# Java Based Memory Latency Simulator

Pat Baumgardner, Adam Dachenhausen, Shah Syed
Department of Computer Science
Siena College
Loudonville, NY 12211

**Abstract**

The Java Based Memory Latency Simulator was developed to provide a way to study how long it takes to move bytes of data around a common computer system containing hard drive disks, random access memory, and a central processing unit with cache memory. This project aims first to create a modular system, that could be used to simulate data movement between computer components, then second to use that simulator and upgrade it to allow data to be collected about data latency. We first focused on creating data structures that represented various components of the computer, like a hard drive disk. Then we created a dual purpose structure that both holds these components and interacts with the user. For this project, we gathered data about the memory latency as an example of the modularity of the simulator. Our simulation does theoretically have actual delays, however, due to the limitations of Java, the delays are not as accurate as we would like them. We have found that memory management is a complex structure, that cannot be easily represented, but simpiler to approximate.

## 1 Overview

The goal of this project was to create a computer memory latency simulator. Java was chosen as the programming language for this project for its ease of use and it is what we, the authors are most comfortable with. This simulator is designed to simulate data transfer from one system component to another. For example, from the hard drive disk, to the main memory.

In section 2 we discuss abstractly how the memory latency simulator works, and why we built it. Section 3 describes how to build and run the simulator. In section 4 we explain all the types of information collected from the simulator. Then, in section 5 we present the actual data collected from the simulator, as well as actual real world data. Section 6 goes on to explain any discrepancies found between our data and the real world collected data. Finally, in section 7 we summarize our results, as well as report what we have learned, and suggest how this simulator could be better improved or used.

## 2 Simulating memory latency

### 2.1 Abstract Definition

The goal of this project was to create a Java based simulator that will allow access to artificial data so the user could gather statistics and trends from moving this data around the artificial system. The simulator allows simple operations on blocks of data, not individual bytes. The simulator consists of four primary parts: memSim, which is essentially the controller; a CPU with one level of cache; one or more sticks or RAM; and one or more hard drive disks.

Each component's (CPU, RAM, HDD), storage is represented with a simple array of a data structure called a dataBlock, which represents 1024 bytes. The user then has options to handle data on any component via the memSim terminal interface.

### 2.2 Primary Audience

Today, there are many levels of abstraction between the high-level user and their hardware. We wanted to give a way to collect data at this low level of hardware, but not have to be a electrical engineer working on the circuitry.

In our research, there is also very little public access on this type of data, so it could be useful for researchers and students alike to use this simulator to collect data for a variety of projects.

## 3 Building and Running the Simulator

### 3.1 Java

The simulator is written in Java, and therefore, you will need to have Java installed to run it. See `https://www.java.com/en/download/` for more.

## 3.2 Acquiring Source Code

To download the simulator code, go to `osfinal.dachenhausen.org` or `https://github.com/adamdachenhausen/cs330FinalProject` and clone the repository. See `https:docs.github.com/en/free-pro-team@latest/github/creating-cloning-and-archivi cloning-a-repository` for more.

## 3.3 Compiling

- **Command Line** The source code includes two ways to compile the simulator.
  - **Make** The simulator source code includes a Makefile, so if Make is installed, the command

    ```
    make
    ```

    will compile the simulator.
  - **Default** In the absence of Make, the default way to compile the simulator is to run

    ```
    javac *.java
    ```

    which will compile all of the Java files so they could be run.
- **Using an IDE** Given the multitude of IDEs that are available, please see your specific IDE's manual for how to compile and run the simulator.

## 3.4 Licensing

Before running the simulator, we would like to remind you that this project is protected under the MIT License, so proceed at your own risk.

## 3.5 Running the Simulator

- **Command Line** Either way that the simulator was compiled, the command

  ```
  java memSim
  ```

  will start the simulator

- **IDE** Given the multitude of IDEs that are available, please see your specific IDE's manual for how to compile and run the simulator.

Upon running the simulator, you will be prompted for a variety of simulator parameters. Each of these is crucial, and cannot be left blank. The first thing the simulator will ask you is:

```
Is this text blue? Y/n
```

If your terminal supports ANSI colored text, then the question should be blue, and you should respond Y. This helps point out important data. If your terminal does not support the colored text, then you will see something like

```
[34mIs this text blue? [0m Y/n
```

in which case, if you respond n, the simulator will not use any ANSI text and everything will print normally.

