2010 International Conference on Computational and Information Sciences

Java Memory Trajectory Forecasting

Zhang Jing, Lu Kai, Zhou Xu
Department of Computer Science
National University of Defense Technology
Changsha, China
zhjwen520@163.com

Abstract—**Java Memory Trajectory Forecasting is the primary work for energy consumption optimization of memory management system on Java platform. Based on the characteristic of Java memory management, this paper analyzes a large number of memory trajectory data of typical Java applications. On the conclusions obtained from the analysis, this paper designs a prediction algorithm with the combination of memory allocate rate and working memory for Java program memory trajectory and achieves higher prediction accuracy. This work has a significant importance for both Java application performance optimization and Java memory management system energy consumption optimization on Java platform.**

Keywords-Java; memory management; energy consumption optimization; garbage collection; trajectory forecasting

I. INTRODUCTION

Java platform has a specific memory management mechanism---garbage collection. Java program memory trajectory can directly reflect the characteristic of garbage collection and the efficiency of Java platform. So the research and analysis of Java memory trajectory are the foundation of optimization for the Java memory management mechanism. Based on the characteristics analysis of Java program memory trajectory, the laws which are summarized from analysis results can be used to predict the memory trajectory of Java programs. With these laws and prediction methods, the objective of performance optimization for Java platform memory management will be achieved.

At present, energy consumption has become an important factor to measure the system merits. With the development of CPU energy consumption optimization technique[1], CPU energy consumption decreases gradually and memory consumption stands out gradually. Even in some systems, memory has already exceeded CPU to be the most energy-intensive component and its energy consumption proportion in the whole system also increases gradually. Therefore, the study of energy consumption optimization for memory management system becomes urgent. In current condition, Java technology has been very widely used[2], and the study of low power garbage collection algorithm for Java platform become increasingly important. Low power garbage collection algorithm depends on the forecasting of Java memory trajectory. Therefore, it is necessary to analyze the Java program memory trajectory characteristic combining with garbage collection in the period when Java is running, make clear of the laws of memory allocation mode and memory running track etc, and on this basis design appropriate forecasting algorithm to predict memory trajectory of Java programs. Analysis shows that the analysis of Java program memory characteristic and prediction of Java memory trajectory is the principal work to guide the Java memory management system to optimize its energy consumption.

It can be drawn from the above description that Java memory trajectory forecasting has an important significance for improving the efficiency of garbage collection algorithm and reduce energy consumption of Java memory management system. However, the current study of garbage collection behavior is just limited to the tracking algorithm of the life cycle calculation of the object, unable to meet the requirements of energy consumption management. This article tests a large number of memory trajectory data of typical Java applications, and predicts the memory trajectory according to the analysis of Java memory trajectory. After that this article analyzes the prediction result combining to Java program characteristics and its memory trajectory characteristics and illustrates the accuracy of prediction.

The second section describes related work; the third section describes the research methods and analyzes the Java memory trajectory of Spec application; the fourth section designs memory trajectory prediction algorithm and proves the correctness of forecasting algorithm; the fifth section summarizes this work and presents next step work plan.

II. RELATED WORK

A. *Study of the object-level garbage collection characteristics*

Garbage collection tracking algorithm does research on the behavior of garbage collection at the object level, and currently there are mainly Brute Force tracking algorithm, Merlin tracing algorithm and the E-Merlin tracing algorithm[3]. As the Brute Force algorithm is relatively inefficient, while the Merlin algorithm does not use real time as the statistics and the root set is relatively large, the researchers raised the E-Merlin tracing algorithm, which uses real amount of time and do holographic tracking of objects. However, tracing garbage collection algorithm is mainly used to calculate the death time of death objects, the granularity is relatively small, and does not give the behavior characteristics of garbage collection. So its guidance to memory trajectory prediction is not strong.

Supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321801); the HUO Ying-Dong Research Foundation under Grant No. 111072

978-0-7695-4270-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCIS.2010.174

B. *Empirical garbage collection behavior prediction*

Article[4] simply treats frequency as a time unit, based on empirical conclusions of the change of energy consumption, and simply uses historical information to predict memory allocation rate and garbage generation rate. In special hardware conditions and applications, these empirical conclusions may be applicable, but they are not universal, and do not identify memory trajectory in detail.

III. RESEARCHING OF JAVA MEMORY TRAJECTORY

A. *Java Memory Trajectory*

As the Java platform memory management system, garbage collector is primarily to allocate memory and release memory for the running program. The garbage collector allocates memory for objects when the system is running until the heap space is fully occupied. Then the garbage collector triggers a garbage collection, which can recycle a lot of garbage objects and release the memory occupied by the garbage objects. After that the virtual machine can extend the heap space to recover running programs.

