**Nova Southeastern University**

**College of Engineering and Computing**

**CISC 665 Distributed Systems**

**Winter 2017**

**(January 9– May 7, 2017)**

**Assignment #3**

Due Date: May 1, 2017

Instructor: Dr. Gregory Simco

Distributed Systems

Assignment Three

Final Paper

Bryce Tucker

Student #: N01530887

Table of Contents

[Chapter 1: Introduction 4](#_Toc481424068)

[Problem Statement 4](#_Toc481424069)

[Goal Statement 4](#_Toc481424070)

[Chapter 2: Review of Literature 5](#_Toc481424071)

[Purpose 5](#_Toc481424072)

[Cache architecture 5](#_Toc481424073)

[Summary 6](#_Toc481424074)

[Chapter 3: Methodology 7](#_Toc481424075)

[Overview 7](#_Toc481424076)

[System Implementation 8](#_Toc481424077)

[Clients 9](#_Toc481424078)

[Document Server 10](#_Toc481424079)

[Data Layer 10](#_Toc481424080)

[Cache System 12](#_Toc481424081)

[Test Harness 13](#_Toc481424082)

[Test procedures 13](#_Toc481424083)

[Chapter 4: Results 14](#_Toc481424084)

[Data analysis: 14](#_Toc481424085)

[Summary of results: 16](#_Toc481424086)

[Conclusions 16](#_Toc481424087)

[Recommendations: 17](#_Toc481424088)

[References: 18](#_Toc481424089)

[Appendixes: 18](#_Toc481424090)

# Chapter 1: Introduction

This study provides an exploration of the same line of design reasoning as that described in the paper by Nelson [2006] *Concurrent Caching*, the effort is needed to examine an alternative implementation to verify similar results can be obtained. This study applies the concept of concurrent cache to an example application and explores the benefits and issues involved with implementing the architecture using the Java programming language.

## Problem Statement

The problem for this assignment is to explore and evaluate an implementation of a document vending application that uses a caching method of separate processes for each file being served to clients. The primary impetus for the prototype developed comes from Nelson [2006] where the idea of using processes to provide caching for individual datum is explored. The concept was then applied to the document vending scenario to create a testable prototype.

The requirements for the document vending application included accounting for costs and the ability to upload files to be available for distribution. Due to time constraints, the requirements were pared down to the basic operations of transferring the requested files to clients. A significant amount of effort was put into developing a test harness that managed the testing of the prototype using multiple clients in separate processes performing file requests.

## Goal Statement

The goal is to examine the effort involved in the design and implementation of a file server that uses the model of having separate processes provide caching of individual files. The examination will include measurements of performance differences between an implementation of the file server that does not use any caching and the prototype that uses the caching method under examination. This comparison will provide some insight into the cost and benefits of the caching method.

The measurements of performance of the document server will be the duration of time in milliseconds between when the client first requests the file and when the client fully receives the file.

# Chapter 2: Review of Literature

## Purpose

The review of literature pertaining to the implementation of a document vending application, in specific the implementation of a distributed file server with a cache was conducted. The purpose of the review was to establish a foundation of the current state of investigation into the type of implementation pertinent to the stated requirements of this assignment. From this literature review on document vending applications and cache methodologies, the focus became the paper by Nelson [2006] *Concurrent Caching*.

## Cache architecture

The paper by Nelson [2006] describes a cache implementation that uses separate processes to contain cached data. The paper describes advantages that include:

* Each datum can respond actively, independently and in parallel to requests
* The total size of cached data may exceed the memory of a single compute node
* An idle datum may be easily removed by terminating its process
* Stale data may be replaced in parallel when multiple changes to the data store occur
* Replacement policies may be implemented independently from the cache itself
* A cached datum may dynamically migrate to the CPU that best responds to requests

From this paper, the implementation of the solution for this given assignment includes the use of processes to perform the caching requirement. By following the described implementation, the cache manager will spawn processes to cache documents instead of maintaining the documents in a cache itself. When a document request is made, the cache manager will examine a maintained list to determine if the requested document is cached in a process, the IP address/port number will then be provided so the client and the given process can interact directly for the client to download the document.  
Cache replacements will be performed by terminating the process and spawning a replacement process to contain a given document. The cache manager will control the number of cache processes based on a configuration value to meet the requirement that the cache not be infinite. The cache manager will control the replacement of cached documents by tracking the number of accesses the file has received, when a cache process needs to be terminated to allow a new cache process to launch, the cache of the file with the fewest accesses will be terminated.

The implementation of a prototype as described in Nelson [2006] was accomplished using Erlang. The implementation for this effort will be in Java.

## Summary

The review of literature applicable to this assignment of implementing a document vending application, guided the focusing of the problem statement and goals to be on the testing of an implementation of the use of processes to implement a caching method where each datum is cached in a separate process.

# Chapter 3: Methodology

## Overview

The first part of the approach was to establish a test harness that would control a stable testing environment, with repeatable inputs to the system under test. The test harness launches the Document Server (the implementation of the file server) and then creates processes that are clients of the Document Server and requests files from the server. The test harness reads a scenario file that lists out the files being requested by the clients in order. This allows the tests to be consistent in that the same files are requested each time, though it does not guarantee that the messaging will be consistent and that the files will be requested in the same order each time. The client processes may not establish their communications with the document server in a consistent order as the system the tests are running on may introduce delays.

The test harness also implements a logging function that receives log messages via IP sockets from each client process. Each client process will report events with timestamps so the operations of the system can be measured.

The test harness launches both the client processes and the document server process. The document server process launches a process to provide a data layer. The data layer accesses the files that will be served to the clients. The document server establishes a server that listens for connection requests from the clients. When a connection is established, a separate thread is launched to handle the request. Each client performs a connection and request to simulate obtaining a list of available files from the document server, there is then a pause of a random number of milliseconds before the client will request the desired file.

In the case of the prototype that does not have a cache, the file request to the document server results in the port number of the data layer server port that is hard-coded. The client then closes the connection to the document server and opens a connection with the data layer to request the file. The data layer then retrieves the file from disk and transfers the file to the client. Once the client receives the file, the client reports the event to the log server and terminates.

