

RESEARCH ARTICLE

Design of Memory Shifting System Based on Dual-Space Storage Architecture

ZHEHE WANG^{1,2}, SHUANG LI³, JIABAO JIANG⁴, CHUNTENG WANG²,
XIANCHAO WANG⁵, AND TAIYU ZHU⁶

¹School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China

²College of Computer Science and Technology, Hainan Tropical Ocean University, Sanya 572022, China

³College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 200234, China

⁴College of Information Engineering, Chaohu University, Hefei, Anhui 238000, China

⁵School of Computer and Information Engineering, Fuyang Normal University, Fuyang, Anhui 236037, China

⁶Shanghai Jiling Information Technology Company, Shanghai 201108, China

Corresponding authors: Shuang Li (lishuang@shnu.edu.com) and Jiabao Jiang (jjb15820109@163.com)

This work was supported in part by the National Science Foundation of China under Grant 61672006, in part by the Hainan Provincial Natural Science Foundation of China under Grant 121RC1071 and Grant 622MS084, in part by the Education Department of Hainan Province of China under Grant Hnjg2017-46 and Hnjg2022-90, in part by the Natural Science Foundation of Anhui Provincial Education Department of China under Grant KJ2020A0681, in part by the Shanghai Natural Science Foundation of China under Grant 15ZR1415400, and in part by the University-Level General Research Project of Shanghai Normal University of China under Grant SK202121.


ABSTRACT The computer system of the traditional storage architecture has many bottlenecks in massive data transmission, analysis, processing, etc., especially the frequent data migration or copying problem between memory and external storage, which is the most prominent and restricts the full play of CPU performance. To address this, Professor Y. Jin proposed a new solution called dual-space storage architecture based on non-volatile random-access memory (NVRAM) and a data security technique called non-closable window. For the new storage architecture, a corresponding underlying software management model is proposed in this paper. The design scheme of the memory shifting system in this model is introduced in detail, and the feasibility of the scheme is verified through software simulation experiments.

INDEX TERMS Memory shifting system, NVRAM, dual-space storage architecture, non-closeable window.

I. INTRODUCTION

Since the beginning of the 21st century, computer technology has flourished and played a significant role in applications such as massive data storage and processing, memory computing, and data-intensive computing, thereby providing new opportunities for development in each field. However, defects that require frequent data migration between memory and secondary storage are more prominent in traditional computer storage architectures. In response to these defects, academic and industrial research has achieved remarkable results in this area, which can be roughly divided into three categories as shown in **Figure 1**. The first category uses Nor flash(or Nand flash) as a substitute or auxiliary tool for mechanical hard disks [1], [2]. This category is mainly used for

applications with relatively low performance requirements of computing systems, such as solid state drives (SSDs) that replace mechanical hard disk drives for PC and netbooks, and Nor flash(or Nand flash) are used as main memory for embedded systems; the second category uses storage class memory (SCM) [3], [4] as main memory and the buffer between main memory and secondary storage, and the boundaries of memory will no longer be obvious. This category is primarily for enterprise-class or high-performance computing applications and is part of the multicore processor shared memory architecture. For example, IBM, Facebook, Intel, Google, and other research institutes or companies have been researching or adopting storage class memory architecture storage systems [5], [6], [7], which use this structure to build hybrid storage arrays to increase DRAM access speed; Last category uses NVRAM as only storage device [8], [9]. NVRAM technology [10], [11], [12], which combines

The associate editor coordinating the review of this manuscript and approving it for publication was Chong Leong Gan .

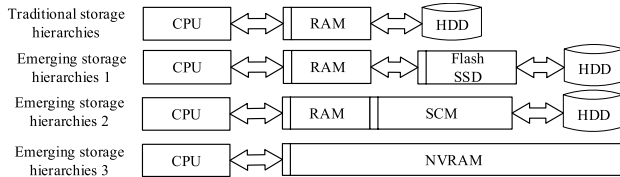


FIGURE 1. Traditional and emerging storage hierarchies.

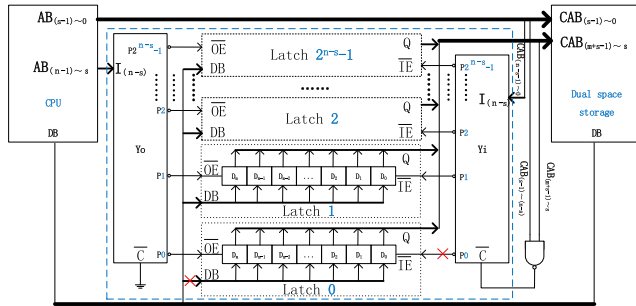


FIGURE 2. Schematic diagram of dual space storage architecture.

low-latency, low-power consumption, non-volatility, high-density, and byte-addressable, has become more mature, and manufacturing costs declining year by year. For example, phase-change memory (PCM), magnetoresistance random-access memory (MRAM), resistive random-access memory (RRAM), and spin-transfer torque random-access memory (STT-RAM), etc., they have both byte-addressable characteristics of memory (such as DRAM) and non-volatile characteristics of external storage (such as HDD). However, most of these storage systems are still in the exploration phase or in small-scale applications owing to the limitation of the processor address line number or manufacturing process of the processor. Currently, the address space that the processor can directly access remains at 4GB (32-bit address lines, personal computer) or 4PB (52-bit address lines, advanced server) [13], and the data to be calculated or processed by the processor still needs to copy data frequently between memory and external storage. In 2013, Jin [14], [15] proposed memory space shifting theory and its implementation technologies. In the same year, a small-capacity dual-space storage with 1GB for the experimental platform was successfully developed, which expanded the CPU addressable space from 2MB to 1GB and zero-copy access data.

