Computer Science and Information Technology 7(5): 129-161, 2019 http://www.hrpub.org
DOI: 10.13189/csit.2019.070501

Mobile Agent Based Distributed Network Architecture with Map Reduce Programming Model

Benard O. Osero[1,*], Elisha Abade[2], Stephen Mburu[2]

[1]Department of Computer Science, Chuka University, Kenya
[2]School of Computing and Informatics, University of Nairobi, Kenya

Received September 9, 2019; Revised October 9, 2019; Accepted October 23, 2019

Abstract In the recent years, the demand for data processing has been on the rise prompting researchers to investigate new ways of managing data. Our research delves into the emerging trends of data management methods, one of which is the agent based techniques and the active disk technology and also the use of Map-reduce functions in unstructured data management. Motivated by this new trend, our architecture employs mobile agents technology to develop an open source framework called SPADE to implement a simulation platform called SABSA. The architecture in this research compares the performance of four network storage architectures: Store and forward processes(SAF), Object Storage Devices(OSD), Mobile agent with a Domain Controller (DMC) enhanced with map-reduce function and Mobile agent with a Domain Controller and child DMC enhanced with Map-reduce (ABMR): both handling sorted and unsorted metadata. In order to accurately establish the performance improvements in the new hybrid agent based models and map-reduce functions, an analytic simulation model on which experiments based on the identified storage architectures were performed was developed and then analytical data and graphs were generated. The results indicated that all the agents based storage architectures minimize latencies by up to 45 % and reduce access time by up to 21% compared to SAF and OSD.

Keywords Domain Controller, Object Storage, Agent Based Storage, Metadata, Network Attached Storage

1. Introduction

Virtualization is a critical determinant which defines the path distributed storage array systems should follow in order to succeed. By its nature, virtualization manages data right from its source. The storage value has been changing from drives to the array cluster controllers while enhanced and data protection policies are included in such systems with time. [64].

2. Overview of the Existing Systems

2.1. Direct Attached Storage

In their classification model [4] refer this model as Direct attached storage while [54] referred this type of storage as store and forward (SAF); in this type of storage the network disks involved have to keep a copy of another redundant disk in the server. Every time a client requests for a file a copy of the file has to be kept before it is forwarded to the client for downloading. [54] further compared SAD and NASD and demonstrated that by keeping a copy of the disk there was a penalty on performance and scalability he further demonstrated an improved security mechanisms using tokenization on these platforms. He concluded that such systems can be improved by use of Object storage management schemes and he proposed further work on mobile agent and mobile code migration on a distributed network.

2.2. Network-Attached Storage Devices

Network-attached storage (NAS) happens at the file-level where one or more dedicated servers and disks store data and share it l other clients on a network.

Network-Attached Secure Disks (NASD), is a Networked object based shared storage system shown [4] in their classification taxonomy, that modifies the interface for the common direct attached storage devices and thus eliminating the server resources required for the movement of data. Figure 1 outlines the major components of NASD ARCHTECTURE [23,54].

The model allows for the transfer data directly from the client to the server via the file manager. The NASD execution sequence shows the requests needed until the file is downloaded to the client as explained [54-55].

The architecture below demonstrates a framework of the NASD Architecture by [54-55].

2.3. Object-Based Storage (OSD)

For Object storage the data is broken into small connected units called objects kept in a single repository (pool), instead of being kept as blocks on servers.

