Virtual and Cache Memory: Implications for Enhanced Performance of the Computer System

Ugah John Otozi
Department of Computer Science,
Ebonyi State University,
Abakaliki

Chigozie-Okwum Chioma
Department of Computer Science Technology,
Federal College of Land Resources Technology,
Owerri

Ezeanyeji Peter C.
Department of Computer Science
Chukwuemeka Odumegwu Ojukwu University Uli.

Mbaocha Nnamdi Raymond
University of Nigeria Teaching Hospital,
Enugu

ABSTRACT
This paper looks at the performance enhancement which virtual and cache memory brings about in our computer systems. It focuses particularly on their implementation mechanisms and also tries to identify the numerous benefits these memories offers that leads to an overall better performance of computer system. The motivation for this discussion is that many usually wonder if virtual and cache memory is actually needed in the computer system. Some erroneously think that virtual and cache memories perform the same functions as the random access memory (RAM) or even the secondary storage devices. The basic idea with virtual memory is to create an illusion of memory that is as large as a disk and as fast as memory. A computer with virtual memory artfully juggles the conflicting demands of multiple programs within a fixed amount of physical memory. Cache memory on the other hand is a small but very fast chunk of memory that is usually situated very close and directly communicates with the CPU. Cache keeps frequently used data and code very close to the CPU so that repeated use of the same data and areas of memory does not result in repeated slow transactions to main memory each time they are needed. The operations of virtual and cache memory enhances multiprogramming. It also helps to eliminate fragmentation, ensure process flexibility, ensures effective memory management and memory protection. Understanding the mechanism of virtual and cache memory will help operating system students and professionals to appreciate how multiple processes are treated without conflicts.

Keywords
Virtual memory, cache memory, enhancement, replacement algorithm, paging, segmentation

1. INTRODUCTION
In a multitasking operating system, processes share the CPU time and the main memory space. However, sharing the CPU and the main memory sometimes posses some special challenges. As demand on the CPU increases, execution of processes is forced to slow down in some reasonably way. If too many processes need too much memory space, then some of the processes will simply not be able to run due to the fact that they will run out of memory space. When a process is out of space, it is said to be out of luck. This condition could cause execution of such process to be aborted before completion. Furthermore, memory is also vulnerable to corruption. Sometimes, the operating system may unintentionally allocate a memory space which is already being used by a given process to another process. When this happens, the process last allocated to that same memory space may fail. In order to help manage memory space more effectively and efficiently with fewer errors, modern computer systems provide an abstraction of main memory known as virtual memory (VM).

The central processing unit (CPU) is the brain of the computer. All of the instructions have to run through the CPU for the various parts of a computer to work together. CPU chips have been getting smaller and faster as chip technology has advanced. One of the slower aspects of computer processing is the interaction between the CPU chip and the main memory- random-access memory (RAM). Installing more memory is not always a solution to this challenge of slow interaction because the major issue is always the chunk of time it takes the CPU to access the memory. To this end chip designers came up with a small form of memory located directly on the chip itself called the cache memory. It is much smaller, but can be accessed much faster than the main memory. The CPU cache stores the most frequently used pieces of information in the cache memory so that this information can be retrieved more quickly. This information is a duplicate of information stored elsewhere, but it is more readily available. By these arrangements, the overall performance of most modern computer systems has been greatly enhanced.

2. CONCEPTUAL OVERVIEW OF VIRTUAL MEMORY
According to [1] Virtual memory is a valuable concept in computer architecture that allows you to run large, sophisticated programs on a computer even if it has a relatively small amount of RAM. RAM has billions of memory locations but sometimes even that is not enough room for all the data the CPU needs. When RAM gets too full, the computer's operating system can help out by temporarily marking sections of secondary storage for the CPU to use as a kind of extra memory. These sections are called virtual memory. The operating system creates a 'swap file' in this area which is used to hold data the CPU does not need immediately. Both Windows and Linux support virtual memory.