II. DUAL SPACE STORAGE ARCHITECTURE AND ITS WORKING PRINCIPLE

A schematic diagram of dual-space storage architecture is shown in Figure 2, which only has dual-space storage, but includes word space and block space, corresponding to memory and secondary storage in the traditional storage hierarchies, respectively. Because they use the same storage medium, the space state is determined by the CPU according to the control signal. Therefore, the components of the dual-space architecture include hardware and software. The

hardware part mainly includes a push and shift latch set, an output decoder, an input decoder, and dual-space storage; the software part mainly includes window wall management, window frame management, shifting vector table management, and shutdown status table management, etc.

The core of building a dual-space storage architecture is design of shifting latch set. This design schematic is shown by the dotted blue box in Figure 2. This technology called memory space shifting technologies automatically map 2^{m+s-n} times ($m > n - s$) larger than itself. Its working principle is summarized as follows.

First, the memory address line output from the processor are divided into two parts: low and high address lines. The low address lines $AB_{(s-1)-0}$ are directly connected to the low address line $CAB_{(s-1)-0}$ of dual space storage; And the high address lines $AB_{(n-s)-s}$ are connected to decoder Y_o through the input signal terminal $I_{(n-s)}$ of Y_o , and then the output signal terminal P_i after the Y_o decoding is strobed to the output control signal terminal \overline{OE} of the corresponding latch $Latch_i$, and finally the value $(D_m D_{m-1} \dots D_0)$ of $Latch_i$ is output to the high address lines $CAB_{(m+s-1)-s}$ of the dual space storage via the data line DB via the output signal terminal Q of the latch.

Secondly, the low address lines $CAB_{(n-s-1)-0}$ of the dual space storage is connected to Y_i through the input signal terminal $I_{(n-s)}$ of decoder Y_i , and then the output control signal terminal P_i via Y_i decoding is gated to the corresponding lock. The write control signal terminal \overline{IE} of latch $Latch_i$, and then the value $(D_m D_{m-1} \dots D_0)$ of the corresponding latch $Latch_i$ is modified to the value on the data line DB , thereby establishing a new mapping relationship to achieve memory space in entire word space. In other words, users can access entire word space by modifying corresponding latch value as needed.

In general, a part of dual-space storage, called non-closeable window, is used to store programs or important data, and these window frames cannot be moved. The value of the corresponding latch will be precluded (in this case, the line marked with red "X" in Figure 2 is turned off) or set by the system designer using software. For example, if the value of each bit of the latch $Latch_0$ in Figure 2 is set to 1 and cannot be arbitrarily modified, then $NO.2^{m-1}$ window wall is regarded as a non-closeable window of the current storage system. The program or data frequently used will be stored in non-closeable window, and mainly includes an initialization program involving the security of the computer system, a transition vector table, an interrupt vector table, a shutdown state table, a window wall management table, a window frame management table, etc., and these parts space play a resident memory role.

III. DESIGN OF MEMORY SHIFTING SYSTEM

Based on the dual-space storage architecture, we propose a dual-space storage management system model that consists of two core subsystems (as shown in Figure 3), including a memory shifting system (blue dashed box) and dual-space

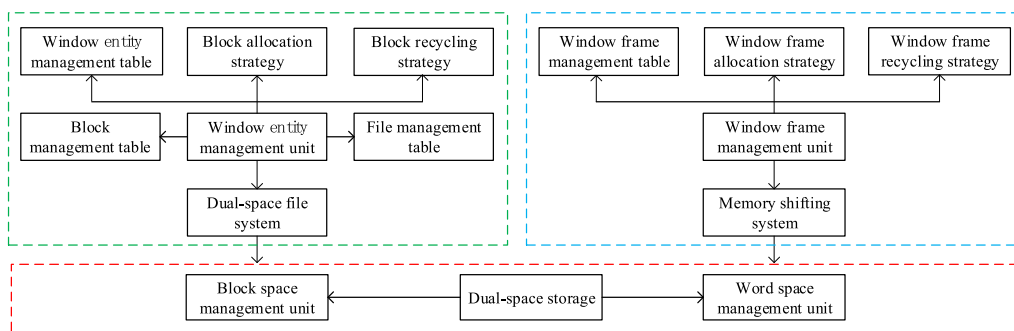


FIGURE 3. Dual space storage management model.

file system (green dashed box), which function similarly to the memory management module and file system in modern operating systems, respectively. In this paper, we discuss only the design scheme of the memory shifting system, and the design scheme of the dual-space file system is described in detail in a new paper.

A. STRUCTURE DESIGN OF THE CORE DATA TABLE

The core data tables of the memory shifting system include window entity management table, window frame management table, window frame status bitmap, shutdown status table, and shifting vector table, etc. Their data structures are as follows.