In the case of the prototype with the cache implementation, when the client requests the file from the document server, the list of files is read through to determine if the file has been cached in a cache process. If the file is in a cache, then the IP address and port number of the cache process is sent to the client. If the file is not in a cache, then the document server determines if a cache process needs to be terminated based on the number of cache processes currently operating. If there is space available for a new cache to begin, then a new cache process is launched with the filename of the file to cache. The document server must wait for the cache process to start and send back via an IP connection the port number that cache process has been assigned by the system. The document server can then send the port number to the client so the client can communicate with the cache process to receive the desired file.

## System Implementation

This prototype implementation was accomplished in the Java programming language without using third party libraries. Each part of the system communicates using TCP sockets available in Java, processes are launched using the ProcessBuilder class and several parts of the system implemented multiple threads of execution to handle TCP communications.

The overriding part of the prototype system is the test harness that is executed first and that causes the document server prototype to execute and launches client processes to interact with the document server. The entirety of the system is executed on the same machine, therefore there were certain limitations in the execution of the testing due to resource availability.

The test harness executes and reads from a scenario file that consists of a certain number of lines where each line contains a filename (that corresponds with a file in the document server) and a delay period. Both values are randomly selected using a scenario builder application. The list of available files and the scope of the randomized delay period can be set in the code for the scenario builder. The scenario builder then executes and produces a scenario file. This file can then be used to stimulate the testing of the prototype repeatedly and can be used to stimulate different prototype implementations. This consistency of stimulus can help to limit the differences between tests.

### Clients

Clients were implemented to be executed as separate processes and log into the document server via a TCP socket using a hard-coded IP address and port number (The IP address used by all the system components for their TCP sockets was ‘localhost’). When the test harness executes the client process, two values are included as arguments: filename and delay period. The filename is the name of the file the client will download from the document server. The delay period is a value used to simulate a user of the client taking time to decide on a file to download.

### Document Server

The document server is the main execution component of the prototype. For the purposes of testing, two prototypes were implemented, one prototype implements cache processes and the other does not, instead allowing direct communication from clients to the data layer to obtain the desired files. This difference allows for a comparison between the amount of time taken for the clients to receive files with or without the prototype cache system.

In both prototypes, the data layer is implemented in the same way and is executed as a separate process from the document server. The document server maintains a list of files stored on disk that the data layer retrieves and serves to clients. In the case of the prototype that does not do caching, the document server always provides the IP address and port number of the data layer to clients when they request a file from the document server. In the case of the prototype with caching and for the purposes of testing, the list of files is hard-coded into the document server, but the server creates a dynamic list in the form of a linked list from the hard-coded list of file names. The dynamic list maintains the data of which files are currently in a cache process and the number of times that filename is requested by clients.

In both cases, the timing of the request for a file by a client starts when the client sends a filename to the document server. The document server and the data layer both have hard coded port numbers to facilitate communication.

### Data Layer

The data layer is a separate application that operates to store the documents and provide those documents upon request. The data layer uses a process with multiple threads to handle the communication. The communication is conducted with TCP Sockets in Java.

In the case of the prototype with no caching, the clients are given the data layer’s IP address and port number and will establish communication with the data layer directly. In the case of the prototype that does have caching, the cache processes are launched by the document server and those processes request the desired file from the data layer. The data layer then transmits the file to the cache process where the file data is kept in active memory.

In both cases of requests for files, the data layer retrieves the file from disk and then transmits the file to the requester. The TCP transmission size available on the Java socket using the writeUTF method has a maximum size limit of around 65K bytes. Therefore, the files must be broken up before transmission. The requesting process will send a signal that a file is being requests, the data layer responds with a request for the filename, the requesting process sends the filename, then the data layer will get the file from disk, analyze it and break it into parts and will send the number of parts and the total file size to the requesting process. When the requesting process receives the file size and number of parts being transmitted, it prepares to receive the file.

In the case of the prototype without caching, the client is the one receiving the file and will prepare to receive and reconstruct the file into a string/text file. Once the file is received completely and reconstructed, the client logs the fact that the file has been received. This is the point when the timing of the request is finished in both prototypes. In the case of the prototype with caching, the cache process receives the information about the file being transmitted from the data layer and prepares to receive the file parts. The cache process does not reconstruct the file, instead the parts are kept separated in an array so when a request from a client is received, the data can immediately be transferred. The client operates the same in either prototype case and receives and reconstructs the file.

## Cache System

The cache implementation using processes to cache files being downloaded is the predominant area of investigation for this study. The cache is implemented as separate processes that are launched by the document server when the client requests a file. The document server manages the cache by maintaining a list of available files and if the file is in a cache, then the file information includes the IP address and port number of the cache process for that file. The number of cache processes the system can have active at one time is set in the code to a certain value and the document server also maintains a count of how many cache processes are active at any time.

When a file is requested, if the file is in cache, then the number of cache processes is checked against the maximum allowed processes. If there is room for more processes, then a new process is launched for a cache and the requested filename is sent as an argument to the process. If there is not room for another cache, then an existing cache must be terminated first. The cache to terminate is determined by traversing the linked list of files and identifying the file that currently has a cache process and has the lowest number of requests for the file. This cache replacement policy is unsophisticated but is used to accomplish the basic operation of the caching system.

Cache processes that are identified to be terminated are sent a termination signal via their TCP socket from the document server. The document server must then use the waitFor method on the process variable to receive the signal that the process has terminated. The whole operation of reading and changing the list of files, starting and terminating cache processes and updating the file list with access count values and the TCP address and port number of a new cache process is conducted during a lock of a MUTEX in the document server.

The benefit of simply terminating a process to clear out a cache is accomplished, but there is potential for a bottleneck that could last for a significant amount of time if the cache being terminated has already received multiple requests for the file and must complete those transmissions before terminating. The locking is necessary because of the potential that file transmissions could be occurring when changes to the cache process list is taking place. Therefore; the described benefit of concurrent starts and terminations of cache processes has not been achieved.