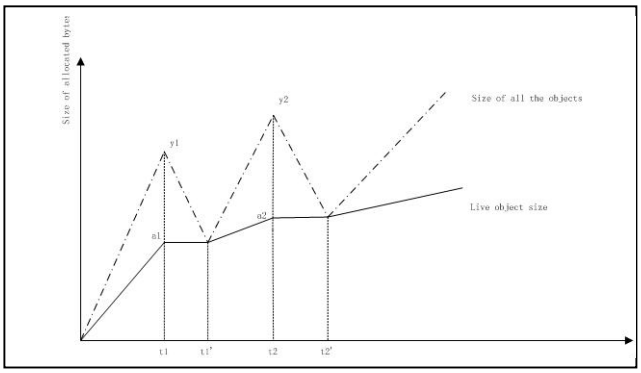

Figure 1. Java memory trajectory graph.

The important parameter reflecting memory allocation of the garbage collector is the memory allocation rate, which is the amount of allocated program memory in a second. Generally, garbage collection time is not taken into consideration when calculating the memory allocation rate. In Figure 1, memory allocation rate is presented as the rise rate of dashed line, but after each garbage collection, dashed line and solid line intersect at one point, and this point is just the survival data structure size of the current system. Because generally memory allocation shall be done with the running of Java programs, while garbage collection is triggered when the heap space is not enough for allocation. After the garbage collection, the garbage is recycled and what remains in the memory are the survival objects. So the memory size of the intersection point is just the amount of memory that occupied by survival objects in current Java programs. This article defines this point as working memory. The trajectory of working memory is not only the parameter that reflects the changes of memory size occupied by survival objects in Java programs, but also the important indicator of memory release.

Therefore, memory allocation rate and working memory are important parameters of Java memory trajectory. Accurately prediction of memory allocation rate and working memory can guide the garbage collector to set a reasonable size of memory space for running program and close excrescent memory to save power.

B. *Java Memory Trajectory Analysis*

To analyze Java memory trajectory, this article selects the typical Java application-SPEC test programs (TABLE 1). SPEC (the Standard Performance Evaluation Corporation) is a global, authoritative third-party application performance testing organization. SPEC application performance testing is a basal test which measures the all-sided system performance in Java applications and can comprehensively test the performance levels of virtual machine running typical Java application, so SPEC application performance testing has the universal representation.

TABLE I. NAMES AND DESCRIPTION OF SPEC APPLICATIONS

Spec applications	Description of Spec applications
compress	Compression Program
db	Database Program
jack	A Java parser generator that is based on the Purdue Compiler Construction Tool Set (PCCTS).
javac	Java Compiler
jess	Language Realization Expert System
mtrt	Raytrace variant, draw three-dimensional dinosaur with two separate threads
raytrace	Draw 3D graphics with ray tracing

This paper made an analysis on the behavior data of spec test-program when running on the JikesRVM2.9.1[5], and used Mark-Compact algorithm[6] in the Java Virtual Machine. In the following diagram, the horizontal axis used ms as unit, and the vertical axis used bytes. The Spec application's initial value and maximum value of memory were both set to 30M in this section. The left column of Figure 2 expressed the changing track of memory usage with time, and the right column was a discrete bar chart which expressed the allocated memory with time at each allocation, the line density in this figure reflects the frequency of memory allocation.

Firstly, memory allocation rate is quite stable and keeps unchanged in the whole Java applications running, but it may express phased due to the frequency and the large objects in allocation. Through comparing the left chart with the right one in Figure 2, memory allocation rate of the applications which has stable memory allocation frequency and rarely allocated large objects were quite stationary. So the stability of memory allocation rate was primarily affected by memory allocation frequency (for example: db) and large object allocations (for example: compress).

Secondly, the working memory remained unchanged or showed a phase change, or periodically in the garbage collection. Working memory of most applications was stable except for mtrt and javac (Figure 2). The working memory of mtrt application changed phased and kept stable in each stage. Because mtrt is a multi-threaded drawing program, when the application ran into a new stage, one thread ended, this could lead to the phased change of working memory of applications. The working memory of javac program changed periodically.

Javac is a compiler with repeating the same operation in each cycle of compiling, so its memory trajectory also reflected periodically. The phased and periodic behavior existed in many applications such as mtrt and javac.