(1) **window entity management table**

/* Used to record the space usage of the dual-space storage. W_ENTITY_BLOCKS_COUNT is the number of blocks contained in a window entity and DSFS_NAME_LEN is the maximum length of the file name */

```
struct window_entity_table {
//window entity number
unsigned long w_entity_no;
// current position of the window entity
unsigned long w_position;
// whether the user is visible or not
unsigned short user_visible;
// whether window entity is closeable or not
unsigned short closeable;
//data blocks
unsigned long
w_blocks[W_ENTITY_BLOCKS_COUNT];
char file_name[DSFS_NAME_LEN]; //file name
/* file type and operation privileges such as drwxrwxr-x*/
unsigned short i_mode;
/* Leverages out-of-page exceptions in traditional storage
architectures to trigger shifting interruptions*/
struct GDT_PAGE{
// pte_P = 0, trigger fault page exception
unsigned short pte_P;
// pte_RW =, allow reading and writing
unsigned short pte_RW;
```

```
.....// Omitted
};
.....// Omitted
};
```

(2) **window frame management table.**

/* Records the status of the window frame assignment. This table is only required to initialize the settings when the system is booted on the first time but is not loaded repeatedly thereafter. */

```
struct window_frame_table {
// window frame number
unsigned long w_frame_no;
unsigned long w_frame_movable; /*
whether window frame can be moved or not*/
unsigned long lnu_count;// least recently used
unsigned short distribution_mark; /*
assign mark to match the window frame status bitmap*/
unsigned short shared_num; /*
If the shared number is not 0, it means this window frame
cannot be reclaimed temporarily. The shared number is
added to 1 when there is an occupied process, subtracted
from 1 when it is released, and is 0 when there is
no occupation*/
// window entity number
unsigned long w_entity_no;
.....// Omitted
};
```

(3) **window frame status bitmap**

/* Designed to quickly query the usage status of window frames, its data comes from the window frame management table.*/

```
struct frame_state_bmp {
unsigned long w_frame_no;
unsigned short distribution_mark; /*assignment identifier,
0 means the window frame is not assigned, 1 means the
window frame is assigned*/
};
```

(4) **shutdown status table.**

/*Record the status of the computer at the time of shutdown. These status values include the open program status values and the values of the general registers, flag

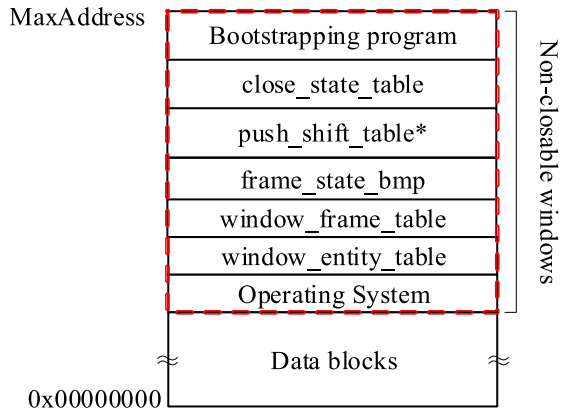


FIGURE 4. Layout of the core tables on the dual space storage.

registers, pointer registers, etc. (stored as a task chain table).*/

```

struct close_state_table {
  unsigned long c_task_id; // task ID
  struct TSS C_task; /* Task status table. Save the
  register information of all opened tasks */
  unsigned short c_task_state; /* Task status.
  The current active task is 1 (only one), and the silent
  task is 2. */
  struct close_state_table * next;
}
/* Task status table. Save the register information
corresponding to the current task*/
struct TSS {
  unsigned long link; // Save the previous
  TSS segment selector
  unsigned long esp0;
  unsigned long eflags;
  unsigned long ldt;
  ..... // Omitted
};

```

(5) shifting vector table

/*Record the value of the shifting latch. This table is implemented directly by the hardware circuit and is listed here for the purpose of illustrating the simulation experiments in this paper.*/

```

struct push_shift_table {
  /*vector number,corresponding window frame number*/
  unsigned long p_shift_no;
  unsigned long p_shift_value; /* vector value,corresponding
  window entity number */
  // wrote port address
  unsigned long write_port_addr;
};

```

The layout of the core data table for dual-space storage is shown by the dashed red box in **Figure 4**. They are generally loaded into non-closable window entities (see the data structure of the window entity table), that can be controlled by hardware or user programs.

B. WORKFLOW DIAGRAM OF MEMORY SHIFTING SYSTEM

According to the working principle of dual space storage, the memory shifting system workflow can be outlined as three steps.

Step 1: The user or program sends a request to the OS to access a file, and then sends the file name to the file system. The file system queries from the window entity management table based on file name. If the corresponding file is not found in the window entity management table, invalid information is sent back to the OS and the OS notifies the user. If the corresponding file is found, then the starting address (dual-space storage address) of the target file and its corresponding window frame and window entity numbers are obtained.

Step 2: According to the window frame number and window entity number obtained in Step 1, the memory shifting system checks whether there is a correspondence between them in the window frame management table. If there is a correspondence between the two, the file is accessed directly according to the target address and the processing result is returned to the OS, which then notifies the user. Go to Step 1. If no correspondence exists between the two, then the interrupt controller sends a shifting interrupt request to the CPU. Then go to Step 3.