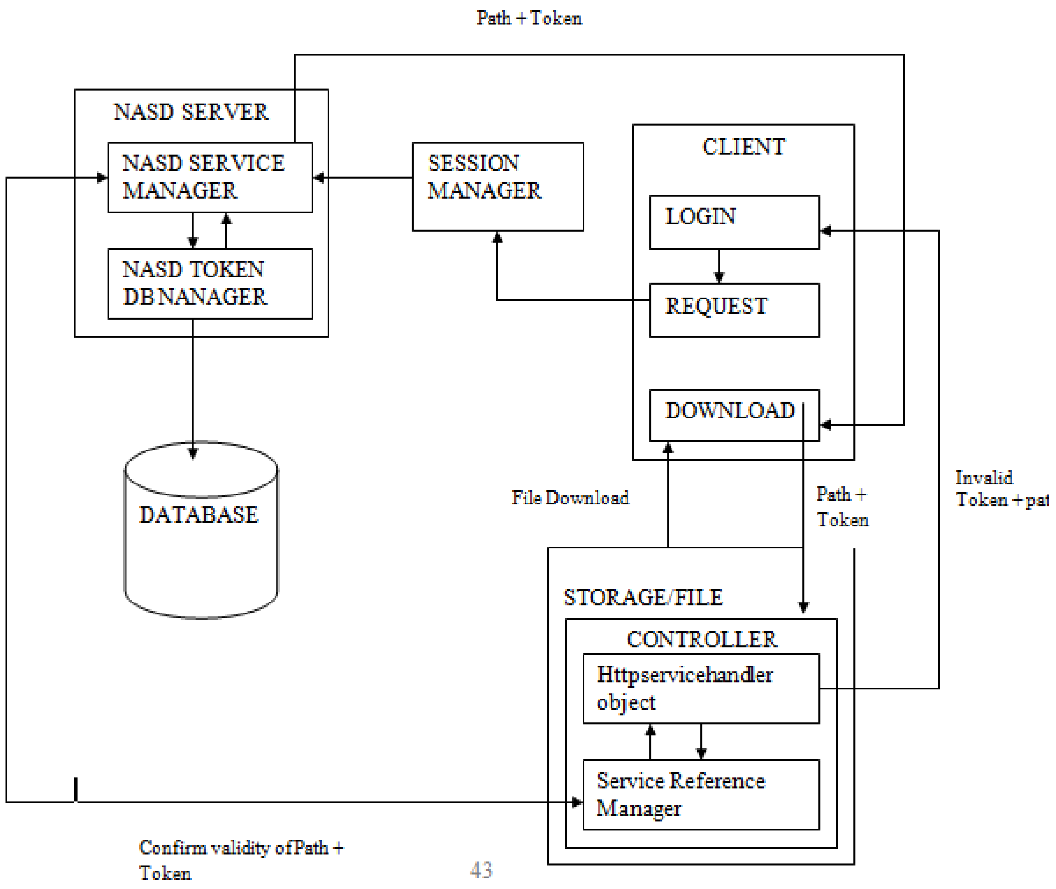


Figure 1. The major components of NASD Architecture

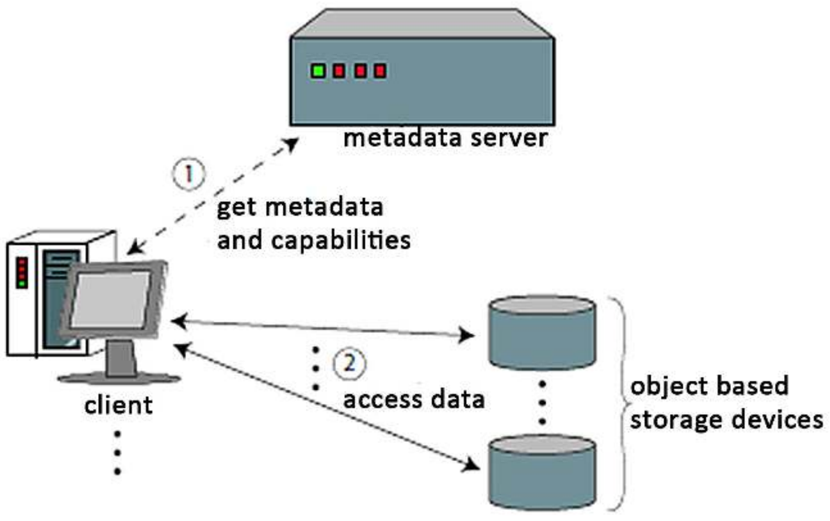


Figure 2. Direct client access

Figure 2 above depicts how client interaction between the server in order to get metadata access information [29]. The Object storage concept builds on previous work on NASD researched [23,54,55].

A storage object may be described as a sequence of addressed bytes, including other associated features accessed via a file interface with read, write and delete commands. This model is limited by the available bandwidth from a file server, clients and connected storage.

Object-based storage Research has been an ongoing subject that continues to elicit a lot of interest from researchers [17,45,18] early products such as Lustre [53], SWIFT[51] Panasas in [59] and [62] have equally appeared in the market. Additionally, object-based storage is being leveraged as a means to put together application-defined and other data features for compliance purposes [12,62]. For object-based storage to be of better use outside its current applications, it has to handle little chunks of files efficiently.

2.4. Deficiencies with the Current Systems

i) There has been an increase in the data processing demands and this requires faster systems that can scale well over short periods of time, most of the systems so far covered in the literature are either traditional in nature like SAF (DAS) systems which are array based or they are object based like NASD and OSD but they experience *high latencies and decreased throughput* as witnessed in [55,54,26,17,45,18].

ii) Low bandwidth and unmanaged Latencies have a major effect on the performance a distributed cluster or network, data prefetching methods have been implemented through predictive prefetching algorithms, but little progress has so far been made on metadata management schemes [86]; Although, [7] provide a solution to bandwidth issues which occur when the client and server interact, this solution only improves on bandwidth and not latencies.

Based on related works discussed above, there are major gaps or potential improvement areas in the approaches in the design and implementation of a distributed network and ways of testing performance of these systems. This research therefore dedicates its effort towards development and testing of concepts, components and implementation of suitable algorithms and models required for implementation of a suitable architectural model for distributed storage on a network.