## Test Harness

The test harness is used to perform tests of the prototypes and apply the same input to both prototypes and for each test. The test harness reads a scenario file to determine what clients to start and when. Due to limitations of the workstation being used and the Java virtual machine settings, the tests of the prototypes were limited to about 80 clients. Greater than that number of clients in the scenario resulted in the JVM running out of memory or not being able to perform garbage collection.

To address that issue, a randomization of the starting times of the client processes was implemented to keep the test harness from trying to start all the client processes at the same time. With this in place, the simulation could hold 150 clients being executed by the test harness. Only around 20 client processes were active at the same time. As each test is performed, the activities of the client are logged and are written to a file by the test harness.

### Test procedures

Testing was conducted of the same scenario of 150 clients on the prototype with no cache five times. The same scenario was also conducted on the prototype with caching where the maximum number of cache processes was six, then 12, then 20 each test conducted five times. The total number of files in the system for testing purposes is 30, so the ratio of cached processes to total number of files got quite high.

After each test, the log file is analyzed by a separate program and converted from all the log entries of the client processes to a list of clients, the filename they requested, and the duration of time between first requesting the document from the document server and receiving the document.

# Chapter 4: Results

The efforts of this study resulted in the construction in the Java language of a document server that uses separate processes to cache files for delivery to clients. Collecting metrics on the time it takes for the prototype to deliver files to clients showed that there is a significant increase in time as compared to an implementation of the document server without using the caching method. However, the significant increase in time appears to be caused by one bottleneck in the implementation that could be improved, the cache management implementation.

## Data analysis:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Totals** | **Run 1** | **Run 2** | **Run 3** | **Run 4** | **Run 5** | **Total Average** |
| No cache | 6510 | 6148 | 7398 | 6505 | 7017 | 6716 |
| 6 cache | 246699 | 348504 | 256339 | 256857 | 177934 | 257267 |
| 12 cache | 175550 | 235152 | 138359 | 224376 | 185116 | 191711 |
| 20 cache | 129367 | 69517 | 66208 | 61922 | 106606 | 86724 |

This table represents the total durations of the 150 client file requests to the prototypes. The first row of values is from the prototype with no caching. The subsequent rows are results from running the same scenario list of clients with the prototype with caching where the maximum number of cache processes is six, 12, and 20 respectively.

The durations are much greater for the prototype with cache processes. The combination of performing searches for files in the file list and starting or stopping and starting cache processes adds significant time to the retrieval of files.

The following tables show the execution of a scenario where there are just 10 clients, each one requesting the same file that is around one megabyte in size. The first table is the durations of the prototype system with no caching and the second one is with caching.

|  |  |  |
| --- | --- | --- |
| Client 0 | file1m5.txt | 120 |
| Client 2 | file1m5.txt | 198 |
| Client 1 | file1m5.txt | 99 |
| Client 4 | file1m5.txt | 92 |
| Client 3 | file1m5.txt | 115 |
| Client 6 | file1m5.txt | 102 |
| Client 7 | file1m5.txt | 74 |
| Client 8 | file1m5.txt | 94 |
| Client 5 | file1m5.txt | 75 |
| Client 9 | file1m5.txt | 93 |
|  |  | 1062 |

|  |  |  |
| --- | --- | --- |
| Client 0 | file1m5.txt | 321 |
| Client 1 | file1m5.txt | 39 |
| Client 5 | file1m5.txt | 51 |
| Client 3 | file1m5.txt | 35 |
| Client 4 | file1m5.txt | 48 |
| Client 2 | file1m5.txt | 39 |
| Client 6 | file1m5.txt | 28 |
| Client 8 | file1m5.txt | 38 |
| Client 7 | file1m5.txt | 48 |
| Client 9 | file1m5.txt | 54 |
|  |  | 701 |

The no cache prototype has a total of 1,062 ms while the prototype with caching has a total of 701 ms. This demonstrates that caching the file, even caching it in a separate process can improve the retrieval speed in a certain condition. The overhead costs of searching the file list for a match, launching a new process and allowing time for the process to start, retrieve the desired file from the data layer and start a TCP server listener socket and report the socket port number takes longer than a direct request and file access from the disk. However, subsequent requests that find the file is already in a cache process can return much quicker and since the cache process does not have to perform any disk accesses, the return of the file can be faster.

Unfortunately, the implementation of the caching architecture creates a significant bottleneck in the cache management area of execution. Though the document server creates threads to handle multiple requests from clients at nearly the same time, the current mutex lock occurs when starting a read of the list of files and does not end until an address and port number is ready to be returned to the client. This causes a significant amount of blocking, especially during the long period when a process must be terminated and a new process started. This scenario can be especially long if the cache process is currently handling file transfers and the blocking must continue until the file transfers complete and the process then terminates.

## Summary of results:

Overall the results show that the caching architecture adds overhead to the file retrieval time even though a test of the caching mechanism when the same file is requested multiple times in a row shows the prototype can produce lower response times than the no cache prototype. The cache management process, as it is implemented in this prototype, adds a bottleneck due to blocking of multithreaded execution at the point where every client must request information.

# Conclusions

Based on the timing data collected from the tests and evaluation of the causes for the time increases, it appears that the prototype implementation of this cache architecture has limited effectiveness due to the bottleneck caused by the approach used to handle cache management operations. The paper by Nelson [2006] lists the parallelization of the cache processes management as a key potential benefit and the parallelization of cache process activation and termination as a key benefit. In this Java prototype implementation, the parallelization of process activation and termination was not accomplished and neither was the parallelization of cache management operations. Those two areas are likely to be the main cause for the much higher retrieval times between the caching prototype and the no cache prototype.

## Recommendations:

Care must be taken when implementing a concurrent caching architecture especially cache management. Next steps should include revamping the prototype’s cache management implementation. The reading of the file list should be made to be parallelized so reads can continue while new cache processes are being launched. This will mean only those file requests that need the same file will block while another file list read is occurring or while a new cache process is being started.

To gain a slightly more accurate comparison between the no cache prototype and the caching prototype, a file list should be implemented in the no cache prototype to require the searching and updating of access counts in the file list. The difference in operations between the two prototypes will then be reduced to the management of cache processes, the starting and stopping of cache processes.