Step 3: After the CPU receives the shifting interrupt request, it continues to execute the running program and starts the field protection mechanism immediately after the running program. It then sends the interrupt response signal to the interrupt controller by saving the current values of all registers involved. Immediately afterward, the CPU jumps to execute the shifting interrupt handler, which checks the window frame usage using the window frame status bitmap. If there are free frames, one of the free frames is randomly (or sequentially) assigned to the current request, and the value of the corresponding latch is modified according to the obtained window entity number to establish the correspondence between the currently assigned window frame number and window entity number of the target file, and the window entity management table, window frame management table, and window frame status bitmap are updated simultaneously. Finally, the target file is directly accessed from dual-space storage, the processing result is returned to the OS, and the OS notifies the user. If there is no free window frame, a certain window frame is reclaimed and reassigned to the current request according to the window frame reclaim policy (e.g. LRU), the window frame is shifted, and the window entity management table, window frame management table, window frame status bitmap, etc., are simultaneously updated. Finally, the target file is directly accessed from dual-space storage, the processing result is returned to the OS, and the OS notifies the user. At this point, the processing of the shifting interrupt is finished, and the CPU needs to start the recovery mechanism immediately, and the values of the registers originally stored in the wharves will be stacked out and passed into the corresponding registers in reverse order, and then the CPU will intermittently execute the program instructions that were forced to abort due to the shifting interrupt.

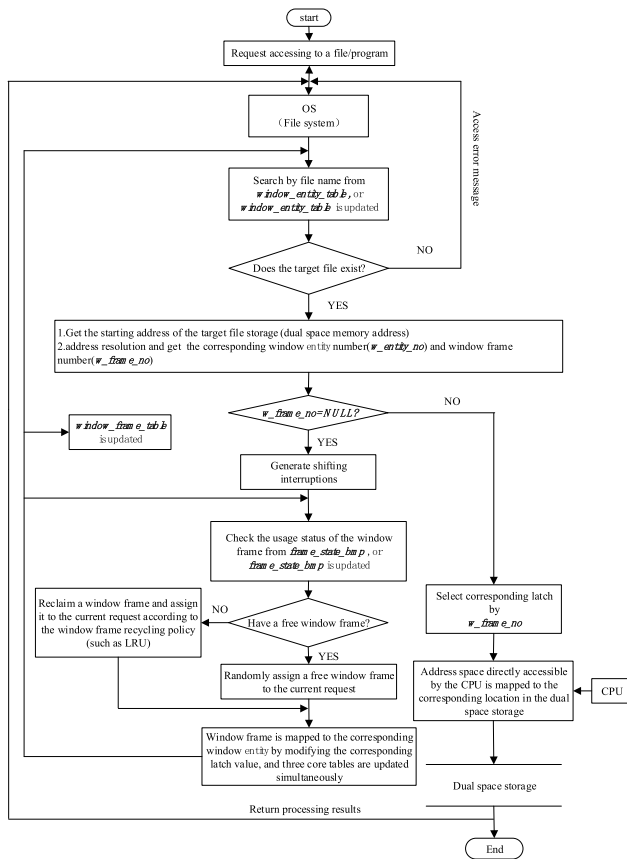


FIGURE 5. Workflow diagram of memory shifting system.

Workflow diagram of memory shifting system is shown in Figure 5.

C. ALLOCATION AND RECYCLING STRATEGY OF WINDOW FRAME

1) WINDOW FRAME ASSIGNMENT STRATEGY

This policy adopts the allocation principle of sequence or priority. If there are multiple processes (tasks) that simultaneously make access requests, the request priority queue is first established and then assigned according to the priority of the processes (tasks) from highest to lowest. Otherwise, the window frame is assigned according to the order of requests, and the implementation process is as follows.

Step 1: Randomly (or sequentially) assign a window frame to the current request if there is a free window frame based on the window frame status bitmap. Otherwise, go to Step 2.

Step 2: One of the allocated window frames is reclaimed according to the window frame reclamation policy, and this window frame is assigned to the current request. For additional details, please refer to Section 3.2.

2) WINDOW FRAME RECYCLING STRATEGY

When there is no free window frame to be allocated, the least recently used policy (LRU) and the program's space local priority principle are used to reclaim the allocated frames

(except for those corresponding to non-closable windows). The window frame with $lru_count = 0$ is the most recently used window frame, and its corresponding window entity number is W_entity_no . The reclamation process is as follows.

Step 1: Calculate the absolute value of the subtraction of the window entity number and W_entity_no in the window frame management table, sort these absolute values in descending order, and prioritize the window frame corresponding to the window entity with the maximum absolute value. If the absolute values are the same, go to Step 2.

Step 2: The value lru_count in the window frame management table are sorted in descending order, and the window frame with the maximum value lru_count is recycled in priority. If the maximum value lru_count is the same, then the first one is taken.

IV. SIMULATION EXPERIMENT OF RESOURCE MANAGEMENT STRATEGY BASED ON DUAL SPACE STORAGE

Currently, many NVRAM are still in the exploratory stage and no products are available. Therefore, this experiment simulated the workflow of a memory shifting system on a PC in order to verify the feasibility of the proposed design scheme.

A. EXPERIMENTAL ENVIRONMENT AND PROCEDURE

1) EXPERIMENTAL ENVIRONMENT

In this experiment, the software and hardware used include: Intel(R) core(TM) i3-5005U CPU @2.00GHz, 4GB DDR3, Windows 7, VMware® Workstation 14 Pro, CentOS 7.0, 16GB USB flash drive, and so on.