3. Proposed Solution

The system to solve the above identified gaps introduces an intelligent way of managing a distributed network through the use of mobile agents, the agents are controlled by a central controller called the Domain Controller that keeps track of all the mobile agents in its registry. This intelligent network has the capability of minimizing latencies by localizing data in its local cache for subsequent client access.

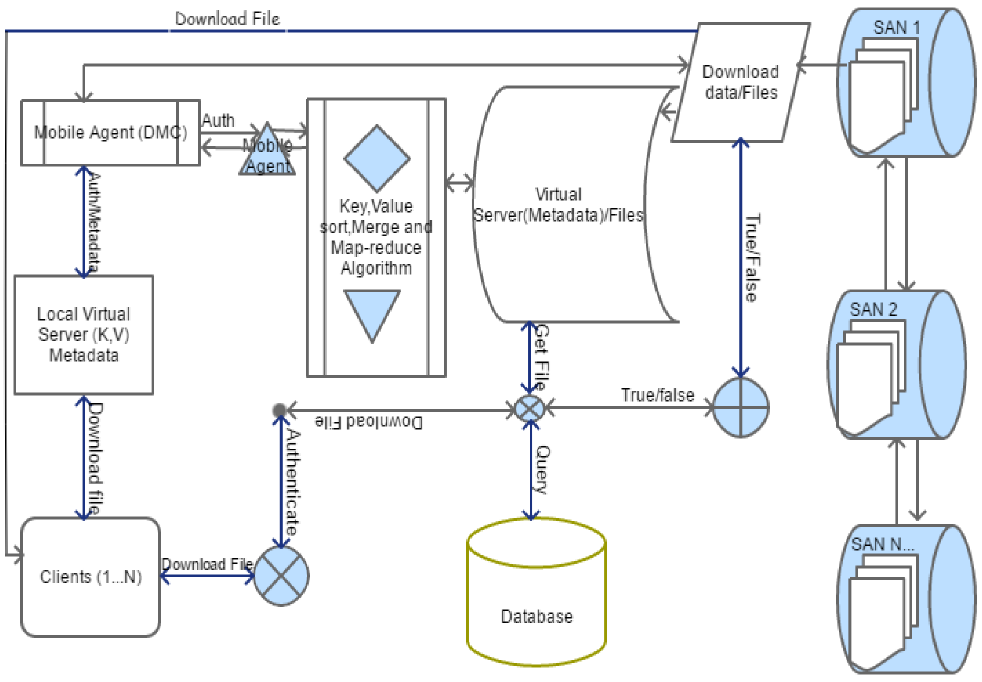


Figure 3. A Conceptual Architecture for intelligent objects using agents and Map-reduce.

Figure 3 above shows an architectural model of the agent based design using map-reduce, it is a three tiered model with the client as the front end the virtual server as the middle tier and storage SAN as the backend, it also includes the following functionalities:

Map-reduce Functions represented as shown below:

$$[(k_i, v_i), \dots, (k_n, v_n)]: \forall i=1 \dots n: \ (k_i, \ \in K, v_i, \in V) \quad (1)$$

k_i = ith input and v_i = ith input value. K represents the key domain and V represents the value domains. Map reduce function performs the responsibility of splitting the key, value input values to subsets and eventually distributing them to respective nodes. The mapping can be obtained as follows

$$Map: K \times V \rightarrow (L \times W)^* \quad (2)$$

$$(k, v) \rightarrow [(l_1, x_1), \dots, (l_1, x_r)] \quad (3)$$

L and W represents the key and value domains indicating the key-value input pairs: (k,v) mapped onto many key and value pairs: $(l_1, x_1), \dots, (l_1, x_r)$]. The reduce phase is as follows:

$$Reduce: L \times W^* \rightarrow W^* \quad (4)$$

$$(l, [y_1, \dots, y_{sl}]) \rightarrow [W_1, \dots, W_{ml}] \quad (5)$$

Keys with similar results are grouped together in this phase, input may have dissimilar values, the output generated has same keys. This requires sorting of results and consolidating them for final processing. This is performed by input function $(L \times W)^*$ which when processed produces $(L \times W)^*$ as sorted input values for the reduce function. $(L \times W)^*$ can also be mapped to generate a new list W^*. Map reduce can generally be summarized as follows:

$$Map\ Reduce: (K \times V)^* \rightarrow (L \times W)^* \quad (6)$$

This function is responsible for sorting and reducing metadata functions which can then be transported to client side for further processing.

3.1. Solution Outline

The Storage virtualisation architecture in this research involves a logical access to storage resources through defined metadata rather than having access to the physical resource itself; because physical file occupies more space. In the storage unstructured files are not necessarily arranged and accessed in a certain predefined sequence; this is made much more difficult when the data is randomly assorted into the storage from random sources; since their metadata is randomly placed in the storage searching for a file requested by the client becomes very difficult. Therefore, several methods or parameters can be employed to shuffle or sort the metadata depending on the intended mapping outcome i.e. one can sort as per the number of occurrences of the file, the IP address similarities, the file Index name or Number similarities etc.

