is no turning point during the specific period of this time series. In this case, we index a random point. n_{turn}=1 means there exists one and only one extreme point during the specific period, and this special extreme point will be indexed. $n_{turn}\geq 2$ means there are more than one turning points during this specific period, and in this case, we will calculate the mean time stamp of all the special extreme points in this period as the time stamp of the special point and the mean value. Fig.6 shows the points indexed with this algorithm when processing a time series. In section A, n_{turn}=3, so the calculated blue point will be indexed. In section B, n_{turn}=1, so the exclusive special point will be indexed. In section C, n_{turn}=0, therefore we index a random point.

4.3 Analysis of Algorithms

As to the two index algorithms mentioned above, they have their own features and their time complexities are acceptable.

- *Peak Focus Reduction* (PFR): PFR is suitable to the cases with and without white noises. This algorithm is a little similar to HSR (it will be mentioned in section 5.2), but it is more reasonable than HSR. Its time complexity is $O(n\log n)$ because the sort algorithm we use is Heapsort [18].
- *Adaptable Interzone Sampling Reduction* (A-ISR): This algorithm does not sort time series but just scans through the set of time series. Hence, the time complexity is not $O(n\log n)$ but $O(n)$.

5 Performance Analysis

5.1 Performance of Classification Algorithm

The whole time consumed is immensely reduced because of the Classification Algorithm. In our practical testing, when the total number of raw data is *12000*, the time consumed by Classification Algorithm is still less than 1ms. However, the transmission time is much longer than it. So the time cost by this algorithm can be ignored. Fig.7 shows the difference of the response time consumed between with and without our approaches. From this figure, we could find that when users request *12000* data, the total time consumed is mainly in transmitting data. However, when we compress them into *600* data with our algorithms before transmission, the whole response time consumed will be cut down greatly.

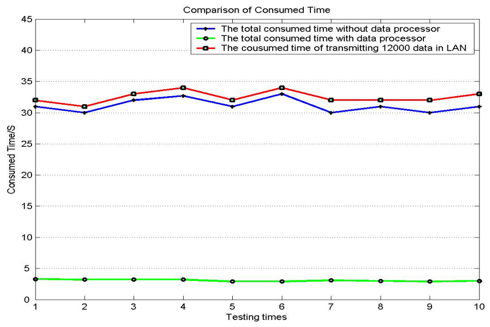
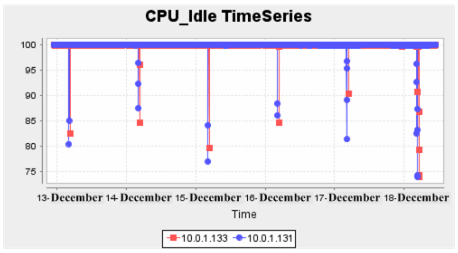


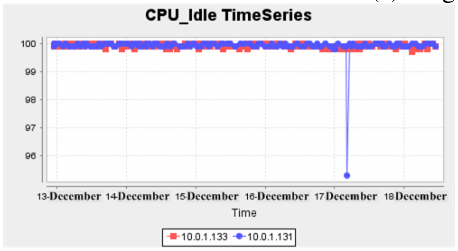
Fig. 7. Comparison of Consumed Time between with and without Our Approaches

5.2 Performance and Effect of Index Algorithm in PLR

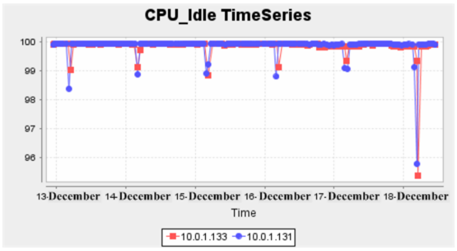
We test all the index algorithms mentioned above with two groups of original monitoring data: one group with white noises and the other without them. In addition, we compare our index algorithms with traditional ones: *Random Walk Algorithm*, *Mean Value Algorithm*, *Height Sorting Reduction* (HSR), and *Angle Sorting Reduction* (ASR). *Random Walk Algorithm* and *Mean Value Algorithm* choose random point and mean value point to replace the values in one segment, respectively. HSR sorts all monitoring data in terms of their values' discrepancies to mean value and indexes the $\overline{s}$ biggest ones. ASR is similar to HSR, and their exclusive difference is ASR's criterion of sorting data is angles of time series lines.



