DP: Dynamic Prepage in Postcopy Migration for Fixed-Size Data Load

Shuang Wu, Ce Yang, Jianhai Chen, Qinming He, Bei Wang

DP: Dynamic Prepage in Postcopy Migration for Fixed-Size Data Load

Shuang Wu, Ce Yang, Jianhai Chen, Qinming He, Bei Wang

College of Computer Science, Zhejiang University, Zheda Rd. 38, Hangzhou 310027, China

{catting, yvxiang, chenjh919, hqm, wangbei}@zju.edu.cn

Abstract. Postcopy migration is a mature technology in virtualization. However the performance of postcopy is not stable. We find many memory intensive loads having a high proportion of independent fixed-size data (FSD) cases. To improve migration performance, we present DP: an algorithm which applies to intelligently tackle FSD load during postcopy migration. We implement DP as an online algorithm triggered by remote paging, and adjusting to prepage the most appropriate amount of pages related to recent page fault records. DP also has a threshold processing mechanism to prevent from noise which is derived from load size fluctuation. The experimental results show that DP algorithm can significantly reduce response time and implicit downtime in postcopy migration, with an high improvement on QoS.

1 Postcopy Migration

Postcopy migration[1,2] is one of the common methods in VM live migration. It forces to move the running VM to target side, then fetches pages from source side when page fault occurs, and pushes the remaining pages to target side at the same time.

In this paper we target to improve migration performance by reducing the response time for each page fault in postcopy migration, using a prepage method. Response time is bottleneck for migration. It is the time waiting for fetching pages from source side when page fault occurs. A proper prepage method can fetch a proper amount of pages in advance, and reduce the response time for each case.

2 Fixed-Size Data

We build up the model for FSD load. It has many independent cases and every case needs a certain amount of memory pages *N* to complete its work. If we fetch enough pages (more than *N*), page fault will not occur during the current case.

In practical situations, some cases in FSD-like load need a different amount of pages, which we called noises. Generalized FSD load should have a low *NoiseRate*.

There are many domains in real benchmark which can be magnified as fig.1. below. These domains show the characteristic of FSD with some noises.

adfa, p. 1, 2011.

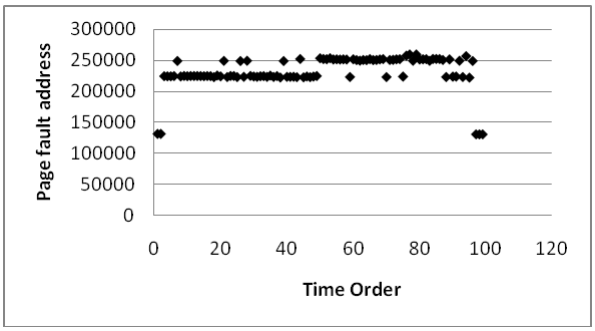

Fig. 1. The FSD-like domains. Page fault addresses are placed in a narrow range.

3 Dynamic Prepage

The purpose of DP is to fetch the most suitable amount of pages before next page fault in a noisy environment. We set a range for guessing from *NMin* to *NMax*, try to include the best amount *N*. Every time we try to fetch an estimated value which called *NTest*, we should decide which range it belongs to, and modify the limit value *NMin* and *NMax* cautiously.

Step 1: We divide the guess result into two situations. If the current page fault address is continuous with the pages we fetched the last time, then they belong to the same case, expressed as (1) below. And the opposite situation expressed as (2).

$$AddrsLast \pm NLast \begin{cases} = Addrs & (1) \\ \neq Addrs & (2) \end{cases}$$

A guessing result *NTest* is smaller than *N* or larger than *N*. The smaller one goes into step 2, and the larger one jump into step 3.

Step 2: From step 1 we get the conclusion $NTest < N$. But now we cannot trust this conclusion immediately. The modify of *NMin* should be very cautious, because we need a reliable value to make sure the subsequent guess is not out of range.

We introduce a new array *MinRecord[5]* to save the recent five *NTest*, and *MinHit* for the continuous times of going into step 2.

When *MinHit* reaches five, we choose the smallest value in *MinRecord[5]* as *NMin*. If noises in record is smaller than *N*, they will not affect the correct conclusion. Otherwise they must be larger than *N*. We avoid these noises by choosing the smallest one in record. Only noises are continuous for five times can lead us to wrong conclusion. Actually the rate of misguidance is related to *NoiseRate* and continuous times we choose to refresh *NMin* as expression 3.

$$Rate_{\text{misguidance}} = NoiseRate^{\text{continuous times}}. \quad (3)$$

When continuous times is 5, we can get a low rate of misguidance about 3.2%, even though *NoiseRate* is up to 50%. This makes the modify of *NMin* credible.

Then we jump into step 4.

Step 3: Similar to step 2, we modify *NMax* cautiously. An array *MaxRecord[5]* to keep records of the recent five *NTest* when it placed in the range between *N* and *NMax*. *MaxHit* saves the continuous times of going into step 3. When MaxHit reaches five we choose the largest one in *MaxRecord[5]*to be *NMax*.