To address the gaps in this research which have a direct impact on both latencies and response times the conceptual Architecture defined in Figure 3 employed metadata sorting and shuffling by use of map reduce algorithm by creating corresponding IP addresses of the respective client domains which were then mapped to a mobile agent for migration to a Domain Controller (DMC), where they were executed henceforth, a client can terminate normally or abnormally in case of an unrecoverable event.

The main objective of this research was to test whether a mobile agent can improve the performance of a distributed network if used together with the map reduce algorithm to sort the metadata within the virtual server in order to create sorted IP domain localities. A conceptual architecture was introduced and discussed in detail in figure 3. However, the conceptual architecture being a high level design model; only identified key architectural artifacts/components that of a three tiered distributed storage architecture.

In order to actualize the concepts in this research into a concrete design and develop a model capable of giving indicative performance, some important issues needed to be tackled as follows:

i) To develop and implement a simulator that will assist in testing the scalability and performance.
ii) To implement a search, sort and mapping mechanism using map-reduce functions that can improve in searching, sorting and mapping of metadata in the virtual server.
iii) To Implement Agents on a Distributed network using the SPADE architecture.
iv) Evaluate the performance of Agent based Distributed system using Map-reduce with non-agent based distributed network environment specifically SAF and OSD in the implemented simulator.

3.2. The Need for the SABSA Simulator

Our simulation model utilized the experimental methodology in order to answer various research questions raised by the researcher. The simulation model used in this research was developed from scratch using open source software tools; this was arrived at after critical analysis of various simulation platforms. the Network simulation models so far existing are only meant for the first three Physical layers of the OSI model (Physical, Datalink and the Network) and therefore, they were suited for the physical characteristics of the network, but our research could not be supported by this popular network platforms as described by [71] and [31] because we were performing file simulation at the sockets level; which occurs at the upper layers of the OSI model (Transport and Session layers)and therefore there was need to design a simulator to cater for the desired and unique file simulation features.

Our simulation model was meant to test performance improvement after introducing agents into the existing SAF and OSD models and also sorting metadata using map reduce approach. The performance measures have been described by [25,90,50] and [85] which were adopted as a performance measure for our SABSA engine to provide the metrics that were used for this research.

3.2.1. Simulator Requirements and Design

Our simulator was developed using standard Python APIs for the design of the client and server machines using Sockets and standard Docker Containers running on the Linux OS cluster for file storage virtualization (SAN) and also SHA1 encryption scheme for security.

3.2.2. System Requirements and Configuration

The system was configured to run on an AWS EC2 t2. micro virtual machine instance with:

The host machine was a HP Folio 9470, equipped with an Intel Core(TM)i7 CPU 2.60 GHz CPU, 8 GB RAM, a 500 GB hard disk drive, 64-bit OS, x64 based processor and a GSM wireless network connection and a Windows 2015 Pro Operating System. The system was also hosted on the Amazon EC2.

The system also required installation of Docker on the virtual machine to run it. Configuration was done using Docker files which specified the VM image used as well as run instructions for system setup. Instructions included setting environment variables that were used by the application and installing of the packages that were used as well as exposing ports which were to be accessed from outside of the virtual machine.

4. Simulations and Results

The simulator in this research was developed using standard Python APIs for the design of the client and server machines using Sockets and standard Docker Containers running on the Linux OS cluster for file storage virtualization (SAN) and also SHA1 encryption scheme for security. Our Model used a Simulation model with various experiments being carried out at various levels.

4.1. Simulation Model

The Simulation and experimental methodologies were employed for our research as described by [89] and [8]. Our model development and testing followed the three basic steps defined in the literature by [8] as follows:

i) Created an Architectural model for approximating the events.
ii) The model was then simulated in a computer software, which allowed for the repeated observation of the model. After one or many simulations of our model
iii) A third step was analysis. Analysis aided in the drawing of conclusions, verification and validating the research, and hence enabling us to make recommendations based on various iterations or simulations of the model.

The simulation model utilized the experimental methodology in order to answer various research questions raised by the researcher and also described by [52] in developing his research methodology. The simulation model used in this research was developed from scratch using open source software tools; this was arrived at after critical analysis of various simulation platforms of the Network simulation models so far existing are only meant for the first three Physical layers of the OSI model (Physical, Datalink and the Network) and therefore, were suited for the physical characteristics of the network, but our research could not be supported by this popular network platforms as described by [71] and [31] because we were performing file level simulation at the sockets level; which takes place at the upper layers of the OSI model (Transport and Session layers)and therefore there was need to design a simulator to cater for the desired and unique file simulation features.