A computer with virtual memory artfully juggles the conflicting demands of multiple programs within a fixed amount of physical memory. A computer that's low on memory can run the same programs as one with abundant RAM, although more slowly. The term "virtual memory" refers to space allocated on a hard drive where data can be stored for rapid access. Virtual memory is slower than solid-

state memory chips so it is typically used as backup memory in certain situations [2]. The basic idea with virtual memory is to create an illusion of memory that is as large as a disk (in gigabytes) and as fast as memory (in nanoseconds). The key principle is locality of reference, which recognizes that a significant percentage of memory accesses in a running program are made to a subset of its pages. Or simply put, a running program only needs access to a portion of its virtual address space at a given time. Virtual memory is a component of most operating systems, such as MAC OS, Windows and Linux. Virtual memory has a very important role in the operating system. It allows us to run more applications on the system than we have enough physical memory to support. Virtual memory is simulated memory that is written to a file on the hard drive. In the case of Windows it is a file called pagefile.sys. The process of moving data from RAM to disk (and back) is known as swapping or paging. When there is no more space in physical RAM, the Virtual memory manager will take the least used application and place it in the page file on the hard drive. The process of taking an application from the physical RAM and putting it in the page file is called paging out. The process of moving the application from the page file back into physical RAM is called paging in. Disk thrashing occurs when the amount of physical memory is too low. In that case the data must constantly be moved from physical RAM, to disk, and back again, [3]. A computer accesses the contents of its RAM through a system of addresses, which are essentially numbers that locate each byte. Because the amount of memory varies from computer to computer, determining which software will work on a given computer becomes complicated. Virtual memory solves this problem by treating each computer as if it has a large amount of RAM and each program as if it uses the PC exclusively. The operating system, such as Microsoft Windows or Apple's OS X, creates a set of virtual addresses for each program. The OS translates virtual addresses into physical ones, dynamically fitting programs into RAM as it becomes available, [4]. Virtual memory is volatile. If the computer is turned off, the operating system loses track of what was kept where in virtual memory. The data is lost. But even though the CPU can directly access virtual memory, it is slow.

When the CPU needs data held in virtual memory, it asks the operating system to first load it into RAM, which is quick to access. Less-used data is moved from RAM to virtual memory. Data that the CPU needs to use right now moves from virtual memory to RAM.

2.1 Conceptual Overview of Cache Memory

Cache memory is a small high speed memory usually Static RAM (SRAM) that contains the most recently accessed pieces of main memory. They are the high speed buffers which are inserted between the processors and main memory to capture those portions of the contents of the main memory which are currently in use. Since cache memories are typically 5 -10 times faster than main memory they can reduce the effective memory access time if carefully designed and implemented, [5]. The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program. If the CPU finds the data in the cache memory, it does not have to go looking for such data again in the main memory or even the secondary storage media. Most programs use very few resources once they have been opened and operated for a time, mainly because frequently re-referenced instructions tend to be cached. This explains why measurements of system performance in computers with nslower processors but larger caches tend to be faster than measurements of system performance in computers with faster processors but more limited cache space. This simply mean that, the less frequently access is made to certain data or instructions, the lower down the cache level the data or instructions are written, [6].

2.1.1 Types of Cache Memory

According to [7], there are two types of cache memory which includes memory cache and the Disk cache.

- **Memory Cache**: A memory cache sometimes called a cache store or RAM cache is a portion of memory made of high-speed static RAM (SRAM) instead of the slower and cheaper dynamic RAM (DRAM) used for main memory. Memory caching is effective because most programs access the same data or instruction over and over. By keeping as much of this information as possible in SRAM, the computer avoids accessing the slower DRAM.
- **Disk Cache:** Disk caching works under an undistinguishable guideline from memory reserving, yet as opposed to utilizing rapid static- RAM, a plate/disk cache utilizes traditional primary memory. When a program needs to get to information from the disk, it first checks the circle cache to check whether the information is there. Disk caching drastically enhances the execution of programs as getting to a byte of information in RAM can be a huge number of time quicker than getting to a byte on a hard disk.