The final recommendation for this study would be to implement a better cache replacement strategy. The current strategy is based on the total accesses. Instead the cache processes to be terminated may be based on something else such as a time to live value that is reset when file requests are received or based on the size of file stored in the cache, where larger files may be more likely to stay in cache as they gain the most benefit from being cached.

# References:

Nelson, J. (2006). Concurrent Caching. *Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, 32-38. doi: <https://doi.org/10.1145/1159789.1159797>

# Appendixes:

Results from measuring the time between a client requesting a file and fully receiving that file.

Table 1: Five executions of the same scenario executed with the prototype with no caching:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | filename | Durations in ms. Executions 1 to 5 | | | | |  |
| Client 27 | file110k1.txt | 18 | 10 | 36 | 11 | 34 |  |
| Client 79 | file110k1.txt | 33 | 33 | 13 | 34 | 28 |  |
| Client 86 | file110k1.txt | 9 | 35 | 13 | 33 | 9 |  |
| Client 124 | file110k2.txt | 9 | 31 | 21 | 29 | 23 |  |
| Client 104 | file110k3.txt | 28 | 13 | 35 | 33 | 29 |  |
| Client 49 | file110k3.txt | 33 | 33 | 32 | 28 | 29 |  |
| Client 92 | file110k3.txt | 17 | 14 | 45 | 29 | 26 |  |
| Client 142 | file110k4.txt | 33 | 34 | 32 | 32 | 27 |  |
| Client 78 | file110k4.txt | 35 | 27 | 33 | 27 | 33 |  |
| Client 120 | file110k5.txt | 33 | 33 | 9 | 34 | 51 |  |
| Client 61 | file110k5.txt | 22 | 31 | 28 | 27 | 12 |  |
| Client 107 | file130k1.txt | 34 | 32 | 15 | 37 | 36 |  |
| Client 114 | file130k1.txt | 31 | 36 | 37 | 30 | 36 |  |
| Client 119 | file130k1.txt | 31 | 34 | 37 | 10 | 36 |  |
| Client 13 | file130k1.txt | 36 | 22 | 30 | 33 | 35 |  |
| Client 131 | file130k1.txt | 31 | 30 | 30 | 37 | 16 |  |
| Client 14 | file130k1.txt | 12 | 35 | 39 | 21 | 31 |  |
| Client 41 | file130k1.txt | 12 | 9 | 13 | 21 | 35 |  |
| Client 60 | file130k1.txt | 18 | 35 | 33 | 10 | 13 |  |
| Client 84 | file130k1.txt | 46 | 9 | 37 | 30 | 36 |  |
| Client 122 | file130k2.txt | 36 | 32 | 33 | 38 | 37 |  |
| Client 134 | file130k2.txt | 9 | 45 | 31 | 32 | 37 |  |
| Client 15 | file130k2.txt | 9 | 38 | 29 | 17 | 16 |  |
| Client 35 | file130k2.txt | 10 | 37 | 38 | 31 | 37 |  |
| Client 37 | file130k2.txt | 39 | 10 | 31 | 16 | 32 |  |
| Client 0 | file130k3.txt | 61 | 18 | 43 | 18 | 35 |  |
| Client 116 | file130k3.txt | 10 | 35 | 36 | 32 | 32 |  |
| Client 33 | file130k3.txt | 13 | 33 | 37 | 11 | 32 |  |
| Client 48 | file130k3.txt | 40 | 32 | 38 | 32 | 37 |  |
| Client 126 | file130k4.txt | 38 | 40 | 38 | 35 | 14 |  |
| Client 140 | file130k4.txt | 37 | 37 | 96 | 36 | 36 |  |
| Client 26 | file130k4.txt | 35 | 10 | 32 | 23 | 35 |  |
| Client 45 | file130k4.txt | 37 | 32 | 31 | 37 | 37 |  |
| Client 52 | file130k4.txt | 30 | 32 | 36 | 23 | 22 |  |
| Client 90 | file130k4.txt | 11 | 37 | 38 | 12 | 37 |  |
| Client 18 | file130k5.txt | 11 | 11 | 33 | 28 | 15 |  |
| Client 40 | file130k5.txt | 14 | 10 | 22 | 14 | 33 |  |
| Client 73 | file130k5.txt | 37 | 22 | 21 | 34 | 41 |  |
| Client 80 | file130k5.txt | 36 | 38 | 38 | 11 | 33 |  |
| Client 85 | file130k5.txt | 41 | 40 | 18 | 37 | 34 |  |
| Client 71 | file1m1.txt | 74 | 112 | 93 | 56 | 62 |  |
| Client 91 | file1m1.txt | 91 | 54 | 100 | 71 | 89 |  |
| Client 32 | file1m2.txt | 98 | 59 | 87 | 89 | 109 |  |
| Client 72 | file1m2.txt | 89 | 56 | 94 | 107 | 73 |  |
| Client 74 | file1m2.txt | 81 | 98 | 91 | 95 | 88 |  |
| Client 10 | file1m3.txt | 110 | 80 | 106 | 187 | 99 |  |
| Client 118 | file1m3.txt | 101 | 93 | 55 | 99 | 108 |  |
| Client 137 | file1m3.txt | 77 | 95 | 175 | 96 | 86 |  |
| Client 146 | file1m3.txt | 88 | 82 | 89 | 88 | 93 |  |
| Client 25 | file1m3.txt | 102 | 57 | 67 | 93 | 88 |  |
| Client 70 | file1m3.txt | 113 | 56 | 93 | 95 | 88 |  |
| Client 83 | file1m3.txt | 52 | 72 | 87 | 82 | 89 |  |
| Client 11 | file1m4.txt | 107 | 56 | 89 | 95 | 97 |  |
| Client 16 | file1m4.txt | 58 | 58 | 94 | 72 | 90 |  |
| Client 82 | file1m4.txt | 66 | 56 | 89 | 91 | 89 |  |
| Client 102 | file1m5.txt | 102 | 90 | 89 | 92 | 102 |  |
| Client 113 | file1m5.txt | 98 | 89 | 103 | 54 | 98 |  |
| Client 59 | file1m5.txt | 106 | 93 | 90 | 87 | 93 |  |
| Client 6 | file1m5.txt | 99 | 73 | 113 | 103 | 91 |  |
| Client 94 | file1m5.txt | 95 | 75 | 112 | 93 | 55 |  |
| Client 2 | file200k1.txt | 48 | 39 | 57 | 20 | 80 |  |
| Client 22 | file200k1.txt | 41 | 16 | 60 | 13 | 45 |  |
| Client 44 | file200k1.txt | 45 | 39 | 42 | 24 | 44 |  |
| Client 62 | file200k1.txt | 41 | 40 | 50 | 44 | 37 |  |
| Client 77 | file200k1.txt | 13 | 43 | 46 | 42 | 41 |  |
| Client 88 | file200k1.txt | 19 | 21 | 44 | 48 | 25 |  |
| Client 115 | file200k2.txt | 25 | 36 | 34 | 40 | 13 |  |