2) EXPERIMENTAL DESIGN

According to the construction principle of dual-space storage in Section 2, in this experiment, we assume that the addressable space of the CPU is 2 MB ($= 2^{21}$), where three high address lines of the CPU are used as selected shifting latches (window frames), each latch is 12 bits and is used as the high address line of dual-space storage, represented by the shifting vector table; the remaining 18 low address lines are directly connected to the low address line of dual-space storage. Its addressing space is extended to 1 GB ($= 2^{30}$) by the memory shifting system, and its extension schematic is shown in Figure 6.

3) EXPERIMENTAL PROCEDURE

a) Write the boot program *Boot.bin*, the loader program *Loader.bin*, the simple kernel program *Dsmos_kernel.bin* and the initialization program *Dsmos_init.bin*, where the initialization program includes the installation and initialization of the window wall management table, the shifting vector table, the window frame management table, the window frame status bitmap, the shutdown status table, etc.;

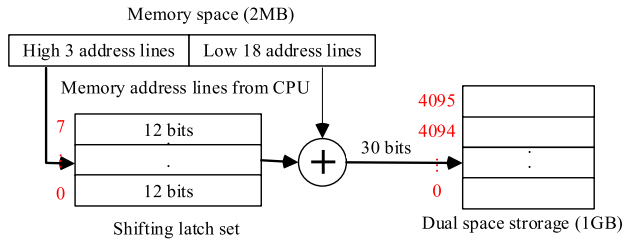


FIGURE 6. Diagram of the expansion of 2MB to 1GB addressable space.

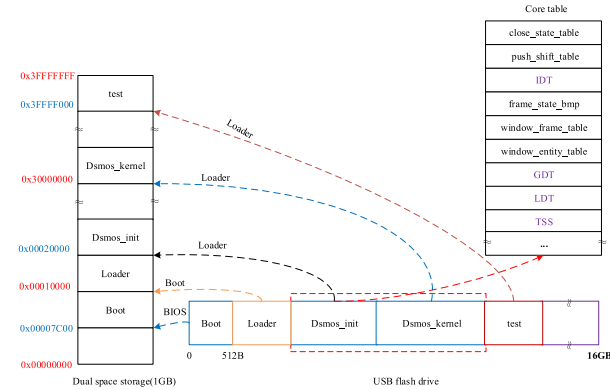


FIGURE 7. Layout of simulation program before and after loading into dual space storage.

```

Please select experiment NO.:
0: Exp0-test example.
1: Exp1-test moveable instruction and watch latch's value changing.
2: Exp2-test to access any word-memory units and write any data into these units randomly.
3: Exp3-test window-frame moving from one window-entity to other window-entity.
4: Exp4-test to verify whether NO.0 window-frame can't be moved to other window-entity from non-closable window.
5: Exp5-test to init latch whether get from high 32 bytes in non-closable window automatically.
6: EXP6-test to get flashid.
7: EXP7-test to initialize dsms core data tables.
8: EXP8-test to generate interrupt(open led).
9: EXP9-test to access any dual space storage address(ds_addr) for pshsystem(after excute NO.7 to init latch).
10: EXP10-test to operate data from any PC address(pc_addr).
11: EXP11-test to operate data from any dual space storage address(ds_addr).
>>
    
```

FIGURE 8. Memory shifting system main interface.

- b) Writing the shifting interrupt-handling simulation program;
- c) Writing of window frame allocation and recycling simulation programs;
- d) Creating a system boot integration image and burning it in a USB flash drive;
- e) Load the integrated image into the specified location of the DDR3 memory using the USB flash drive bootloader, as shown in Figure 7;
- f) Randomly input ten sets of addresses of dual-space storage and verify that the above simulation program execution process is executed according to the preset flow.

B. SIMULATION RESULTS AND ANALYSIS

In this experiment, there are 8 window frames and 4096 window entities. It is assumed that window frame #0, corresponding to window entity #0 and window frame #1 corresponding to window entity #1 are solidified. These are called non-closable window entities. In other words, only six window frames can be moved between window entity #2 and #4095. The main interface appears after the

```

Please select experiment NO.:
0: Exp0-test example.
1: Exp1-test moveable instruction and watch latch's value changing.
2: Exp2-test to access any word-memory units and write any data into these units randomly.
3: Exp3-test window-frame moving from one window-entity to other window-entity.
4: Exp4-test to verify whether NO.0 window-frame can't be moved to other window-entity from non-closable window.
5: Exp5-test to init latch whether get from high 32 bytes in non-closable window automatically.
6: EXP6-test to get flashid.
7: EXP7-test to initialize dsms core data tables.
8: EXP8-test to generate interrupt(open led).
9: EXP9-test to access any dual space storage address(ds_addr) for pshsystem(after excute NO.7 to init latch).
10: EXP10-test to operate data from any PC address(pc_addr).
11: EXP11-test to operate data from any dual space storage address(ds_addr).
>>7<<
All core tables have been initialized successfully.
>>
    
```

FIGURE 9. Initializing all core tables of dual space storage.