2.1.2 Levels of Cache memory

According to [8], type of Cache Memory is divided into different levels that are L1, L2, L3:

- **Level 1 (L1) cache or Primary Cache:** L1 is the primary type cache memory. The Size of the L1 cache very small comparison to others that is between 2KB to 64KB, it depends on computer processor. It is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).It is an embedded register in the computer microprocessor(CPU).The Instructions that are required by the CPU that are firstly searched in L1 Cache. Examples of registers are accumulator, address register, Program counter etc.
- **Level 2 (L2) cache or Secondary Cache:** L2 is secondary type cache memory. The Size of the L2 cache is more capacious than L1 that is between 256KB to 512KB.L2 cache is Located on computer microprocessor. After searching the Instructions in L1 Cache, if not found then it searched into L2 cache by computer microprocessor. The high-speed system bus interconnecting the cache to the microprocessor.
- **Level 3 (L3) cache or Main Memory:** The L3 cache is larger in size but also slower in speed than L1 and L2,it's size is between 1MB to 8MB.In Multi-core processors, each core may have separate L1 and L2,but all core share a common L3 cache. L3 cache double speed than the RAM.

3. PRINCIPLES OF OPERATION OF VIRTUAL AND CACHE MEMORY

3.1 Virtual Memory Implementation Techniques

Virtual memory is commonly implemented by demand paging and Demand segmentation

3.1.1 Demand Paging

Paging is a technique in which physical memory is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes), [9]. When a process is to be executed, its corresponding pages are loaded into any available memory frames. Logical address space of a process can be non-contiguous and a process is allocated physical memory whenever the free memory frame is available. Operating system keeps track of all free frames. Operating system needs "n" free frames to run a program of size "n" pages. External fragmentation is avoided by using paging technique. A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced. From Figure 3 above it is seen that while executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

Address generated by CPU is divided into two:

- **Page number (p)** : page number is used as an index into a page table which contains base address of each page in physical memory.
- **Page offset (d)** : page offset is combined with base address to define the physical memory address.

Advantages of Demand Paging: Listed below are the advantages of demand paging among others:

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging: some disadvantages of demand paging include:

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques**.**

3.1.2 Demand Segmentation

Segmentation is a technique to break memory into logical pieces where each piece represents a group of related information, (IDC Technologies, n.d). For example, data segments or code segment for each process, data segment for operating system and so on. Segmentation can be implemented using or without using paging. Unlike paging, segments have varying sizes and thus eliminate internal fragmentation. External fragmentation still exists but to lesser extent. Address generated by CPU is divided into two namely;

- **Segment number (s) --** segment number is used as an index into a segment table which contains base address of each segment in physical memory and a limit of segment.
- **Segment offset (o) --** segment offset is first checked against limit and then is combined with base address to define the physical memory address.

3.1.3 Page Fault

A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory, [10]. The virtual memory system uses something called a "page table" to map virtual addresses to physical addresses. Since our machine could possibly have less RAM than our program thinks it has, it's possible to have more virtual addresses than physical addresses. That means not all virtual addresses in a page table will have a valid corresponding physical address (i.e. not all virtual addresses will have a valid entry in the page table). If a virtual address has no valid entry in the page table, then any attempt by your program to access that virtual address will cause a **page fault** to occur— if you're familiar with software exceptions (like in C++, Java, Python, etc.), a page fault is very much like an exception, except in hardware, rather than software. The page fault happens because the requested virtual address actually corresponds to a page that is currently sitting on disk, rather than in RAM (and therefore the virtual address cannot possibly be translated into a physical address).

Handling Page Faults

According to [11], when the hardware raises a page fault interrupt, the page fault handler follows the following steps to handle the page fault.

1. Check the location of the referenced page in the PMT
2. If a page fault occurred, call on the operating system to fix it
3. Using the frame replacement algorithm, find the frame location
4. Read the data from disk to memory
5. Update the page map table for the process
6. The instruction that caused the page fault is restarted when the process resumes execution.