The simulator will then ask you whether you would like to LOAD or create a NEW system.

## Load An Existing System

The simulator will then show you three system specifications and a prompt, in which you should choose a system by entering its system number.

## Create A New System

If you opt to create a new system, you will be presented with the prompt shown below, which has sample data included. Note: very large numbers will make the simulator lag the Java virtual machine, so please be careful when inputting this data.

```
How many Hard Drives would you like?
1
And how big (in bytes) should each one be?
1024
How many RAM sticks would you like?
1
And how big (in bytes) should each one be?
512
How big (in bytes) would you like your CPU cache to be?
32
How long would you like it to take in seconds,
 to read/write to the cache?
0
How long would you like it to take in seconds,
 to read/write to the RAM?
5
Finally, how long would you like it to
take in seconds, to read/write to a HDD?
10

***********System Config**********
1 number of hard drives, each 1024 bytes big.
1 sticks of RAM, each 512 bytes big.
CPU cache is 32 bytes big.
Cache delay is: 0seconds
RAM delay is:   5seconds
HDD delay is:   10seconds
**********************************
```

After setting up the simulator, the components will each start up, and you will be prompted to choose an option from the menu. You can choose to either select based on the menu number, or the name.

```
Please enter the number of your selection: 1
What would you like to do next?
1.      Move Data
2.      Read Data
3.      Write Data
4.      Delete Data
5.      Help
6.      Exit
```

In all actions, the simulator starts its timing only when user input is done, so your data will not reflect your typing speed.

## Reading Data

Reading data is fairly simple, upon selecting read by entering its number into the prompt, the simulator will show the following menu:

```
Where would you like to read the data from?
1.      Hard Drive
2.      RAM
3.      CPU Cache
Please enter the number of your selection:
```

You should then enter the number corresponding to your selection. If you select a hard drive or a RAM stick, you will first get shown:

```
Please choose a drive from 1 - 2
```

Then, you will get shown that drive/stick's specific contents:

```
****Data on this hard drive****
0: 1024 bytes big
2: 1024 bytes big
3: 1024 bytes big
*******************************
```

You should then select the data that you wish to be read.

If you select the CPU cache, the entire contents of the cache will be read.

## Writing Data

Writing data is similar to reading data, as you will still be shown a prompt of where you would like to write, and which specific drive/stick's contents to write to. You will then be shown:

```
How many bytes would you like to write?
```

In which, you should enter the number of bytes to write. Note, if you enter a number greater than the space remaining on the drive/stick, then the write will fail.

If you select the CPU cache, there is no limiter (other than that of Java) on maximum data that can be written. The data structure is set up so that writing to cache is circular, so the last n bytes are the ones stored in the cache, where n is the size of the cache.

## Deleting Data

Deleting data is very similar to reading data, as they are essentially the same operation. In this case, every iteration through the stored data, the current index is set to null.

In the CPU cache, deleting data move the last added pointer back, so you can overwrite the data deleted.

### Moving Data

A move is a combination of the three aforementioned commands. First, the simulator will prompt like a read from where to read, and then delete the data from. Then the simulator will prompt for where this data should be written. At this point the simulator will also make sure that you do not try to read from one location, then write to that exact same location, ie. read from drive 1, chunk 0, then try to write to drive 1, chunk 0. The simulator will re-prompt until this is not the case.

## 4 Explanation of Data Gathered

The primary path of study through the simulator was to gather the statistics about the events, latencies and delays that take place in the very low level of memory management. Memory management as the name suggests is the act of managing memory in the low level of a computer system. Often represented by allocation of chunks of data and freeing them once they are done being used.

If we consider memory management as the super class, then the statistics that we gathered, such as the latency of memory regulation are the one of the aspects of the memory management. The latency is the delay or the time that it takes the data to be read from memory. Since the movement of this data is bound to be within the speed of light, and furthermore the medium it is traveling in, this means that even though that speed is above human physical comprehension but still there would be a delay, based on the size of the data.

We gathered the latency time, which is the time delay taking place in data transfer, reading and writing at the lowest level of memory management. This was the primary statistic for us to consider when building the simulator.