simulation program finishes initializing the relevant data and loading the program as shown in Figure 8. There are eleven options, and we mainly choose #7 and #9 to perform the test experiments. Option #7 performs the initialisation of the core tables for dual space storage as shown in Figure 9. By inputting different target addresses, we focus on observing the changes of the four core data tables. For ease of description, the addresses in this simulation are written in 32-bit hexadecimal form. At the same time, considering the length of the article, we only take the target address $ds_addr = 0X00008000$ as an example to analyze the entire access process. And then obtained its high 12-bit address 0000000000000000 ($w_entity_no = 0$) and low 18-bit address 100000000000000000 in Figure 6. Therefore, the high 3-bit address 000 ($w_frame_no = 0$) and the low 18-bit address from ds_addr are stitched together to form the processor's access address $pc_addr = 0X00008000$. According to the workflow diagram of the memory-shifting system (in Figure 5), the first step is to examine the window entity management table to determine whether there is a corresponding window frame exists for window entity #0. Because we have already assumed that window entity #0 is a non-closable window, and it corresponds to window frame #0. In other words, a mapping relationship exists between the two. Target address ds_addr can be accessed directly via pc_addr . Once accessed, the window entity management table (Figure 8), window frame management table (Figure 9), and window frame status bitmap (Figure 10) must be updated simultaneously. The shifting vector table (Figure 11) is mainly used to modify the value of the shifting latch for simulation. It is automatically implemented using a hardware circuit; therefore, it can be ignored. The remaining nine sets of test data are presented in Table 1. In Table 1, the bold blue font indicates that there is already a correspondence between the window frame and window entity; therefore, the target address can be accessed directly. The bold red font indicates that the window frame and window entity do not have correspondence, such as $ds_addr = 0X00FF0020$; the corresponding window frame must be recycled, and the recycled window frame is then assigned to the current request.

From the test results in Table 1, the simulation experiment meets the expected goal, which proves the feasibility of the memory shifting system workflow and the correctness of the window frame allocation and recycling strategy.

TABLE 1. Ten sets of test data and results.

NO	ds_addr	w_entity_no	w_frame_no	pc_addr	lnu_count	Note	result
1	0x0008000	0	0	0X0008000	0	Non-closable window	y
2	0x0004000	1	1	0X0004000	0	Non-closable window	y
3	0x0008000	2	2	0X0008000	1		y
4	0x000C0020	3	3	0X000C0020	1		y
5	0x0018000	6	4	0X0010000	1		y
6	0x001C0020	7	5	0X00140020	1		y
7	0x003C0020	F	6	0X00180020	1		y
8	0x002C0020	B	7	0X001C0020	1		y
9	0x003F0020	F	6	0X001B0020	2	Mapping relationship	y
10	0x00FF0020	3F	7	0X001F0020	2	Recycle and redistribute	y

```

Please select experiment NO.:
.....
>>9-
Please input ds_addr in 0x00000000-0x1FFFFFFF:
>>0008000-
The following data from current window entity management table (only top 10)
-----
w_entity_no    w_frame_no
-----
0x00000000    0x00000000
0x00000001    0x00000001
0x00000002    0x00000008
0x00000003    0x00000008
0x00000004    0x00000008
0x00000005    0x00000008
0x00000006    0x00000008
0x00000007    0x00000008
0x00000008    0x00000008
0x00000000    0x00000008
-----
.....
    
```

FIGURE 10. Window entity management table after accessing to the target address 0X0008000.

```

Please select experiment NO.:
.....
>>9-
Please input ds_addr in 0x00000000-0x1FFFFFFF:
>>0008000-
.....
The following data from current window frame status bitmap
-----
w_frame_no    distribution_mark
-----
0x00000000    1
0x00000001    1
0x00000002    0
0x00000003    0
0x00000004    0
0x00000005    0
0x00000006    0
0x00000007    0
-----
.....
    
```

FIGURE 12. Window frame status bitmap after accessing to the target address 0X0008000.

```

Please select experiment NO.:
.....
>>9-
Please input ds_addr in 0x00000000-0x1FFFFFFF:
>>0008000-
.....
The following data from current window frame management table
-----
w_frame_no    w_entity_no    distribution_mark    lnu_count
-----
0x00000000    0x00000000    1                    0
0x00000001    0x00000001    1                    0
0x00000002    0x00000400    0                    0
0x00000003    0x00000400    0                    0
0x00000004    0x00000400    0                    0
0x00000005    0x00000400    0                    0
0x00000006    0x00000400    0                    0
0x00000007    0x00000400    0                    0
-----
.....
    
```

FIGURE 11. Window frame management table after accessing to the target address 0X0008000.

```

Please select experiment NO.:
.....
>>9-
Please input ds_addr in 0x00000000-0x1FFFFFFF:
>>0008000-
.....
The following data from current shifting vector table
-----
p_shift_no    p_shift_value    write_port_addr
-----
0x00000000    0x00000000    0x081FFFE0
0x00000001    0x00000001    0x081FFFE4
0x00000002    0x00000000    0x081FFFE8
0x00000003    0x00000000    0x081FFFE0
0x00000004    0x00000000    0x081FFFF0
0x00000005    0x00000000    0x081FFFF4
0x00000006    0x00000000    0x081FFFF8
0x00000007    0x00000000    0x081FFFE0
-----
>>
    
```

FIGURE 13. Shifting vector table after accessing to the target address 0X0008000.

V. COMPARISON WITH TRADITIONAL STORAGE ARCHITECTURE AND STORAGE MANAGEMENT MODEL

Compared with traditional storage architecture, the dual-space storage architecture has many advantages.

A. MORE FLEXIBLE SCALABILITY AND COMPATIBILITY

Users do not need to replace major devices such as motherboards or CPUs, but simply choose the right size of dual-space storage modules to be installed in memory slots according to their needs.