3.1.4 Page Replacement Algorithms

Page replacement algorithms are the techniques with which an operating system decides which memory pages to swap out or write to disk when a page of memory needs to be allocated, [12]. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages. When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm. A page replacement algorithm looks at the

limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults.

First in First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Optimal Page algorithm: An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Least Recently Used (LRU) algorithm: Page which has not been used for the longest time in main memory is the one which will be selected for replacement.

- Easy to implement, keep a list, replace pages by looking back into time.

Page Buffering algorithm: This algorithm involves:

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool, [13].

Least frequently Used (LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used (MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

3.1.5 Cache Memory configurations

Cache memory is configured such that, whenever data is to be read from RAM, the system hardware first checks to determine if the desired data is in cache. If the data is in cache, it is quickly retrieved, and used by the CPU. However, if the data is not in cache, the data is read from RAM and, while being transferred to the CPU, is also placed in cache (in case it is needed again later). From the perspective of the CPU, all this is done transparently, so that the only difference between accessing data in cache and accessing data in RAM is the amount of time it takes for the data to be returned. Caching configurations continue to evolve, but memory cache traditionally works under three different configurations :

- **Direct mapping:** In direct mapping, each block is mapped to exactly one cache location. Conceptually, this is like rows in a table with three columns: the data block or cache line that contains the actual data fetched and stored, a tag that contains all or part of the address of the fetched data, and a flag bit that connotes the presence of a valid bit of data in the row entry.
- **Fully associative mapping:** In fully associative mapping**,** structure, the operating system allows a block to be mapped to any cache location rather than to a pre-specified cache location (as is the case with direct mapping).
- **Set associative mapping:** This mapping techniques can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called *N-way set associative mapping*, which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

Specialized caches: In addition to instruction and data caches, there are other caches designed to provide specialized functions in a system. By some definitions, the L3 cache is a specialized cache because of its shared design. Other definitions separate instruction caching from data caching, referring to each as a specialized cache.
Other specialized memory caches include the translation look-aside buffer (TLB) whose function is to record virtual address to physical address translations. Still other caches are not, technically speaking, memory caches at all. Disk caches, for example, may leverage RAM or flash memory to provide much the same kind of data caching as memory caches do with CPU instructions. If data is frequently accessed from disk, it is cached into DRAM or flash-based silicon storage technology for faster access and response.

3.1.6 Cache Replacement Algorithm

A cache algorithm is a detailed list of instructions that directs which items should be discarded in a computing device's cache of information. Cache Replacement Algorithms are only needed for associative and set associative techniques. There are several Cache replacement algorithms according to [14], and they include the following:

First-in First-out (FIFO) – First in First out is a cache replacement algorithm that replaces the cache line that has been in the cache the longest.

Belady's Algorithm: The most efficient caching algorithm would be to always discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Belady's optimal algorithm or the clairvoyant algorithm. Since it is generally impossible to predict how far in the future information will be needed, this is generally not implementable in practice. The practical minimum can be calculated only after experimentation, and one can compare the effectiveness of the actually chosen cache algorithm.

Least Recently Used (LRU): This algorithm discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards the least recently used item. General implementations of this technique require keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes. LRU is actually a family of caching algorithms with members including: 2Q by Theodore Johnson and Dennis Shasha and LRU/K by Pat O'Neil, Betty O'Neil and Gerhard Weikum.

Most Recently Used (MRU): This algorithm discards, in contrast to LRU, the most recently used items first. In findings presented at the 11th VLDB conference, Chou and Dewitt noted that when a file is being repeatedly scanned in a looping sequential reference pattern, MRU is the best replacement algorithm." [15] submitted that Subsequently other researchers presenting at the 22nd VLDB conference noted that for random access patterns and repeated scans over large datasets (sometimes known as cyclic access patterns) MRU cache algorithms have more hits than LRU due to their tendency to retain older data. MRU algorithms are most useful in situations where the older items are more likely to be accessed.