To represent this level of memory management the simulator was designed to hold some key components. These components are those linked directly with the memory usage, including the central processing unit (CPU), the Hard Drive Disk (HDD) and the Random Access Memory (RAM) sticks.

The CPU further consists of cache, modern day powerful CPUs consist of 3 level of caches, L1, L2, and L3. where L1 is very small compared to L2 in memory. L1 ranging from 2KB to 64KB, whereas the second level is in the range of 256 KB to 2 MB, but much slower than L1. L3 can be anywhere from 4MB to 50MB, but much slower than L2.[5] Cache is used for quicker access by the CPU of the data that it frequently uses. Cache stores that data and thus decreasing the latency. However, in the simulator we implemented a single level of cache with a variable size, which is input by the user.

Further down the hierarchy of memory management, we have the RAM, which is relatively slower to CPU but bigger in size by quite some margin. The RAM is a volatile form of memory built by several memory cells. There are two types of RAMs; DRAM and SRAM, although we did not distinguish between them in our simulator. Given that the RAM is slower compared to the CPU(cache) it takes time to read and move data off it. In the simulator we kept the size and the number of RAM sticks a variable, so the user inputs the value for each. Since the RAM is volatile it loses all the data it has on it, once it stops being powered. As soon as the power is turned on, the most important section of the operating system(OS) is read from the hard drive and put onto the RAM so some important functions can be accessed quickly instead of reading it off the hard drive which is slower. This OS data along with other make up a lot the RAM usage when the computer is running. Running other process lead to more writing and reading off the RAM which leads to different delays.

Finally, the hard drive, which is the biggest form of permanent memory but the slowest one. It is extremely large compared to both RAM and cache. Almost all the software including the OS is stored on the HDD. HDDs have evolved in many ways over the course of times, making them quicker, but they have still remained slower compared to RAM and CPU cache. In our simulator we have built an HDD on an array of dataBlocks, just like the RAM.

The movement, reading and writing of the data to any of the components was timed, which helped us to collect the major statistics we needed here. The latency which was going to be the main focus of the simulation was calculated through a stopwatch implemented inside our program. As we kept gathering data, the time was a projected ratio of the time that was supposed to take place on the basic level. Thus, the time gathered was reduced to the ratio fully suited for the level of details being dealt with in the simulator.

The memory, which was the core component of the simulator and the primary part of each component was represented by arrays of dataBlocks. As the simulation went on, the array in each of the components were modified depending upon the event taking place. This modification gave us the data for how the size of data acts proportional to the latency and the delay in the transfer, writing and reading of data. The size of data was one of the major field of the statistics being gathered. We studied upon the size of the data as how much of variance does it brings to the clock.

As the variables we had in the simulation were the number of cores in the CPU, the number of hard drives, the size of hard drive in bytes, the number of RAM sticks, the size of each of the RAM stick and the size of the CPU Cache, and the time it took to read or write one megabyte to each component, we kept a record of each combination. However, even with the CPU having a variable number of cores, for the path of study being conducted, we maintained it to be 1 throughout the study. The simulator was structured to take in a value for each of the variables, except the number of cores. This led to new fields of data for statistics being gathered. The size of cache in the CPU, which is a small form of memory and the quickest one out of RAM and HDD, was varied to yield a different result, which went into the into the table of statistics too. The same was done with the RAM, which relatively larger form of memory. We gathered how the data access and transfer on RAM vary in the aspect of Time as with different sizes, and how slower or faster was the case with the accessing, editing and transferring from and to the Hard drives.

The number of RAM sticks and the number of hard drive was also an important variable in deciding the latency, the access time and the writing time, given that increasing the number of these components meant more size but a different access time.

## 5 Data Gathered

Latency time in the all 3 major components being implemented in the simulator vary accordingly to their sizes, that becomes the reason why cache is the fastest between the three component types.

First, the hard drive latency time for the input and output of data as it is the largest of both memory size, and latency time. The latency of a hard drive is a combination of it RPM, Seek Time and Transfer time. In the simulator that we implemented, we did not go into details to a point that we offer the RPM, seek time and the transfer time as a variable to the user. However we implemented the HDD on data blocks instead of platters, which acted as partitions.