B. FASTER ACCESS SPEED

Logically, dual-space storage can be divided into word and block spaces. They belong to the same media and physical storage entity with the same access speed. Next, the speed difference between the two storage architectures is discussed in terms of the time required for the CPU to read the same data D with u MB. Assuming that T_{latch} denotes the time required to modify the shifting latch value (typically in nanoseconds). According to the DDR4 SDRAM data sheet from micron website, read/write speeds of the current mainstream DRAM

can reach to 40GB/s, written as $V_{R_{DRAM}} = V_{W_{DRAM}} = 40\text{GB/s}$. By the literature [10], [12], we can deduce that read speed of PCM about 20GB/s, written as $V_{R_{PCM}} = 20\text{GB/s}$. In the HDD/SSD data sheet from the Western Digital website, typical read/write speeds of HDD can be top out at 200 MB/s, SSD top out at 7100MB/s, and are denoted as $V_{R_{HDD}} = V_{W_{HDD}} = 200\text{MB/s}$, $V_{R_{SSD}} = 7100\text{MB/s}$, respectively. In a conventional storage structure with DRAM/ SSD (or HDD), the time required T_{old} is:

if D is in the DRAM at moment then

$$T_{old} = u/V_{R_{DRAM}} = u/(40 \times 1024) \approx 2.4u \times 10^{-5} \text{ s,}$$

else

$$\begin{aligned} T_{old} &= u/V_{R_{SSD}} + u/V_{W_{DRAM}} + u/V_{R_{DRAM}} \\ &= u/7100 + u/(40 \times 1024) + u/(40 \times 1024) \\ &\approx 16.4u \times 10^{-5} \text{ s,} \end{aligned}$$

at this time, D must first be copied from the HDD into the DRAM;

In the new storage structure, the time required T_{new} is:

if the window entity where D is located has a mapping relationship with the window frame then

$$T_{new} = u/V_{R_{PCM}} = u/(20 \times 1024) \approx 4.8u \times 10^{-5} \text{ s,}$$

else

$$\begin{aligned} T_{new} &= u/V_{R_{PCM}} + T_{latch} \\ &= u/(20 \times 1024) + T_{latch} \approx 4.8u \times 10^{-5} \text{ s,} \end{aligned}$$

Clearly, T_{latch} is negligible when D increases and $T_{new} \ll T_{old}$. If a conventional storage structure with DRAM/HDD, T_{old} will become larger. Although the write operation of PCM is unsatisfactory, it can be solved in the future by improving the fabrication process.

C. SUITABLE FOR MULTI-CORE OR MULTI-PROCESSOR SHARED STORAGE

It is only necessary to modify the value of the corresponding shifting latches to the same value to share the same storage space. Therefore, it is more useful in computation-intensive applications.

D. HIGHER SECURITY

The non-closable window allows hardware or software control. For example, the shift vector table can be solidified by hardware when it leaves the factory. At the same time, core data tables can be installed in the non-closable window, and general users are not allowed to access these space. Therefore, this technology can enhance system security.

Additionally, compared with traditional storage management model, the dual-space storage management model also has some advantages.

1) STORAGE SPACE MANAGEMENT MADE EASY

Storage space is no longer separated into memory and external storage, and only logically divided into word space and block space in the same storage entity. All spaces are allowed random access by byte in units. Other technologies, such as virtual memory and address relocation technology will be discarded and replaced by memory space shifting technology. This technology is automatically implemented using hardware. Therefore, from a space management perspective, we maintain only a few core data tables and leave the remainder of the work to hardware.

2) ZERO COPY

To be compatible with conventional storage devices, dual-space storage is logically divided into word and block space, which correspond to the internal and external memory of conventional storage devices, respectively. However, in practice, both parts of the space can be accessed randomly by bytes. When there is no mapping relationship between window entity and window frame, it is sufficient to modify the value of the corresponding shifting latch. Therefore, there is no need to migrate data between the word space and block space.

3) WORK-ON-START

When the computer system is ready to shut, it automatically records its current running status and writes it into the shutdown status table; When computer system is ready to start, it directly starts from its last status. Therefore, we call it work-on-start(WOS). Therefore, these non-closable windows become a new fast startup mode and security mode, which makes users feel that program initialization or application loading process do not occur.

VI. CONCLUSION

In this paper, we first briefly introduce a dual-spacer storage architecture and its working principle. To address the management issues of this new storage architecture, the design of a memory-shifting system is proposed, and its design and implementation processes are described in detail. The correctness and feasibility of the system design are verified through software simulation experiments. Finally, compared with traditional storage, the advantages and potential applications of this new architecture and its space management model are discussed from the perspective of hardware and space management.

However, the implementation of the proposed memory-shifting system relies on immature NVRAM technology. Therefore, this is verified only through software simulations, and a preliminary prototype of the memory-shifting system is established. This study provides a theoretical basis for future research into dual-space file systems.

ACKNOWLEDGMENT

The authors would like to thank to Prof. Jin Yi, Prof. Shen Yunfu, Dr. Ouyang Shan, all Ph. D candidates, and master's

graduate students of their research center for their kind help and valuable discussions in preparing the article.

DISCLOSURES

The authors declare that there are no conflicts of interest regarding the publication of this article.