Pseudo-LRU: (PLRU) For CPU caches with large associativity (generally >4 ways), the implementation cost of LRU becomes prohibitive. In many CPU caches, a scheme that almost always discards one of the least recently used items is sufficient. So many CPU designers choose a PLRU algorithm which only needs one bit per cache item to work. PLRU typically has a slightly worse miss ratio, has a slightly better latency, and uses slightly less power than LRU.

Random Replacement (RR): Randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history. For its simplicity, it has been used in ARM processors. It admits efficient stochastic simulation.

Segmented LRU (SLRU): An SLRU cache is divided into two segments, a probationary segment and a protected segment. Lines in each segment are ordered from the most to the least recently accessed. Data from misses is added to the cache at the most recently accessed end of the probationary segment. Hits are removed from wherever they currently reside and added to the most recently accessed end of the protected segment. Lines in the protected segment have thus been accessed at least twice. The protected segment is finite, so migration of a line from the probationary segment to the protected segment may force the migration of the LRU line in the protected segment to the most recently used (MRU) end of the probationary segment, giving this line another chance to be accessed before being replaced. The size limit on the protected segment is an SLRU parameter that varies according to the I/O workload patterns. Whenever data must be discarded from the cache, lines are obtained from the LRU end of the probationary segment.

3.1.7 Write Policies

A write policy determines how the cache deals with a write cycle. [16] Submits that the two common write policies are **Write-Back** and **Write-Through**.

- **Write-Back policy**: Here the cache acts like a buffer. That is, when the processor starts a write cycle the cache receives the data and terminates the cycle. The cache then writes the data back to main memory when the system bus is available. This method provides the greatest performance by allowing the processor to continue its tasks while main memory is updated at a later time. However, controlling writes to main memory increase the cache's complexity and cost.
- **Write-Through policy:** This is the second write policy method. As the name implies, the processor writes through the cache to main memory. The cache may update its contents, however the write cycle does not end until the data is stored into main memory. This method is less complex and therefore less expensive to implement. The performance with a Write-Through policy is lower since the processor must wait for main memory to accept the data.

4. IMPORTANCE OF VIRTUAL MEMORY AND MEMORY CACHING IN THE COMPUTER SYSTEM

Virtual memory techniques and memory caching are both important and helps in optimizing system utilization. Their importances are stated in the sections below:

4.1 Importance of Virtual Memory Technique in Computer Systems

Virtual memory technique and its implementations are very vital to the overall functionality of the computer system. The Importance of virtual memory technique includes:

- **Flexibility:** If computers only relied on the main memory chips, far less memory would be available and the usefulness of many software programs would be severely limited. Even though virtual memory is slower, it is still useful because it greatly expands a computer's functionality.

- **Saves Cost and Time:** When virtual memory was first created, solid-state memory chips were much smaller and more expensive. However, today's memory chips can store many gigabytes of data at very low cost and moreover as memory chips continue to grow in capacity, prices are falling also.

- **Makes Multiprogramming Easier:** [17] further states that When a computer user opens multiple programs at once, the data for these programs must be stored in memory for quick access. The more programs are opened, the more memory is needed. When the computer's physical memory is full, the excess data is stored in virtual memory. Virtual memory with paging lets a computer run many programs at the same time, almost regardless of available RAM. This benefit is called multiprogramming and it is a key feature of modern PC operating systems. This feature enable modern computers to accommodate many utility programs such as printer drivers, network managers and virus scanners at the same time as your applications -- Web browsers, word processors, email and media players.

- **Makes it easier for Programmers to Write and Run Large Programs:** In addition to multitasking, virtual memory allows

programmers to create larger and more complex applications. When these programs are running, they occupy physical memory as well as virtual memory.

- **Paging File:** With virtual memory, the computer writes program pages that have not been recently used to an area on the hard drive called a paging file. The file saves the data contained in the pages; if the program needs it again, the operating system reloads it when RAM becomes available. When many programs compete for RAM, the act of swapping pages to the file can slow a computer's processing speed, as it spends more time doing memory management chores and less time getting useful work done. Ideally, a computer will have enough RAM to handle the demands of many programs, minimizing the time the computer spends managing its pages.