Rotational latency of a hard drive is how long does it take a platter to spin, as after all it is a mechanical component bounded by a certain speed. According to [4]

| RPMs | Rotational Latency(ms) |
|---|---|
| 5400 | 11 ms |
| 7200 | 8 ms |
| 10000 | 6 ms |
| 15000 | 4 ms |

Seek time the time it takes for the read-write head to move up and down the platter. The seek time itself is comprised of a series of phase times. The first being the acceleration phase, the read-write arm starts to move, then comes the coasting phase when the head is moving at full speed. This is followed by the deceleration phase where the read-write head slows down. Then finally comes the settling phase as the read write head stops right at the data it needs. The average seek time of hard drives of size 300 GB to 1TB ranged from an average time of 4ms to 9ms. (source: [1] )

The last part of total I/O latency for hard drive is the transfer time, which is the time it takes the data to be transfer from or to the read-write head. Compared to rotational latency and seek time, transfer time is insignificant, given that the mechanical movements take more time than data movement which is more of movement of energy packets.

After all this we come to an equation:

```
I/O Latency Time of HDD = Rotational Latency + Seek Time + Latency Time
```

In an extensive calculation done by Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau, two hard drives one Cheetah, and the other being Barracuda was compared for the I/O latency time.[1] The size of data being dealt with was 100 MB. However, another controller in the comparison was whether the data reading or writing was sequential or random. Sequential means that large chunks of data are just being read and written which are in sequence, whereas the random transfer just means random data was selected from a read issue somewhere on the disk, which is how the data for applications work. Note that the data we represented in the simulator was also sequential and not random, as the hard drive implemented was built of data blocks.

| HDD | Cheetah 15K.5 | Barracuda |
|---|---:|---:|
| Capacity | 300 GB | 1 TB |
| RPM | 15,000 | 7,200 |
| Average Seek | 4 ms | 9ms |
| Platters | 4 | 4 |
| T I/O Random | 6 ms | 13.2 ms |
| T I/O Sequential | 800 ms | 900 ms |

## In our simulation

RAM is a much smaller storage device compared to HDD and is volatile, but that makes it quicker than the HDD. The Latency of data movement in RAM is comprised primarily of CAS latency which translates to Column Address Strobe Latency. Since the RAM is built upon columns or cells of memory, CAS latency means how much time it would take the RAM to access the memory cells, read the data and make it available as an output. And again, in the simulation we were not able to completely configure the design of the RAM to this extent however we did implement a RAM on data blocks.

Over the course of time the latency of RAM has improved, but compared to the bandwidth and capacity it has been on the very low side. According to a research done by Kevin Chang from CMU, in his paper 'Understanding and Improving the Latency of D-RAM based Memory Systems'([3]) in the period of 1999 - 2017, the capacity of RAM has increased a gigantic 128 times, where as the Bandwidth of a RAM has increased a measly 20 times, however the improvement in latency had only a factor of 1.3x. We can safely assume that the latency of a RAM has been consistent compared to the latency of HDD or CPU over time.

Compared to the RAM, the gap between the CPU cache latency and the RAM latency has been growing at a high rate. As discussed previously the latency of a RAM has not had much improvement in the last 3 decades. Sangyeun Cho from the University of Pittsburg, states the improvements in RAM latency has been less than 10 times than it was in 1980. However on the other hand the CPU latency has improved to be 10000 times better in 2005 than it was in 1980.

When we talk about the CPU cache, we are dealing with the smallest form of major memory in the system yet the fastest one. As mentioned in the previous section, the CPU consists of caches to save the time to fetch frequent data again from the memory. Given that the modern CPUs have 3 levels of cache built in them, we were only able to implement a single level on our simulator. The CPU latency is comprised of many factors that depend upon how efficient is cache working. If the CPU cache does not have the data required than the CPU has to use the buses to fetch the data from the memory, compromising the bus bandwidth and timing in the process. These two different situations can yield a very different latency period.