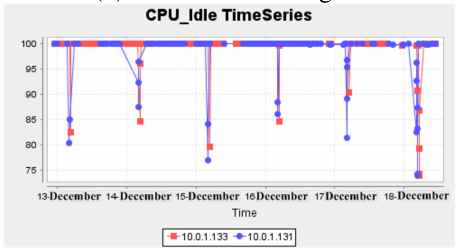
(a) Original Data



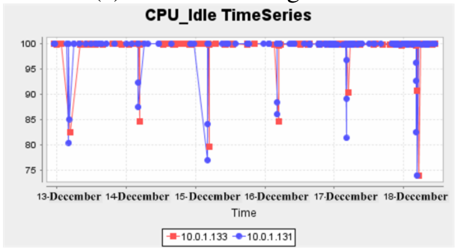
(b) Random Walk Algorithm



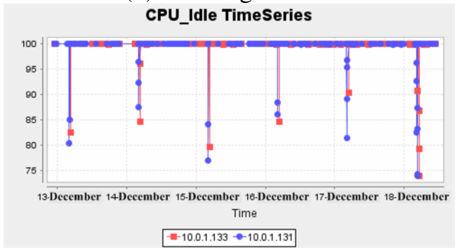
(c)Mean Value Algorithm



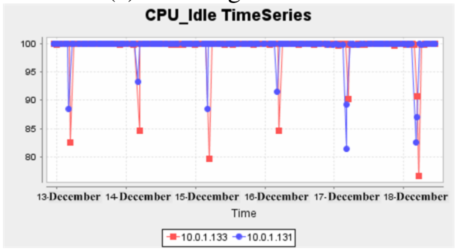
(d)HSR Algorithm



(e) ASR Algorithm



(f) PFR Algorithm



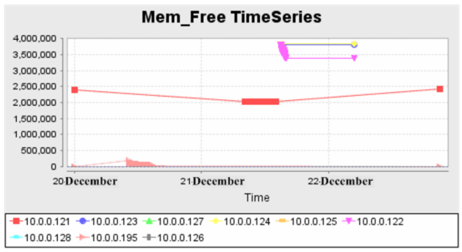
(g) A-ISR Algorithm

Fig. 8. Testing Effects of All the Compression Algorithms in the Situation with White Noises

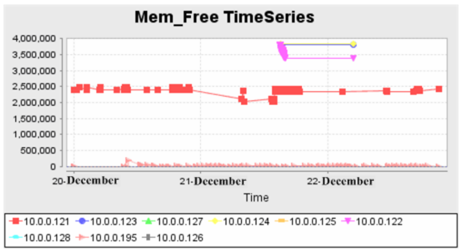
Fig 8 (a) presents the original time series and its size is *6076*. The result size of time series processed in each figure is *300*. So each compression ratio is up to (6076-300)/6076*100%=95%.

The other sub-figures in Fig.8 show the different compression effects of the various algorithms when it comes to many white noises in the original time series. Fig.8(b) and Fig.8(c) show that the fluctuations generated by *Random Walk Algorithm* or *Mean Value Algorithm* are highly different from the original time series. Fig.8(d), Fig.8(e), Fig.8(f), and Fig.8(g) indicate that ASR algorithm and A-ISR algorithm generate better compression effects and HSR algorithm and PFR algorithm come into being the closest fluctuations to the original time series.

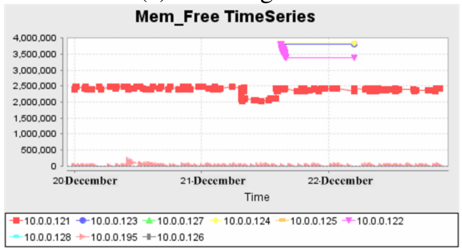
Fig 9 shows the compression effects with the group of data without many white noises (the corresponding original time series is shown in Fig.4). They reveal that in this case, PFR algorithm and A-ISR algorithm still keep the closest fluctuations to the original time series.



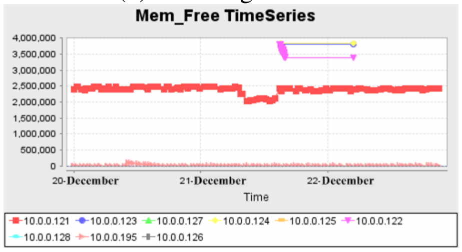
(a) HSR Algorithm



(b) ASR Algorithm



(c) PFR Algorithm



(d) A-ISR Algorithm

Fig. 9. Testing Effects of All Compression Algorithms in the Situation without White Noises

As a result, our two new index algorithms, PFR and A-ISR, indeed own values in compressing time series.

Fig.10 presents the relative time consumed. In this figure, it is easy to find that A-ISR, RWA, and MVA algorithms cost least time. Although the other three algorithms cost relatively more time, the time increases in terms of $O(n\log n)$. Apparently, these costs still adapt to users' commands.

From the testing results shown in Fig.8, Fig.9, and Fig.10, we can summarize as follows:

In the white noise situations, PFR algorithm and HSR algorithm have the best compression effects, and *Random Walk Algorithm*'s compression effect is the worst. In the situation without white noises, PFR algorithm and A-ISR algorithm generate better compression effects than the others. In addition, *Random Walk Algorithm*, *Mean Value Algorithm* and A-ISR algorithm consume the shortest time because their time complexities are all $O(n)$. In fact, the time consumed by each algorithm mentioned above could be ignored compared with the time cost via network. So, we recommend PFR algorithm and A-ISR algorithm in most situations.