- **VM as a Tool for Caching:** Conceptually, a virtual memory is organized as an array of N contiguous byte-sized cells stored on disk. Each byte has a unique virtual address that serves as an index into the array. The contents of the array on disk are cached in main memory. As with any other cache in the memory hierarchy, the data on disk (the lower level) is partitioned into blocks that serve as the transfer units between the disk and the main memory (the upper level). Virtual memory systems handle this by partitioning the virtual memory into fixed-sized blocks called *virtual pages (VPs)*. Each virtual page is P = 2p bytes in size. Similarly, physical memory is partitioned into *physical pages (PPs)*, also P bytes in size. (Physical pages are also referred to as *page frames*) [18]. At any point in time, the set of virtual pages is partitioned into three disjoint subsets:

• **Unallocated:** Pages that have not yet been allocated (or created) by the VM system. Unallocated blocks do not have any data associated with them, and thus do not occupy any space on disk.
• **Cached:** Allocated pages that are currently cached in physical memory.
• **Uncached:** Allocated pages that are not cached in physical memory.

- **VM as a Tool for Memory Protection:** Any modern computer system must provide the means for the operating system to control access to the memory system. A user process should not be allowed to modify its read-only text section. Nor should it be allowed to read or modify any of the code and data structures in the kernel. It should not be allowed to read or write the private memory of other processes, and it should not be allowed to modify any virtual pages that are shared with other processes, unless all parties explicitly allow it (via calls to explicit inter-process communication system calls). A computer without virtual memory can still run many programs at the same time, although one program might change, accidentally or deliberately, the data in another if its addresses point to the wrong program. Virtual memory prevents this situation because a program never "sees" its physical addresses. The virtual memory manager protects the data in one program from changes by another.

- **VM as a Tool for Memory Management:** The combination of demand paging and separate virtual address spaces has a profound impact on the way that memory is used and managed in a system. In particular, Virtual Memory simplifies linking and loading, the sharing of code and data, and allocating memory to applications.
  • **Simplifying linking:** A separate address space allows each process to use the same basic format for its memory image, regardless of where the code and data actually reside in physical memory.
  • **Simplifying loading:** Virtual memory also makes it easy to load executable and shared object files into memory.
  • **Simplifying sharing:** Separate address spaces provide the operating system with a consistent mechanism for managing sharing between user processes and the operating system itself. In general, each process has its own private code, data, heap, and stack areas that are not shared with any other process. In this case, the operating system creates page tables that map the corresponding virtual pages to disjoint physical pages. However, in some instances it is desirable for processes to share code and data.
  • **Simplifying memory allocation:** Virtual memory provides a simple mechanism for allocating additional memory to user processes. When a program running in a user process requests additional heap space (e.g., as a result of calling malloc), the operating system allocates an appropriate number, say k, of contiguous virtual memory pages, and maps them to k arbitrary physical pages located anywhere in physical memory. Because of the way page tables work, there is no need for the operating system to locate k contiguous pages of physical memory. The pages can be scattered randomly in physical memory.

4.2 Importance of Cache Memory in Computer Systems

Cache Memory is expensive to implement and have limited capacity but they are very important in achieving optimum performance of the computer system. According to [19],[20], the following are the importance of cache memory:

- **Speed:** Cache memory is faster than the main memory; hence deploying cache memory increases the speed of processing tremendously. Cache memory consumes less access time when compared to the main memory.

- **Quick Access to Frequently used data**: The cache memory stores data for temporary use, storing the programs that can be executed within a short period of time. To this end, access to frequently used data is achieved at a quicker rate.

- **Reduction of Latency:** Analytical and transactional workloads have reduced query-response time because the solid-state drive (SSD) storage has

lower latencies. If you use server-side caching, the average latency for a transactional workload can be reduced by half.

- **Increased Throughput:** Online transaction processing (OLTP) workloads have higher transaction rates because the Solid State Drives storage provides better throughput.