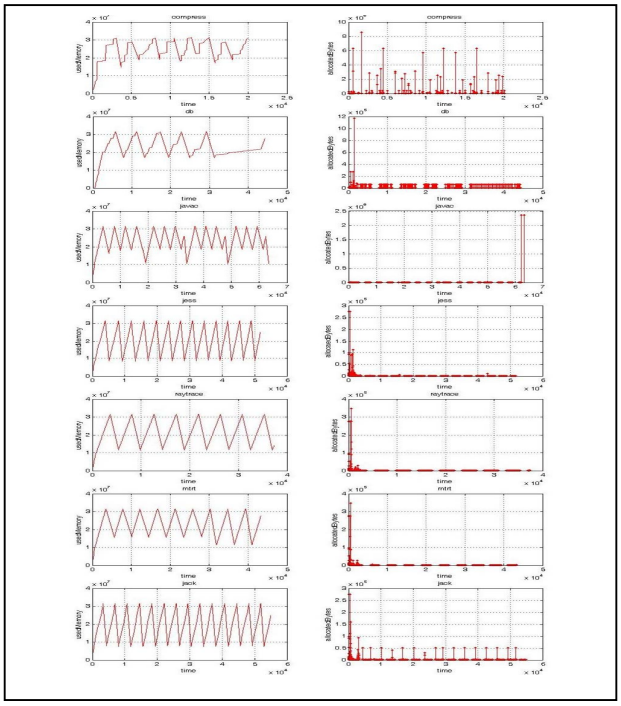

Figure 2. Java memory trajectory of Spec applications

IV. JAVA MEMORY TRAJECTORY FORECASTING

Based on the conclusions obtained above, predictions on Java memory trajectory will be made through the idea of pattern matching.

A. Prediction of Memory Allocation Rate

According to the analysis above, memory allocation rate which is allocated by garbage collector is quite stable and keeps unchanged in the whole Java applications running, so the historical memory allocation rate is an effective basis to predict memory allocation rate in the next running period. Furthermore memory allocation rate may express phased due to the frequency and the large objects in allocation, therefore, the memory information in the current running period has a great reference value for predicting memory allocation rate in the next running period. Commonly, the prediction efficiency of memory allocation rate in the current running period is as 5 times as historical periods. So we use the following formula to predict memory allocation rate(k =1, l =5), Vi will be considered valid if the difference between vi and the historical mean rate is less than 18%, and then take the real rate for Vn in this running period into formula, or take 0.

$$V_{n+1} = (k \cdot \sum_{i=1}^{n-1} V_i + l \cdot V_n) \Big/ (k + l) \qquad (1)$$

Further, memory trajectory of Java programs may be affected by allocation frequency and large object allocation. Programs like db allocates small objects frequently may don't need too big heap, while programs like compress allocates more large objects need larger heap. So the heap size of db and compress can be adjusted like in Table 3.

TABLE II. SPEC PROGRAMS MEMORY ALLOCATION RATE FORECASTING RESULTS

Application names	Size of stack	Accuracy of prediction		
		Ratio above 80%	*Ratio above 90%*	*The best case*
compress	40M	100%	0%	87%
db	25M	84%	53%	98%
jack	30M	100%	85%	99%
javac	30M	75%	58%	99%
jess	30M	100%	92%	99%
mtrt	30M	100%	88%	99%
raytrace	30M	100%	74%	99%

It can be elicited from TABLE 2 that the memory allocation rate predictions of application jess, raytrace, mtrt and jack during the program executing is perfect, but the predictions of application compress, db and javac are affected a lot by the size of object and periodically changes of the working memory. The objects of program compress would be large mostly, so a little difference of object number in each period will make more influence to the prediction of the memory allocation rate. But the fact that all the prediction veracities are above 85 percents proves that the prediction is good as a whole. Opposite to the application compress, though the objects size is small, the diversification of the allocation frequency would have a great affection on the memory allocation rate, so only half of the prediction results have veracity above 95 percents. When it comes to the last executing period of the program db, the memory allocation rate is declined suddenly. Because this sudden change cannot be prognosticated, it brings great windage to the prediction result of the executing period. The program javac is different from the two mentioned above. This program is Java compiler and there will be a preparing work for the next compiling work after each compiling task of programs, so though the memory isn't all occupied, there is a garbage collection after every ending of compiling work. Meantime, the working memory also has a sudden change. During this shorter executing period, garbage collector only allocates memory for a few objects but releases a lot of memory in a very short time, resulting in a high memory allocation rate (Figure 2). Because the length of the objective program which the javac compiled cannot be predicted, the situation that the memory allocation rate has a great sudden change also cannot be predicted. So only 53 percent of the prediction results have veracity above 95 percents, and except for the several executing periods in which the working memory suddenly changed, the prediction veracity of other periods are all over 90 percents.