Our simulation model was meant to test performance improvement after introducing agents into the existing SAF and OSD models and also sorting metadata using map reduce approach. The performance measures have been described by [25,90,50] and [85] which were adopted as a performance measure for our SABSA engine to provide the metrics that were used for this research.

4.2. The Simulation Procedure

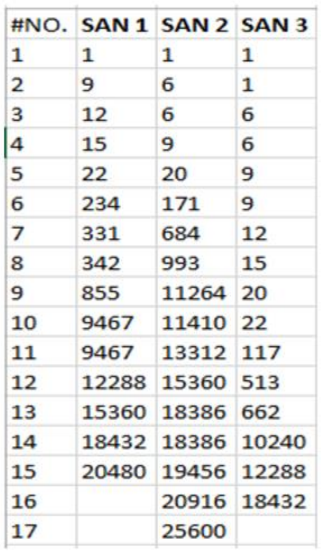
To easily address the gaps in this research a simulation testbed referred to as SABSA (Secure Agent Based Architecture) was developed in phases as follows:

1. Phase 1: Design and implementation a direct file access.
2. Phase 2: Design and Implementation of Object storage device (OSD).
3. Phase 3: Mobile agent Domain Controller (DMC) enhanced with map-reduce function.
4. Phase 4: Mobile agent based Domain Controller with child DMC controllers enhanced with Map-reduce.
5. Repeat Phase 2, 3 and 4 with sorted metadata.

The results were then presented in regard to the following identified parameters: Latencies, throughput and Scalability; either with sorted or unsorted metadata.

The existing files (Bytes) were first classified in the following table before they were run in the simulator:

Table 1. Files Sizes in the SAN container in Bytes.



#NO.	SAN 1	SAN 2	SAN 3
1	1	1	1
2	9	6	1
3	12	6	6
4	15	9	6
5	22	20	9
6	234	171	9
7	331	684	12
8	342	993	15
9	855	11264	20
10	9467	11410	22
11	9467	13312	117
12	12288	15360	513
13	15360	18386	662
14	18432	18386	10240
15	20480	19456	12288
16		20916	18432
17		25600	

The above files were extracted from the SAN container (SAN 1, SAN2 and SAN 3) and then entered into excel sheet and sorted in ascending order and the files were then sorted according to the combinations of three sizes as demonstrated in table 2 below.The file sizes were capped at 30000 bytes; because files beyond this size slowed down the machine drastistially.

4.2.1. File Classifications in Bytes (1 File Per San).

Table 2. Workload distribution matrix

SAN 1	**SAN 2**	**SAN 3**	**RANDOMLY SELECTED FILE OPTIONS**	**CASE DISCUSSION**
SMALL	SMALL	SMALL		CASE 1
SMALL	SMALL	MEDIUM	x	
SMALL	SMALL	LARGE	x	
SMALL	MEDIUM	SMALL		CASE 2
SMALL	MEDIUM	MEDIUM	x	
SMALL	MEDIUM	LARGE	x	
SMALL	LARGE	SMALL		CASE 3
SMALL	LARGE	MEDIUM	x	
SMALL	LARGE	LARGE	x	
MEDIUM	SMALL	SMALL	x	
MEDIUM	SMALL	MEDIUM	x	
MEDIUM	MEDIUM	MEDIUM		CASE 4
MEDIUM	MEDIUM	LARGE	x	
MEDIUM	LARGE	SMALL	x	
MEDIUM	LARGE	MEDIUM		CASE 5
MEDIUM	LARGE	LARGE	x	
LARGE	SMALL	SMALL	x	
LARGE	SMALL	MEDIUM	x	
LARGE	SMALL	LARGE	x	
LARGE	MEDIUM	SMALL	x	
LARGE	MEDIUM	MEDIUM	x	
LARGE	MEDIUM	LARGE	x	
LARGE	LARGE	SMALL	x	
LARGE	LARGE	MEDIUM	x	
LARGE	LARGE	LARGE		CASE 6

5. Workload Analysis for Time, Latencies and Throughput

Various workloads were classified as per the defined set parameters in: Table 1, then various workloads were run per each SAN as indicated in table 1: under the defined parameters a summary of various outputs were generated under various Cases were used to the analyze performance of SABSA engine under varied workloads in all the SAN containers as indicated in Tables 3 -6 below.

Table 3, 4 and 6 show Comparison graphs as per the file classification matrix

The graphs have been inserted into a table with three columns; column 1 indicated by part a) represents 100 client requests, column 2 indicated by b) represents 1000 client requests and column three represents the case numbers identified by the file classifications in table 2. The classifications were done as either sorted or unsorted metadata functions as follows:

i) Design and implementation a direct file access-In this case there is no metadata involved, but the file is transmitted directly to the client.

ii) Design and Implementation of Object storage device (OSD)-In this case the metadata is centralized and presented as an unsorted array in the virtual server. The metadata can further be sorted before being handled by the mobile agents which creates sorted metadata domains.