| Client 117 | file200k2.txt | 49 | 43 | 39 | 37 | 18 |  |
| Client 148 | file200k2.txt | 51 | 46 | 43 | 39 | 38 |  |
| Client 101 | file200k3.txt | 46 | 30 | 47 | 45 | 45 |  |
| Client 127 | file200k3.txt | 13 | 47 | 40 | 39 | 46 |  |
| Client 47 | file200k3.txt | 42 | 40 | 42 | 39 | 39 |  |
| Client 99 | file200k3.txt | 41 | 39 | 44 | 24 | 52 |  |
| Client 100 | file200k4.txt | 39 | 37 | 45 | 27 | 45 |  |
| Client 42 | file200k4.txt | 25 | 13 | 51 | 46 | 40 |  |
| Client 65 | file200k4.txt | 39 | 14 | 47 | 52 | 14 |  |
| Client 129 | file200k5.txt | 42 | 35 | 37 | 44 | 25 |  |
| Client 132 | file200k5.txt | 39 | 50 | 26 | 25 | 21 |  |
| Client 66 | file200k5.txt | 39 | 14 | 14 | 43 | 38 |  |
| Client 81 | file200k5.txt | 39 | 46 | 37 | 39 | 38 |  |
| Client 125 | file350k1.txt | 54 | 50 | 55 | 21 | 31 |  |
| Client 28 | file350k1.txt | 21 | 47 | 63 | 35 | 54 |  |
| Client 53 | file350k1.txt | 57 | 58 | 63 | 58 | 60 |  |
| Client 96 | file350k1.txt | 22 | 34 | 57 | 21 | 21 |  |
| Client 138 | file350k2.txt | 54 | 53 | 51 | 27 | 59 |  |
| Client 149 | file350k2.txt | 49 | 32 | 59 | 38 | 55 |  |
| Client 43 | file350k2.txt | 30 | 23 | 55 | 59 | 53 |  |
| Client 57 | file350k2.txt | 21 | 55 | 66 | 53 | 55 |  |
| Client 58 | file350k2.txt | 42 | 35 | 22 | 54 | 54 |  |
| Client 93 | file350k2.txt | 55 | 83 | 43 | 66 | 58 |  |
| Client 54 | file350k3.txt | 57 | 51 | 21 | 41 | 55 |  |
| Client 55 | file350k3.txt | 60 | 60 | 56 | 32 | 59 |  |
| Client 9 | file350k3.txt | 55 | 28 | 55 | 31 | 95 |  |
| Client 1 | file350k4.txt | 89 | 33 | 54 | 55 | 24 |  |
| Client 112 | file350k4.txt | 62 | 57 | 27 | 52 | 31 |  |
| Client 135 | file350k4.txt | 52 | 54 | 48 | 58 | 21 |  |
| Client 20 | file350k4.txt | 32 | 58 | 56 | 26 | 52 |  |
| Client 21 | file350k4.txt | 55 | 21 | 55 | 56 | 21 |  |
| Client 29 | file350k4.txt | 22 | 32 | 53 | 57 | 58 |  |
| Client 36 | file350k4.txt | 54 | 23 | 24 | 53 | 60 |  |
| Client 50 | file350k4.txt | 36 | 54 | 60 | 30 | 55 |  |
| Client 8 | file350k4.txt | 39 | 39 | 154 | 55 | 67 |  |
| Client 123 | file350k5.txt | 35 | 58 | 54 | 63 | 66 |  |
| Client 128 | file350k5.txt | 37 | 61 | 22 | 38 | 56 |  |
| Client 38 | file350k5.txt | 67 | 23 | 62 | 57 | 44 |  |
| Client 46 | file350k5.txt | 49 | 52 | 68 | 21 | 54 |  |
| Client 69 | file350k5.txt | 55 | 61 | 53 | 26 | 63 |  |
| Client 12 | file500k1.txt | 73 | 35 | 67 | 72 | 82 |  |
| Client 145 | file500k1.txt | 61 | 68 | 81 | 71 | 70 |  |
| Client 39 | file500k1.txt | 33 | 65 | 68 | 68 | 46 |  |
| Client 133 | file500k2.txt | 65 | 62 | 70 | 65 | 28 |  |
| Client 3 | file500k2.txt | 76 | 52 | 67 | 66 | 61 |  |
| Client 34 | file500k2.txt | 35 | 47 | 61 | 64 | 68 |  |
| Client 108 | file500k3.txt | 60 | 74 | 73 | 65 | 67 |  |
| Client 121 | file500k3.txt | 72 | 68 | 70 | 35 | 62 |  |
| Client 141 | file500k3.txt | 28 | 65 | 66 | 76 | 71 |  |
| Client 68 | file500k3.txt | 63 | 29 | 80 | 42 | 70 |  |
| Client 95 | file500k3.txt | 86 | 51 | 67 | 82 | 73 |  |
| Client 136 | file500k4.txt | 48 | 64 | 80 | 42 | 77 |  |
| Client 144 | file500k4.txt | 70 | 71 | 67 | 69 | 75 |  |
| Client 5 | file500k4.txt | 69 | 49 | 67 | 79 | 74 |  |
| Client 130 | file500k5.txt | 29 | 66 | 67 | 61 | 65 |  |
| Client 56 | file500k5.txt | 76 | 61 | 68 | 72 | 75 |  |
| Client 87 | file500k5.txt | 71 | 72 | 71 | 66 | 60 |  |
| Client 89 | file500k5.txt | 67 | 66 | 61 | 61 | 66 |  |
| Client 106 | file75k1.txt | 25 | 18 | 28 | 17 | 26 |  |
| Client 147 | file75k1.txt | 26 | 7 | 18 | 28 | 27 |  |
| Client 4 | file75k1.txt | 28 | 54 | 38 | 29 | 31 |  |
| Client 109 | file75k2.txt | 25 | 24 | 28 | 26 | 18 |  |
| Client 111 | file75k2.txt | 27 | 27 | 26 | 27 | 26 |  |
| Client 139 | file75k2.txt | 29 | 11 | 25 | 27 | 27 |  |
| Client 17 | file75k2.txt | 8 | 7 | 23 | 17 | 8 |  |
| Client 23 | file75k2.txt | 7 | 7 | 20 | 24 | 25 |  |
| Client 31 | file75k2.txt | 9 | 8 | 26 | 12 | 26 |  |
| Client 63 | file75k2.txt | 27 | 26 | 12 | 26 | 28 |  |
| Client 76 | file75k2.txt | 27 | 26 | 18 | 27 | 26 |  |
| Client 67 | file75k3.txt | 20 | 27 | 24 | 7 | 7 |  |
| Client 143 | file75k4.txt | 27 | 26 | 26 | 23 | 27 |  |
| Client 19 | file75k4.txt | 30 | 9 | 26 | 13 | 27 |  |
| Client 30 | file75k4.txt | 27 | 20 | 11 | 15 | 26 |  |
| Client 51 | file75k4.txt | 26 | 28 | 16 | 22 | 26 |  |
| Client 98 | file75k4.txt | 7 | 7 | 26 | 26 | 26 |  |
| Client 103 | file75k5.txt | 23 | 25 | 22 | 26 | 27 |  |
| Client 105 | file75k5.txt | 23 | 28 | 8 | 19 | 26 |  |
| Client 110 | file75k5.txt | 26 | 25 | 18 | 26 | 26 |  |
| Client 24 | file75k5.txt | 9 | 28 | 26 | 14 | 24 |  |
| Client 64 | file75k5.txt | 27 | 31 | 29 | 11 | 31 |  |
| Client 7 | file75k5.txt | 31 | 11 | 28 | 18 | 113 |  |
| Client 75 | file75k5.txt | 22 | 25 | 27 | 26 | 26 |  |
| Client 97 | file75k5.txt | 13 | 26 | 94 | 22 | 28 | Average |
|  |  | 6510 | 6148 | 7398 | 6505 | 7017 | 6716 |