B. Prediction of Working Memroy

The prediction of the program working memory will be very different from the memory allocation rate. The working memory this article is defined by the size of the existing objects after the garbage collection and its essence is to reflect the size

changing track of the existing objects. According to the analysis to Java memory trajectory, the working memory is stable in most occasions with staged and periodically change. So the prediction can be got by the history mean value when the working memory is stable generally; if the working memory has a jump change, it can be assumed that the working state of the program has changed to the next steady state, and the new working memory should closely to the working memory just after the jump change happened; when the working memory changed periodically, the pattern matching can be used to predict the next value of the working memory. In the method of the pattern matching, the working memory of the latest three times is used to match the historical information, after they are matched, the working memory which follows the matched historical value is chosen as the value of the prediction.

TABLE III. SPEC PROGRAMS WORKING MEMORY FORECASTING RESULTS

Application names.	db	javac	jess	raytrace	mtrt	jack	compress
Garbage collection numbers	8	17	13	8	9	7	7
Numbers of accuracy above 90%	8	15	13	8	9	7	5

The prediction results of working memory are analyzed according to the Java memory trajectory analysis picture of Spec testing program (Figure 2). The working memory of db, jess, raytrace and jack is stable during program running time. According to historical information, it can be accurately predicted. Although the working memory of compress program is also stable during program running, because of its larger objects it happens twice that the accurate rate is less than 80%. The working memory of mtrt has a transition in the 7th garbage collection, and then jumps to the next stage. So prediction accuracy of the 7th period of mtrt reaches only 93%. Finally, it comes to Javac application whose working memory has periodic change. The prediction was quite inaccurate in the 5th and 9th running period due to the jump of working memory, which can't be predicted because there were many reasons that could cause the jump of working memory. But the jump of working memory occurred in the 13th and 17th running period can be accurately predicted through successfully working memory pattern matching, so higher prediction accuracy is got in the two running periods.

C. *Prediction of Memory Trajectory*

According to the periodical wave of Java memory trajectory, the memory size can be decided by the latest working memory and the memory allocation rate of the current wave period.

$$S(I)_{memory} = S(I)_{working\ memory} + T*V'(I)_{memory\ allocation\ rate}$$

$$T = T_{current\ time\ value} - T_{time\ value\ of\ the\ latest\ garbage\ collection}$$

In the formula above, $S(I)_{memory}$ is the memory size for the current system, $S(I)_{working\ memory}$ is the latest working memory size, $V'(I)_{memory\ allocation\ rate}$ is the prediction value for this wave period, T is the time value from latest garbage collection to the moment when the prediction is executed. In our experiment, for each wave period of the testing program, $T_{current\ time\ value}$ is the moment when next garbage collection is triggered. Because Java memory trajectory is affected both by the undulation of working memory and memory allocation rate, the prediction algorithm of Java memory trajectory must consider both the two sides.

TABLE IV. MEMORY TRAJECTORY PREDICTION RESULTS

Application names	Accuracy of prediction	
	Ratio above 80%	*The best case*
compress	83%	86%
db	86%	99%
jack	100%	99%
javac	75%	99%
jess	100%	99%
mtrt	87.5%	99%
raytrace	86%	99%

V. CONCLUSION

Through the analysis of memory trajectory statistical data of Java programming, the method of Java memory trajectory forecasting which consists of memory allocation rate and working memory is very effective for the prediction of the classic Java applications memory trajectory. It also has important guidance significance for the performance optimization of garbage collection. Java application memory trajectory information can be used not only for promoting the performance of garbage collector, but also being the basis to design effective algorithms for low power garbage collection. On the basis of Java memory trajectory forecasting, the next work is to design reasonable garbage collection triggering conditions and low power garbage collection algorithms, which can reduce the energy consumption of Java platform memory management system.

REFERENCES

[1] Chandrakasan A P, Bowhill W J, Fox F. Design of high-performance microprocessor circuits[M]. Wiley-IEEE Press, 2000.

[2] Cramer T, Friedman R, Miller T, et al. Compiling Java just in time[J]. IEEE Micro. 1997, 17(3): 36-43.

[3] FAN Kai_li, LU Kai, LI Gen, et al. The design and implements of E-Merlin garbage collection tracking arithmetic[A]. In: Proceedings of National Software and Application Conference 2007 [C]. Xi'an, 2007, 244-248.

[4] Chen G, Shetty R, Kandemir M, et al. Tuning garbage collection for reducing memory system energy in an embedded java environment[J]. ACM Transactions on Embedded Computing Systems (TECS). 2002, 1(1): 27-55.

[5] Alpern B, Butrico M, Cocchi A, et al. Experiences porting the Jikes RVM to Linux/IA32[C].Proceedings of the 2nd Java™ Virtual Machine Research and Technology Symposium.2002,51-64.

[6] Jones R, Lins R. Garbage collection: algorithms for automatic dynamic memory management[M]. Wiley New York, 1996.