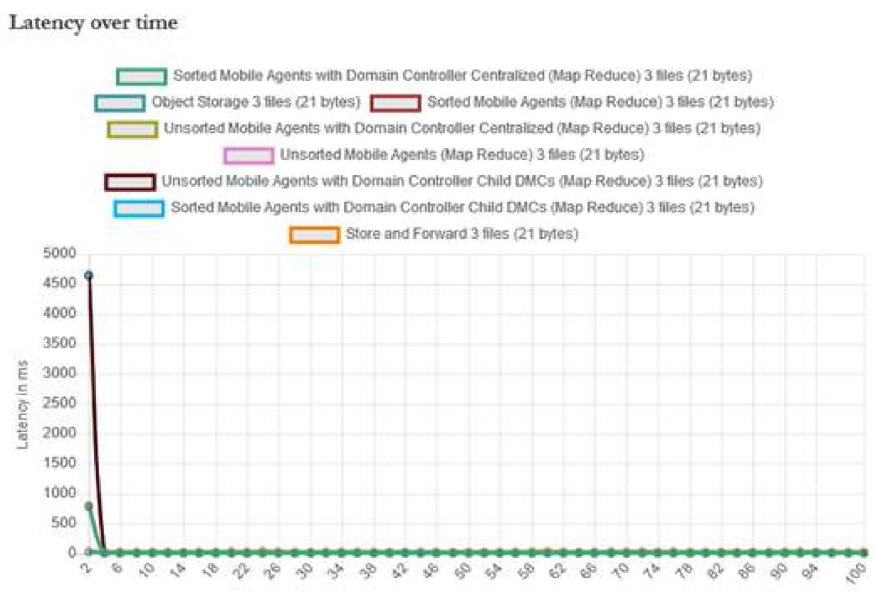
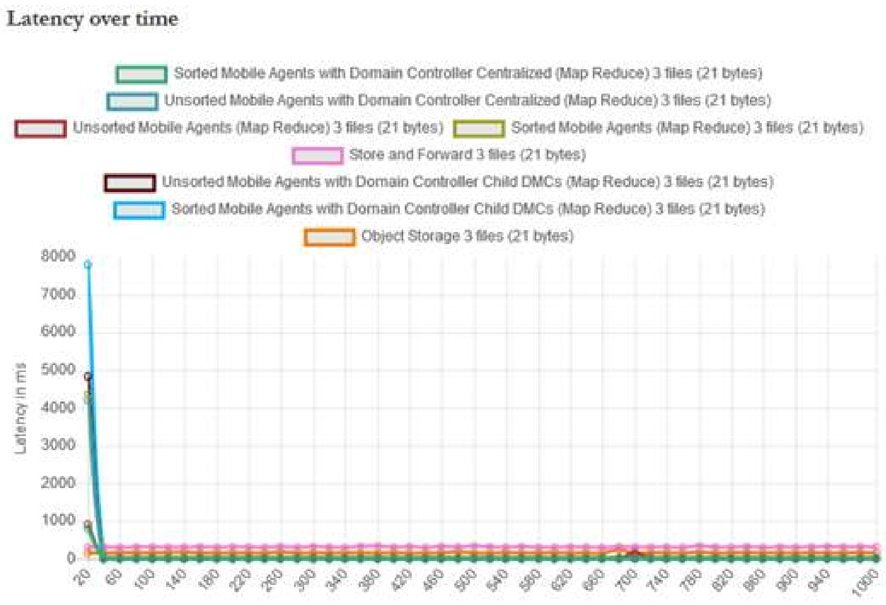
iii) Mobile agent Domain Controller (DMC) enhanced with map-reduce function-For mobile agents to function they are centrally managed by the domain controller, the agents collects the sorted metadata and moves it to the domain controller for further distribution. The unsorted metadata remains as case ii above.