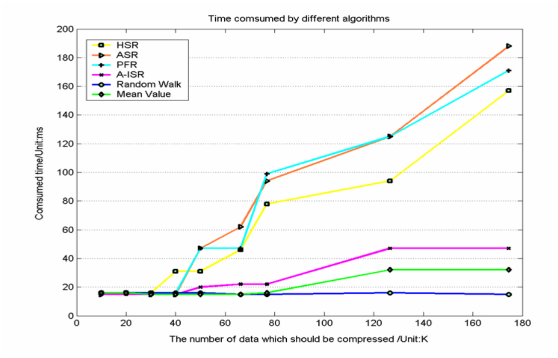


Fig. 10. Time Consumed by Different Algorithms

6 Conclusion and Future Work

In order to solve the problem of transmitting massive data in the distributed monitoring field, we studied how to classify and compress time series to be compared. The classification algorithm is a recursive algorithm that can make full use of memory space of clients' devices. This algorithm is also a key step of preparing for the following time series compression. As to the time series compression method, we adopted *Piecewise Linear Representation* (PLR) method to process data. Based on this method, we designed two kinds of fresh index algorithms and analyzed their performance. The testing results present that both of them are suitable to the cases with and without white noises and consume acceptable time.

In the future, we still have some things to study, such as:

- We are supposed to make progress on index algorithms, especially on compression effect of A-ISR algorithm. In addition, we should try to exploit other better ones.
- Different index algorithms are adapted to different kinds of time series. So, we will enhance the adaptability of archive in CGSV by adopting various algorithms to process diverse kinds of time series.

References

1. Lee, D., Dongarra, J., Ramakrishna, R.: Visperf: Monitoring Tool for Grid Computing. In: Proceeding of ICCS, pp. 1–12 (2003)
2. Newman, H.B., Legrand, I.C., Galvez, P., Voicu, R., Cirstoiu, C.: MonALISA: A Distributed Monitoring Service Architecture. In: Proceedings of CHEP, La Jolla, CA, pp. 1–8 (2003)
3. Guangbo, N., Jie, M., Bo, L.: GridView: A dynamic and visual grid monitoring system. In: Proceedings of HPC Asia, pp. 89–92 (2004)
4. OutOfMemoryException and other pathological cases, http://haacked.com/archive/2004/02/11/189.aspx
5. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications 15(3) (2001)

6. Zheng, W., Liu, L., Hu, M., Wu, Y., Li, L., He, F., Tie, J.: CGSV: An Adaptable Stream-Integrated Grid Monitoring System. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 22–31. Springer, Heidelberg (2005)
7. Wu, D., Angrawal, D., Abbadi, A.E., Singh, A., Smith, T.R.: Efficient Retrieval for Browsing Large Image Databases. In: Proceedings of 5th International Conference on Knowledge Information, pp. 11–18 (1996)
8. Agrawal, R., Faloutsos, C., Swami, A.: Efficient Similarity Search in Sequence Databases. In: Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms, pp. 69–84 (1993)
9. Chan, K., Fu, W.: Efficient Time Series Matching by Wavelets. In: Proceedings of the 15th IEEE International Conference on Data Engineering, pp. 126–133 (1999)
10. Agrawal, R., Lin, K.I., Sawhney, H.S., Shim, K.: Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Times-series Databases. In: Proceedings of 21th International Conference on Very Large Data Bases, pp. 490–500 (1995)
11. Das, G., Lin, K., Mannila, H., Renganathan, G., Smyth, P.: Rule Discovery from Time Series. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, pp. 16–22 (1998)
12. Perng, C., Wang, H., Zhang, S., Parker, S.: Landmarks: a New Model for Similarity based Pattern Querying in Time Series Databases. In: Proceedings of 16th International Conference on Data Engineering, pp. 33–42 (2000)
13. Keogh, E.J., Pazzani, M.J.: Scaling up Dynamic Time Warping to Massive Datasets. In: Żytkow, J.M., Rauch, J. (eds.) Principles of Data Mining and Knowledge Discovery. LNCS (LNAI), vol. 1704, pp. 1–11. Springer, Heidelberg (1999)
14. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 151–162 (2001)
15. CGSV Manual, http://www.chinagrid.edu.cn/CGSV/doc/CGSV-Manual/en/html/ book.html
16. Schroeder, M.: Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise. W.H. Freeman and Company, New York (1991)
17. Bechmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 322–331 (1990)
18. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction To Algorithms, 2nd edn., pp. 127–128. MIT Press, Cambridge