REFERENCES

- [1] A. Badam and V. S. Pai, "SSDAlloc: Hybrid SSD/RAM memory management made easy," in *Proc. 8th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Mar. 2011, pp. 1–14.
- [2] P. Yang, "HB-storage: Optimizing SSDs with a HDD write buffer," in *Proc. Int. Conf. Web-Age Inf. Manage.* Berlin, Germany: Springer, 2013, doi: 10.1007/978-3-642-39527-7_5.
- [3] S. W. Fong, C. M. Neumann, and H.-S. P. Wong, "Phase-change memory-towards a storage-class memory," *IEEE Trans. Electron Devices*, vol. 64, no. 11, pp. 4374–4385, Nov. 2017, doi: 10.1109/TED.2017.2746342.
- [4] R. Freitas, "Storage class memory: Technology, systems and applications," in *Proc. IEEE Hot Chips Symp. (HCS)*, Aug. 2010, pp. 1–37, doi: 10.1109/HOTCHIPS.2010.7480060.
- [5] H. Kim, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," *ACM Trans. Storage*, vol. 10, pp. 1–22, Oct. 2014, doi: 10.1145/2668128.
- [6] A. Eisenman, D. Gardner, I. AbdelRahman, J. Axboe, S. Dong, K. Hazelwood, C. Petersen, A. Cidon, and S. Katti, "Reducing DRAM footprint with NVM in Facebook," in *Proc. 13th EuroSys Conf.*, Apr. 2018, pp. 1–13, doi: 10.1145/3190508.3190524.
- [7] *NVML Library*. Accessed: Feb. 2, 2022. [Online]. Available: <http://pmem.io/nvml>
- [8] C. Xiao, L. Zhang, and M. Zhou, "Tnvmalloc: A thread-level-based wear-aware allocator for nonvolatile main memory," *J. Circuits, Syst. Comput.*, vol. 31, no. 4, Mar. 2022, Art. no. 2250066, doi: 10.1142/S0218126622500669.
- [9] Q. Liu and P. Varman, "Ouroboros wear leveling for NVRAM using hierarchical block migration," *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–31, Dec. 2017, doi: 10.1145/3139530.
- [10] Z. Zhang, Z. Wang, T. Shi, C. Bi, F. Rao, Y. Cai, Q. Liu, H. Wu, and P. Zhou, "Memory materials and devices: From concept to application," *InfoMat*, vol. 2, no. 2, pp. 261–290, Mar. 2020, doi: 10.1002/inf2.12077.
- [11] A. Makarov, V. Sverdlov, and S. Selberherr, "Emerging memory technologies: Trends, challenges, and modeling methods," *Microelectron. Rel.*, vol. 52, no. 4, pp. 628–634, 2012, doi: 10.1016/j.microrel.2011.10.020.
- [12] W. Banerjee, "Challenges and applications of emerging nonvolatile memory devices," *Electronics*, vol. 9, no. 6, p. 1029, Jun. 2020, doi: 10.3390/electronics9061029.
- [13] Intel. (2022). *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1*. Accessed: Jul. 16, 2022. [Online]. Available: <https://www.intel.com>
- [14] Y. Jin, S. Ouyang, S. Yunfu, P. Junjie, and X. Liu, "Dual space storage management system and data read/write method," U.S. Patent 10042759, Jun. 7, 2017.
- [15] J. Peng, Y. Shen, S. Ouyang, X. Liu, W. Li, and Y. Jin, "Structure and theory of dual-space storage for ternary optical computer," *SCIENTIA SINICA Informationis*, vol. 46, no. 6, pp. 743–762, Jun. 2016, doi: 10.1360/n112015-00036.



SHUANG LI was born in 1988. She received the Ph.D. degree from the School of Computer Engineering and Science, Shanghai University, Shanghai, in 2019. She is currently a Lecturer with the College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, China. Her research interests include parallel computing, swarm intelligence systems, and ternary optical computer.



JIABAO JIANG was born in 1968. He is currently pursuing the Ph.D. degree with the School of Computer Engineering and Science, Shanghai University. He is an Associate Professor with Chaohu University. His main research interests include embedded systems and ternary optical computers.



CHUNTENG WANG was born in 1976. He is currently an Associate Professor with the School of Computer Engineering, Hainan Tropical Ocean University. His main research interests include storage architecture, file systems, embedded systems, and ternary optical computers.



XIANCHAO WANG was born in Suzhou, Anhui, China in 1973. He received the B.S. degree in mathematics education from Fuyang Normal University, Anhui, China, in 1997, the master's degree in computer science from Northeastern University, Liaoning, China, in 2004, and the Ph.D. degree in computer science from Shanghai University, Shanghai, China, in 2011.

From 2010 to 2016, he was an Assistant Professor at the School of Mathematics and Statistics, Fuyang Normal University, where he has been a Professor with the School of Computer and Information Engineering, since 2017. He has coauthored a book, about 30 conference publications, journal articles, and holds five patents. His research interests include queueing modeling and theory, computer software, optical computing, complex network, big data, and embedded systems. He is a member of ACM and a Senior Member of the China Computer Federation.



ZHEHE WANG was born in 1980. He is currently pursuing the Ph.D. degree with the School of Computer Engineering and Science, Shanghai University. He is a Lecturer with Hainan Tropical Ocean University. His main research interests include storage architecture, file systems, embedded systems, and ternary optical computers.



TAIYU ZHU received the bachelor's degree from the Shenyang Institute of Chemical Technology in 1995. He is currently a Senior Engineer with the Technology Centre of Shanghai Jiling Information Technology Company.

...