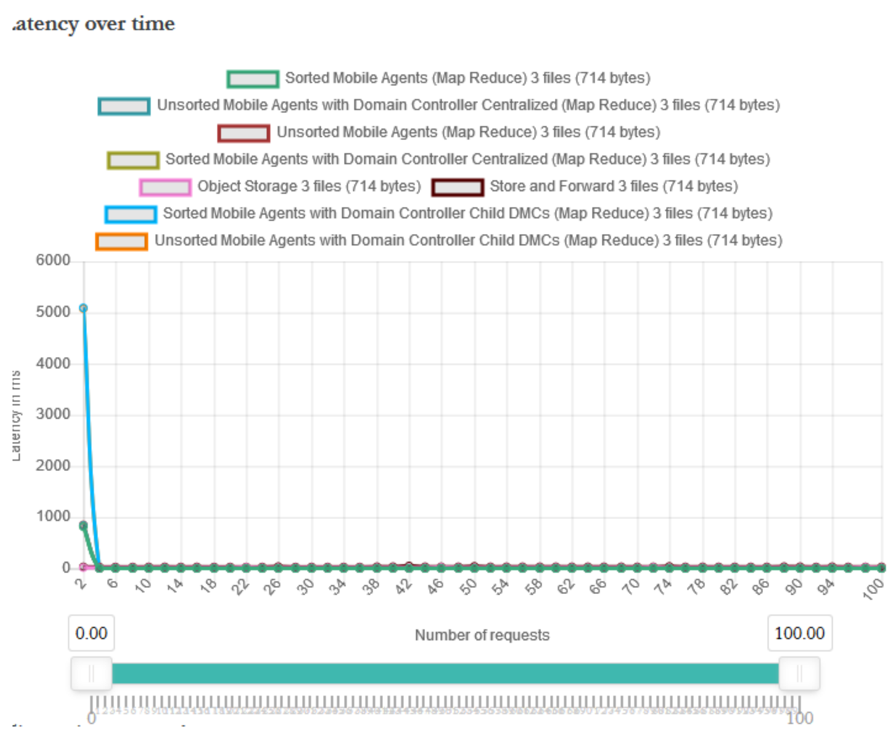
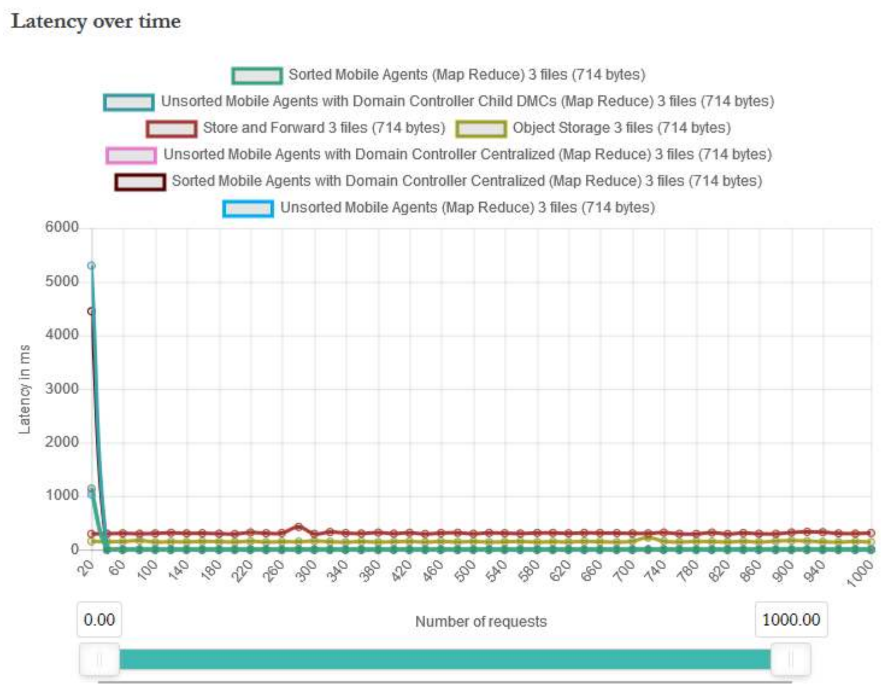
iv) Mobile agent based Domain Controller with child DMC controllers enhanced with Map-reduce- Case 2 above can further be decentralized to create more child processes distributed and controlled by a centralized agent controller still referred to as the domain controller. The agents in this case can also handle both sorted and unsorted metadata objects.