As of 2012 these were some of the Latencies, that includes different operation on a low level and at the high level.(Source: [2])

| | |
|---|---|
| L1 cache reference | 0.5 ns |
| Branch mispredict | 5 ns |
| L2 cache reference | 7 ns |
| Mutex lock/unlock | 25 ns |
| Main memory reference | 100 ns |
| Compress 1k bytes with zippy | 3000 ns / 3 us |
| Send 1 KB over Gbps network | 10000 ns / 10 us |
| Read 4 KB randomly from memory | 150000 ns / 150 us |
| Read 1 MB sequentially from memory | 250000 ns / 250 us |
| Round trip within same data center | 500000 ns / 500 us |
| Read 1 MB sequentially from SSD | 1000 us / 1 ms |
| Disk Seek | 10000 us / 10 ms |
| Read 1 MB sequentially from the disk | 20000 us / 20 ms |

Not all of this information, especially concerning network is that closely relevant to our field of study, but we can certainly see the latencies that included reading, writing and referencing from the CPU caches, RAM and the hard drives had a huge difference in between them. The low level operations that took place within the CPU were the quickest of all, referencing caches and branching. Secondly referencing the RAM (7 ns) was slower than referencing the the caches (0.5 ns and 7 ns) but alot faster that going to a and reading from disk which was almost a 20 million times slower than referencing the L1 Cache and up to a hundred thousand times slower than referencing the RAM.

In the simulation where we computed the actual data, we came up with the following statistics:

Moving a data block from RAM to HDD takes 0 ms Moving a data block from HDD to RAM takes 1 ms Moving a data block from CPU cache to the RAM is 1 ms Moving a data block from Cache to HDD takes 0 ms Reading the HDD takes 10 ms Reading from the RAM is 2.7 s Reading from the cache is 0 ms Writing to the HDD is 1817 ms Writing to the RAM is 1328 ms Writing to Cache is 0

## 6 Discrepancies

The goal of the project was to find out about the varying latencies of some of the major components of the CPU, with or without the simulator. By implementing a simulator good enough to differentiate and manage data we were able to find out more about the memory management in an environment ran by threads. However, since there exists some complex and deep details and components in the real world units that we represented, we were only able to implement as much as the basic level of each component. The real world data that we represented made up most of our information in the paper, but that was along with the fact that the data we imported, was a combination of research and several different projects. Starting from the hard drive we did not represent the sectors, platters and other mechanical components, instead we implemented the HDD using data blocks, thus we were only able to gather stats for a sequential read and write. A similar implementation of RAM meant that we did not explicitly render the memory cells in the RAM but only the data blocks which act as those cells. For the CPU we only represented a single level of cache a L1, while almost all the modern CPUs work on dual level of cache. Being programmed in java our simulator was bounded by the high level barrier, so to represent the low level of memory management, we did have to implement delays ourselves. Our delays were then rationalized according to the time it takes in the low level. Due to the limitations of java, we ran into some issues with our delays that caused the RAM only to be accessed at a very longed latency, which was the only major error in our simulator A lot that was not gathered through the simulator, a deep analysis of sources did cover for that. Especially getting to know the latencies of new combinations of components that we did not implement in out simulator helped us to a more realistic approach. The research itself was based upon a latency simulator with a concentration on what makes up the delays in a real world programming system.

## 7 Conclusions

Memory Management within a memory system is a complex set of organizations of several different pieces, which themselves are a set of several different elements. So if we are to study the time delays as a superset of memory latencies, we cannot just discuss the general latencies of each component separately. But one must realize that the components are very closely linked with each other. An example of this would be that if a a CPU references either cache and it hits, that would be an ideal situation for the a real model, but if does not find what it needs then it has to ref the memory or eventually the Disks, resulting in so many different factors for delays, including but not only, the CAS latency of the RAM, the HDD I/O latency, which further includes the delays of rotation, read write head, settling time etc. We found out that after the research that our simulator which based upon a simple memory system, could not keep up with a modern memory system, which included tonnes of elements to care about.

## References

[1] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. CreateSpace Independent Publishing Platform, 2008.

[2] J. Boner. latency comparison numbers in 2012. Github, 2012.

[3] K. K. Chang. Understanding and improving the latency of dram-based memory systems, May 2017.

[4] E. Shanks. Disk latency concepts. The IT Hollow, 2013.

[5] P. Volvoikar. How does cpu cache work and what are l1, l2, and l3? Make Use Of.