- **Write throughput:** In environments where the storage area network (SAN) is congested, the flash device, which is used as a cache, can offload a significant percentage of read requests. When read requests are offloaded, the SAN can have better write throughput, and can effectively serve a larger number of clients and hosts.

- **Smaller memory footprint:** If a flash cache device is configured, some workloads can perform even with a lower memory footprint.

5. WHY VIRTUAL AND CACHE MEMORIES ARE NEEDED IN OUR COMPUTER

Virtual memory is an elegant interaction of hardware exceptions, hardware address translation, main memory, disk files, and kernel software that provides each process with a large, uniform, and private address space. With one clean mechanism, virtual memory provides three important capabilities.

- It uses main memory efficiently by treating it as a cache for an address space stored on disk, keeping only the active areas in main memory, and transferring data back and forth between disk and memory as needed.

- It simplifies memory management by providing each process with a uniform address space.

- It protects the address space of each process from corruption by other processes. Virtual memory is one of the great ideas in computer systems. A major reason for its success is that it works silently and automatically, without any intervention from the application programmer. Since virtual memory works so well behind the scenes, why would a programmer need to understand it? There are several reasons. Some of them are discussed below

Virtual memory is central. Virtual memory pervades all levels of computer systems, playing key roles in the design of hardware exceptions, assemblers, linkers, loaders, shared objects, files, and processes. Understanding virtual memory will help one better understand how the computer system works in general. Understanding virtual memory will help you harness its powerful capabilities in your applications. Virtual memory gives applications powerful capabilities to create and destroy chunks of memory, map chunks of memory to portions of disk files, and share memory with other processes. However, it is expedient for us to understand that virtual memory could also be dangerous. This is because virtual memory applications interact with virtual memory every time they reference a variable, dereference a pointer, or make a call to a dynamic allocation package such as malloc. If virtual memory is used improperly, applications can suffer from perplexing and insidious memory related bugs. For example, a program with a bad pointer can crash immediately with a "Segmentation fault" or a "Protection fault," run silently for hours before crashing, or run to completion with incorrect results. Understanding how virtual memory works and the allocation packages such as malloc that manages it can help one avoid these errors.

On the other hand, Cache is a small amount of memory which is physically closer to the CPU than RAM is. The more cache there is, the more data can be stored closer to the CPU. Cache memory is beneficial because:

- Cache memory holds frequently used instructions/data which the processor may require next and it is faster access memory than RAM, since it is on the same chip as the processor. This reduces the need for frequent slower memory retrievals from main memory, which may otherwise keep the CPU waiting.

- The more cache the CPU has, the less time the computer spends accessing slower main memory and as a result programs may run faster.

6. CONCLUSION

Virtual memory is a technique for allowing the computer to act as though it has more physical memory by using the hard drive (which is almost always much larger than main memory) as temporary storage space. If the session reaches a point where programs are asking for more memory space than is available, the operating system will look for the pages of RAM that have been least recently accessed, and will move their contents off to the hard drive to make room for the new requests for space in main RAM. If one of those other programs then comes back and attempts to access one of the pages that is no longer resident, the CPU traps this situation and lets the operating system know that page is needed again, at which point the operating system will go and retrieve it from the hard drive again and swap it with something else. This is a much slower process, of course, than simply being able to access RAM directly in the first place, so having enough memory in the system in the first place to avoid having to make frequent use of the virtual memory space helps significantly with overall performance. Memory virtualization does have other uses beyond artificial expansion of physical RAM, such as giving applications the appearance of having their own unique address space to run in, which helps to isolate them from other processes running on the system. Cache memory on the other hand is a small but very fast (compared to main RAM) chunk of memory that usually resides directly on the same die as the CPU cores themselves. The purpose of cache is to keep frequently used data and code as close to the CPU core as possible, so that repeated use of the same areas of memory does not result in repeated slow transactions to main memory. Since small loops are a very common construct in fully compiled code, having all of the code and data that it uses be accessible at the full speed of the processor allows the program to continue to run at the fastest possible speed. It is only when the program moves on to need a different area of code and/or data that the contents of the cache are evicted to either the next level of cache (which is larger but slightly slower to access), or eventually, all the way back to the main off-CPU system RAM. The difficult aspect of implementing a cache memory is maintaining coherency, or ensuring that any attempt to access the "real" main RAM location when the cached copy has been changed results in a reconciliation that ensures that the most recently altered contents are what is actually returned, rather than out of date

data. We therefore conclude here by stating that virtual and cache memory offers numerous benefits that bring about an overall better performance of our computer systems. Every computer user has a need to seek an understanding of the techniques and mechanisms of the operations of virtual and cache memories as explained in this paper.