Table 2: Five executions of the same scenario executed with the prototype with a maximum of six cache processes:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Filename | Durations in ms, for executions 1 to 5 | | | | |  |
| Client 27 | file110k1.txt | 723 | 5024 | 677 | 1659 | 1505 |  |
| Client 79 | file110k1.txt | 2227 | 819 | 695 | 1491 | 1382 |  |
| Client 86 | file110k1.txt | 858 | 984 | 882 | 368 | 858 |  |
| Client 124 | file110k2.txt | 1991 | 5710 | 4476 | 1932 | 1304 |  |
| Client 104 | file110k3.txt | 631 | 1667 | 714 | 2433 | 696 |  |
| Client 49 | file110k3.txt | 2931 | 690 | 2131 | 5292 | 682 |  |
| Client 92 | file110k3.txt | 1633 | 2258 | 1268 | 1202 | 1223 |  |
| Client 142 | file110k4.txt | 3021 | 2124 | 1501 | 4072 | 2063 |  |
| Client 78 | file110k4.txt | 2296 | 1132 | 687 | 2064 | 2032 |  |
| Client 120 | file110k5.txt | 678 | 2442 | 682 | 673 | 710 |  |
| Client 61 | file110k5.txt | 964 | 2834 | 2681 | 772 | 1386 |  |
| Client 107 | file130k1.txt | 359 | 427 | 500 | 423 | 18 |  |
| Client 114 | file130k1.txt | 4 | 3349 | 7255 | 641 | 707 |  |
| Client 119 | file130k1.txt | 1568 | 1279 | 3831 | 1524 | 892 |  |
| Client 13 | file130k1.txt | 2385 | 5676 | 2685 | 2059 | 3063 |  |
| Client 131 | file130k1.txt | 407 | 2409 | 4530 | 356 | 367 |  |
| Client 14 | file130k1.txt | 702 | 2948 | 659 | 667 | 693 |  |
| Client 41 | file130k1.txt | 2063 | 132 | 1018 | 4609 | 14 |  |
| Client 60 | file130k1.txt | 1010 | 1590 | 695 | 4 | 3 |  |
| Client 84 | file130k1.txt | 443 | 13 | 2041 | 2152 | 1455 |  |
| Client 122 | file130k2.txt | 687 | 3488 | 1588 | 643 | 694 |  |
| Client 134 | file130k2.txt | 1463 | 649 | 5588 | 1474 | 1054 |  |
| Client 15 | file130k2.txt | 2346 | 9026 | 699 | 2009 | 3004 |  |
| Client 35 | file130k2.txt | 5088 | 10920 | 20 | 449 | 319 |  |
| Client 37 | file130k2.txt | 3818 | 7617 | 1246 | 1194 | 1672 |  |
| Client 0 | file130k3.txt | 264 | 129 | 145 | 130 | 128 |  |
| Client 116 | file130k3.txt | 634 | 745 | 1253 | 802 | 837 |  |
| Client 33 | file130k3.txt | 4849 | 9236 | 696 | 918 | 2465 |  |
| Client 48 | file130k3.txt | 690 | 702 | 735 | 696 | 694 |  |
| Client 126 | file130k4.txt | 1906 | 5586 | 8290 | 1806 | 1239 |  |
| Client 140 | file130k4.txt | 2081 | 781 | 696 | 1736 | 1592 |  |
| Client 26 | file130k4.txt | 2300 | 10295 | 682 | 1492 | 2586 |  |
| Client 45 | file130k4.txt | 663 | 648 | 825 | 2684 | 1223 |  |
| Client 52 | file130k4.txt | 1856 | 1337 | 1791 | 1714 | 1811 |  |
| Client 90 | file130k4.txt | 672 | 2622 | 1530 | 1248 | 1517 |  |
| Client 18 | file130k5.txt | 2119 | 9789 | 706 | 1963 | 639 |  |
| Client 40 | file130k5.txt | 1233 | 705 | 723 | 706 | 691 |  |
| Client 73 | file130k5.txt | 673 | 695 | 680 | 1243 | 701 |  |
| Client 80 | file130k5.txt | 2279 | 1240 | 805 | 2212 | 2069 |  |
| Client 85 | file130k5.txt | 856 | 3500 | 2446 | 1870 | 672 |  |
| Client 71 | file1m1.txt | 1664 | 998 | 1449 | 2309 | 1534 |  |
| Client 91 | file1m1.txt | 1150 | 3048 | 2785 | 1311 | 1967 |  |
| Client 32 | file1m2.txt | 4445 | 8080 | 729 | 2209 | 2096 |  |
| Client 72 | file1m2.txt | 2344 | 760 | 799 | 2927 | 2052 |  |
| Client 74 | file1m2.txt | 1361 | 1387 | 1341 | 1970 | 1361 |  |
| Client 10 | file1m3.txt | 2267 | 244 | 1918 | 799 | 2882 |  |
| Client 118 | file1m3.txt | 1663 | 4007 | 7354 | 1619 | 768 |  |
| Client 137 | file1m3.txt | 3536 | 1649 | 1105 | 4738 | 2454 |  |