Every time executing step 3, we should reset *MinHit* to zero. Similarly when starting from step 2 we should reset *MaxHit* to zero.

Then the program jump into step 5.

Step 4: Our purpose is to rapidly narrow the gap between *NMin* and *NMax*, and also correctly. So we adjust our guessing value *NTest* according to the current *MinHit*.

We fetch a smaller amount of pages when *MinHit* is close to the predetermined threshold. Otherwise a large adjustment as equation 4.

$$NLast = \frac{NMax - NTest_{last}}{2 \times MinHit}. \quad (4)$$

NLast is the amount we fetch this time and as a parameter transferred to the next round. $NTest_{last}$ is the value last round we guessed. So that *NTest* can be calculated as equation 5.

$$NTest = NTest_{last} + NLast. \quad (5)$$

Then we get the amount of pages and return to Step 1.

Step 5: This time we try to rapidly narrow the gap from the NMax side.

We fetch the amount of pages closer to *NMax* when *MaxHit* is close to the predetermined threshold. Otherwise a further address from *NMax* as equation 6.

$$NTest = NTest_{last} - \frac{NTest_{last} - NMin}{2 \times MaxHit}. \quad (6)$$

Then set the *NLast* equal to *NTest* and return to Step 1.

Strategy of guessing process is as fig. 2. below. *MinHit* and *MaxHit* decide the next position of our guess. The purpose of recording hit times is to rapidly and reliably modify the limit both minimum side and maximum side. Each time going into Step 4 or Step 5, the opposite hit times will reset to zero.

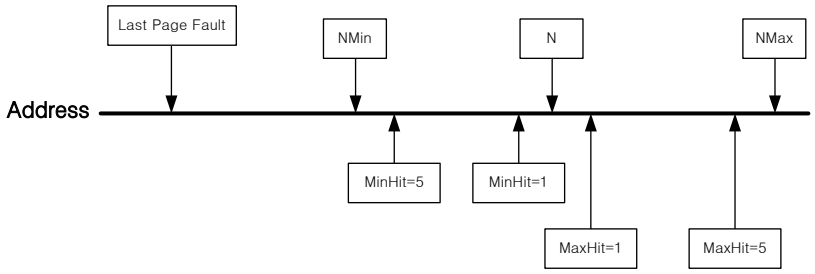

Fig. 2. The way adjusting *NTest* in both smaller side and larger side. Also the amount that we fetch is relating to *MinHit* or *MaxHit*. In this figure we see the different position of next guessing when *MinHit* = 1 and *MinHit* = 5. The same way that *MaxHit* works.

The complexity of DP algorithm is O(N), N is for the rounds of page fault in the whole migration.

4 Experimental Evaluation

We use simulation and benchmark ways to evaluate DP algorithm. In simulation experiment DP algorithm works well with the deviation below 0.05 when noisy rate is lower than 20 as fig.3. below.

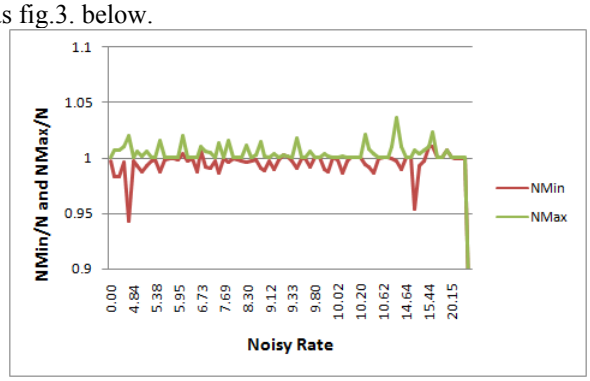

Fig. 3. Noisy rate and the result of *NMin/N* and *NMax/N* after DP algrithm in simulation experiment. Vertical axis shows the result of *NMin* and *NMax*, *N* is always set to be 1. Horizontal axis means the noisy rate for one case.

We add DP algorithm into postcopy migration and test the new migration on QEMU1.4 [3], Xeon CPU 2.13GHz, 16GB memory and gigabit network. In benchmark way we use STREAM [4] which is a FSD-like benchmark. And DP algorithm can reduce 33% response time in benchmark test.

Acknowledgments. This research is partly supported by the key Science and Technology Innovation Team Fund of Zhejiang under Grant~(No.\ 2010R50041).

5 References

1. Hines, M.R., Deshpande, U., Gopalan, K.: Post-copy live migration of virtual machines. ACM SIGOPS Operating Systems Review 43(3), 14–26 (2009)
2. Hirofuchi T, Nakada H, Itoh S, et al.: Reactive consolidation of virtual machines enabled by postcopy live migration. In: Proceedings of the 5th international workshop on Virtualization technologies in distributed computing. ACM, 11-18 (2011).
3. McCalpin, John D.: STREAM: Sustainable memory bandwidth in high performance computers. Silicon Graphics Inc (1995).
4. QEMU, `http://www.qemu.org/`