7. REFERENCES

[1] John Papiewski (2018). "The Concept of Virtual Memory in Computer Architecture. Available From www.smallbuisness.chron.com/Concept-Virtual-memory.

[2] Jacob Queen (2018). "Why is Virtual Memory Important?" Available fromwww.techwalla.com/articles/why-is-virtual-memory-important.

[3] Cicnavi (2010). "What is Virtual Memory and why do we need it?" Available fromwww.utilizewindow.com.

[4] John Papiewski (2018). "The Concept of Virtual Memory in Computer Architecture. Available From www.smallbuisness.chron.com/Concept-Virtual-memory.

[5] Santosh .S. Padwal, Ahishi . P. Duthate, Shivkumar Vishnupurikar and P.M. Chawman, (2012). "Cache Memory organization". International Journal of Networking and Parallel computing. Vol 1, issue 2, November 2012. Page 12-16.

[6] Margaret Rouse (2014). "Cache memory" Available from www.searchstorage.tectarget.com/definition/cache-memory.

[7] Howtoitz (2017). " What is Cache Memory? " Available from www.howtoitz.com. Retrieved on 4/4/18.

[8] Dinesh T, (2010) "Operating System". E-Computer Notes. Available from www.ecomputernotes.com. Retrieved on 5/4/18.

[9] IDC-Technologies. "Operating System Memory Mangement". Available from www.idc-online.com/technical -references/information-technologies/O/S-memory-mgt.

[10] Geeks for Geeks(n.d). "Page Replacement Algorithms". Available from www.geeksforgeeks.org/operating-system-page-replacement-algorithm.

[11] Operating Systems Study Guide (2015). "Page faults". Available from faculty.salima.k-state.edu.

[12] Juhi Kumari, Sonam Kumari and Devendra Prasad (2016). " A comparison of Page Replacement Algorithms: A survey". International Journal of Scientific & Engineering Research. Volume 7, issue 12, December 2016.

[13] Juhi Kumari, Sonam Kumari and Devendra Prasad (2016). " A comparison of Page Replacement Algorithms: A survey". International Journal of Scientific & Engineering Research. Volume 7, issue 12, December 2016.

[14] Priyanka Yadav, Vishal Sharma and Priti Yadav, (2014). "Cache Memeory; various algorithm". International Journal of computer science and mobile computing. Vol 3, issue 9, sept 2014, pp 838-840.

[15] Priyanka Yadav, Vishal Sharma and Priti Yadav, (2014). "Cache Memeory; various algorithm". International Journal of computer science and mobile computing. Vol 3, issue 9, sept 2014, pp 838-840.

[16] Santosh .S. Padwal, Ahishi . P. Duthate, Shivkumar Vishnupurikar and P.M. Chawman, (2012). "Cache Memory organization". International Journal of Networking and Parallel computing. Vol 1, issue 2, November 2012. Page 12-16.

[17] Jacob Queen (2018). "Why is Virtual Memory Important?" Available fromwww.techwalla.com/articles/why-is-virtual-memory-important.

[18] Carniegie Mellon University World Wide Knowledge Base . CMU school of computing Science. Available at http:// www.cs.cmu.edu. Retrieved on 23rd January, 2016.

[19] General Note (2018). " Cache Memory". Available from www.gegralnote.com/Basic- computer/cache-memory-php.

[20] IBM Knowledge Center (n.d). Advantage of Storage data caching". Available from www.ibm.com/support/knowledge-centre.