| Client 146 | file1m3.txt | 3925 | 1436 | 756 | 2645 | 1221 |  |
| Client 25 | file1m3.txt | 1113 | 2171 | 742 | 1446 | 1500 |  |
| Client 70 | file1m3.txt | 2329 | 1941 | 2055 | 3056 | 2248 |  |
| Client 83 | file1m3.txt | 710 | 747 | 2095 | 1636 | 1617 |  |
| Client 11 | file1m4.txt | 2880 | 851 | 2473 | 2511 | 3481 |  |
| Client 16 | file1m4.txt | 1875 | 3901 | 1229 | 1754 | 1797 |  |
| Client 82 | file1m4.txt | 1442 | 912 | 1264 | 820 | 1484 |  |
| Client 102 | file1m5.txt | 778 | 769 | 773 | 727 | 783 |  |
| Client 113 | file1m5.txt | 707 | 5148 | 7902 | 1199 | 1191 |  |
| Client 59 | file1m5.txt | 3266 | 2888 | 1978 | 743 | 727 |  |
| Client 6 | file1m5.txt | 781 | 204 | 262 | 202 | 225 |  |
| Client 94 | file1m5.txt | 1279 | 3323 | 734 | 1600 | 2166 |  |
| Client 2 | file200k1.txt | 175 | 674 | 2427 | 680 | 642 |  |
| Client 22 | file200k1.txt | 709 | 2899 | 24 | 2310 | 1679 |  |
| Client 44 | file200k1.txt | 16 | 186 | 1082 | 4177 | 1127 |  |
| Client 62 | file200k1.txt | 520 | 2536 | 1164 | 23 | 1247 |  |
| Client 77 | file200k1.txt | 2093 | 764 | 612 | 901 | 1335 |  |
| Client 88 | file200k1.txt | 564 | 2735 | 1231 | 233 | 1123 |  |
| Client 115 | file200k2.txt | 656 | 660 | 3242 | 685 | 922 |  |
| Client 117 | file200k2.txt | 675 | 2882 | 6171 | 934 | 13 |  |
| Client 148 | file200k2.txt | 4526 | 1982 | 1365 | 3168 | 1777 |  |
| Client 101 | file200k3.txt | 1345 | 685 | 683 | 685 | 700 |  |
| Client 127 | file200k3.txt | 2228 | 5279 | 7290 | 699 | 1331 |  |
| Client 47 | file200k3.txt | 2913 | 674 | 1058 | 5365 | 693 |  |
| Client 99 | file200k3.txt | 2608 | 1253 | 1367 | 1871 | 1930 |  |
| Client 100 | file200k4.txt | 2071 | 751 | 832 | 1357 | 1362 |  |
| Client 42 | file200k4.txt | 4998 | 679 | 1216 | 5341 | 723 |  |
| Client 65 | file200k4.txt | 727 | 1359 | 705 | 2118 | 696 |  |
| Client 129 | file200k5.txt | 711 | 2950 | 4819 | 693 | 753 |  |
| Client 132 | file200k5.txt | 1255 | 1795 | 3865 | 1170 | 1259 |  |
| Client 66 | file200k5.txt | 706 | 719 | 659 | 714 | 668 |  |
| Client 81 | file200k5.txt | 2862 | 1143 | 2485 | 1988 | 1990 |  |
| Client 125 | file350k1.txt | 1706 | 4830 | 6651 | 1564 | 664 |  |
| Client 28 | file350k1.txt | 727 | 1614 | 705 | 995 | 995 |  |
| Client 53 | file350k1.txt | 1227 | 715 | 1158 | 1166 | 1223 |  |
| Client 96 | file350k1.txt | 1023 | 2153 | 657 | 692 | 1173 |  |
| Client 138 | file350k2.txt | 2573 | 1239 | 726 | 3522 | 2179 |  |
| Client 149 | file350k2.txt | 2780 | 704 | 713 | 685 | 690 |  |
| Client 43 | file350k2.txt | 714 | 715 | 1582 | 5516 | 687 |  |
| Client 57 | file350k2.txt | 1988 | 808 | 1957 | 3101 | 240 |  |
| Client 58 | file350k2.txt | 2445 | 1233 | 2392 | 2218 | 657 |  |
| Client 93 | file350k2.txt | 845 | 1646 | 880 | 674 | 694 |  |
| Client 54 | file350k3.txt | 1912 | 688 | 1138 | 5023 | 746 |  |
| Client 55 | file350k3.txt | 3488 | 717 | 2673 | 5836 | 10 |  |
| Client 9 | file350k3.txt | 1296 | 701 | 1239 | 712 | 1231 |  |
| Client 1 | file350k4.txt | 237 | 651 | 2743 | 8 | 256 |  |
| Client 112 | file350k4.txt | 639 | 4472 | 10 | 1130 | 561 |  |
| Client 135 | file350k4.txt | 3011 | 1090 | 7226 | 19 | 1995 |  |
| Client 20 | file350k4.txt | 672 | 2284 | 12 | 1234 | 18 |  |
| Client 21 | file350k4.txt | 972 | 3200 | 1685 | 1847 | 1795 |  |
| Client 29 | file350k4.txt | 20 | 1145 | 22 | 484 | 599 |  |
| Client 36 | file350k4.txt | 3756 | 7561 | 1216 | 548 | 1113 |  |
| Client 50 | file350k4.txt | 2262 | 211 | 1429 | 4149 | 1008 |  |
| Client 8 | file350k4.txt | 14 | 6 | 223 | 253 | 20 |  |
| Client 123 | file350k5.txt | 1393 | 6258 | 2299 | 1338 | 711 |  |
| Client 128 | file350k5.txt | 3836 | 5353 | 7390 | 809 | 1330 |  |
| Client 38 | file350k5.txt | 2618 | 6964 | 880 | 695 | 704 |  |
| Client 46 | file350k5.txt | 2467 | 681 | 1229 | 3553 | 1095 |  |
| Client 69 | file350k5.txt | 1026 | 1597 | 1023 | 2491 | 1088 |  |
| Client 12 | file500k1.txt | 747 | 692 | 177 | 721 | 720 |  |
| Client 145 | file500k1.txt | 3318 | 1848 | 711 | 1283 | 1270 |  |
| Client 39 | file500k1.txt | 2922 | 5239 | 1169 | 719 | 765 |  |
| Client 133 | file500k2.txt | 965 | 700 | 5151 | 992 | 453 |  |
| Client 3 | file500k2.txt | 197 | 219 | 2964 | 725 | 216 |  |
| Client 34 | file500k2.txt | 3308 | 6315 | 701 | 666 | 1295 |  |
| Client 108 | file500k3.txt | 660 | 723 | 704 | 700 | 1518 |  |
| Client 121 | file500k3.txt | 1146 | 4600 | 2050 | 986 | 1027 |  |
| Client 141 | file500k3.txt | 3475 | 1224 | 1034 | 5200 | 2411 |  |
| Client 68 | file500k3.txt | 1418 | 1900 | 2055 | 2942 | 2204 |  |
| Client 95 | file500k3.txt | 760 | 2776 | 1323 | 1098 | 1156 |  |
| Client 136 | file500k4.txt | 2736 | 1320 | 6720 | 2226 | 2181 |  |
| Client 144 | file500k4.txt | 3433 | 1556 | 868 | 4959 | 1345 |  |
| Client 5 | file500k4.txt | 192 | 302 | 358 | 241 | 267 |  |
| Client 130 | file500k5.txt | 763 | 1348 | 2728 | 705 | 740 |  |
| Client 56 | file500k5.txt | 2597 | 1262 | 2500 | 4866 | 1235 |  |
| Client 87 | file500k5.txt | 682 | 685 | 703 | 739 | 695 |  |
| Client 89 | file500k5.txt | 1214 | 2163 | 1919 | 1476 | 1850 |  |
| Client 106 | file75k1.txt | 705 | 678 | 703 | 681 | 1518 |  |
| Client 147 | file75k1.txt | 3935 | 1951 | 1420 | 5391 | 1872 |  |
| Client 4 | file75k1.txt | 282 | 198 | 212 | 175 | 201 |  |
| Client 109 | file75k2.txt | 3 | 11 | 11 | 11 | 648 |  |
| Client 111 | file75k2.txt | 3 | 3413 | 142 | 752 | 12 |  |
| Client 139 | file75k2.txt | 2626 | 7 | 11 | 4324 | 786 |  |
| Client 17 | file75k2.txt | 1898 | 4545 | 1718 | 2211 | 1776 |  |
| Client 23 | file75k2.txt | 2911 | 10672 | 668 | 1385 | 2357 |  |
| Client 31 | file75k2.txt | 1737 | 1199 | 16 | 680 | 696 |  |
| Client 63 | file75k2.txt | 883 | 1410 | 1557 | 158 | 1725 |  |
| Client 76 | file75k2.txt | 743 | 791 | 737 | 2057 | 790 |  |
| Client 67 | file75k3.txt | 1086 | 2244 | 2346 | 4047 | 2530 |  |
| Client 143 | file75k4.txt | 3717 | 1814 | 1246 | 4515 | 1622 |  |
| Client 19 | file75k4.txt | 1388 | 2456 | 688 | 1301 | 672 |  |
| Client 30 | file75k4.txt | 1451 | 3433 | 684 | 1668 | 2281 |  |
| Client 51 | file75k4.txt | 2543 | 1341 | 1737 | 4388 | 710 |  |
| Client 98 | file75k4.txt | 1844 | 3860 | 666 | 1150 | 1172 |  |
| Client 103 | file75k5.txt | 963 | 11 | 12 | 14 | 440 |  |
| Client 105 | file75k5.txt | 3 | 17 | 18 | 704 | 332 |  |
| Client 110 | file75k5.txt | 3 | 1240 | 12 | 304 | 438 |  |
| Client 24 | file75k5.txt | 1882 | 8390 | 704 | 1735 | 2219 |  |
| Client 64 | file75k5.txt | 1023 | 2848 | 1649 | 653 | 1840 |  |
| Client 7 | file75k5.txt | 673 | 174 | 1543 | 168 | 2252 |  |
| Client 75 | file75k5.txt | 2306 | 742 | 687 | 3518 | 1348 |  |
| Client 97 | file75k5.txt | 667 | 665 | 667 | 646 | 683 | Average |
| Total ms: | | 246699 | 348504 | 256339 | 256857 | 177934 | 257266.6 |