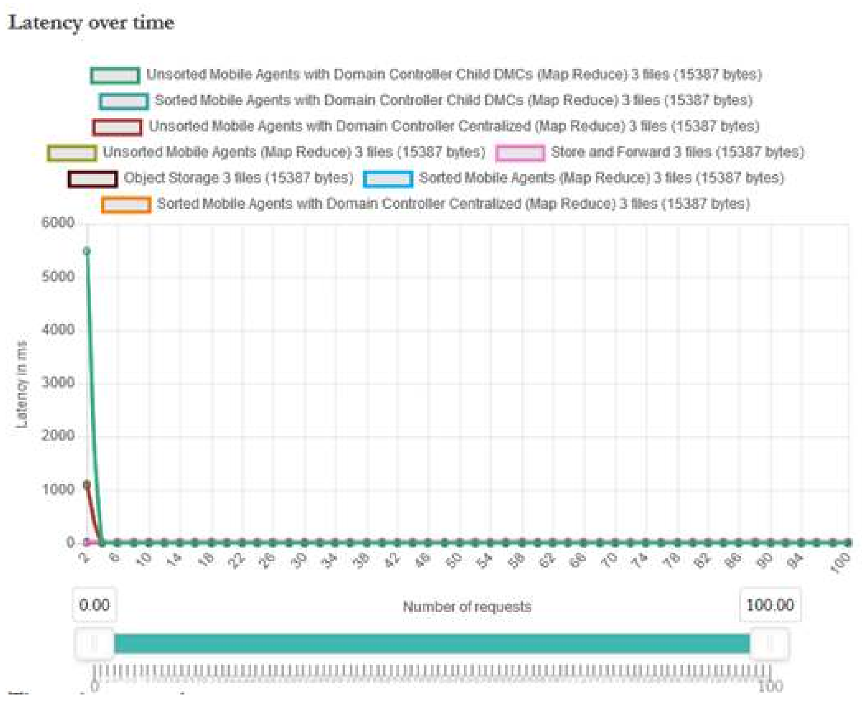
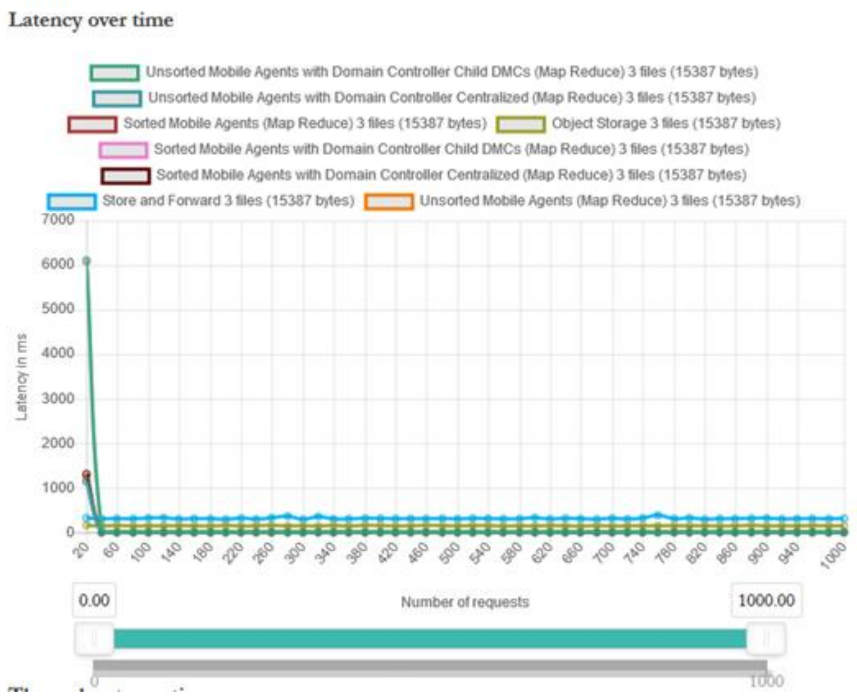
5.1. Latencies Comparison over Time

The graph in this table section represents the changes of latencies over time and it is subdivided into two column sections a) and b): column a) represents 100 client requests and column per millisecond b) represents 1000 client requests per millisecond.

Table 3. Latencies Comparison over time

1000 CLIENT FILE REQ.	CASE NO
 (a)	CASE 1
 (b)	CASE 1

 (a)	CASE 2
 (b)	CASE 2

 (a)	CASE 3
 (b)	CASE 3

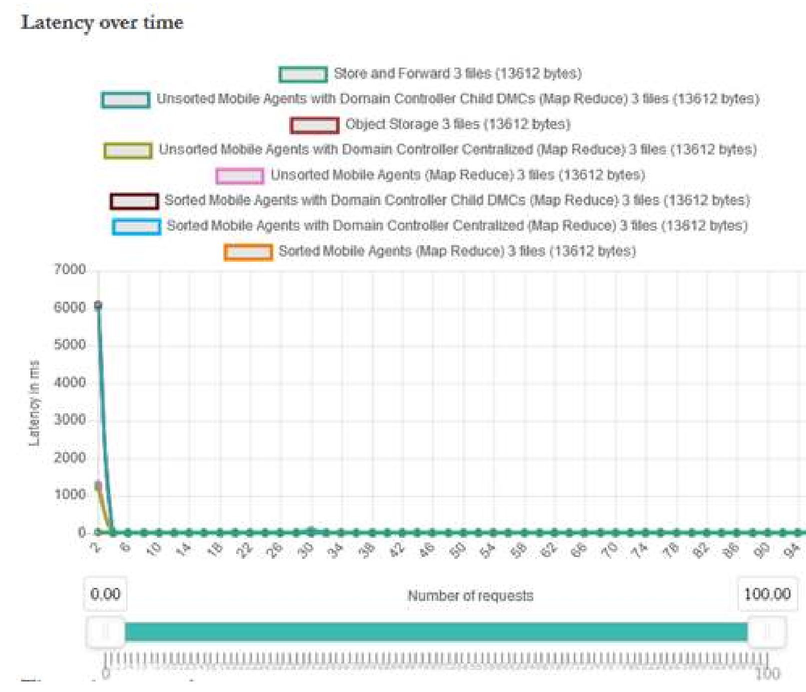
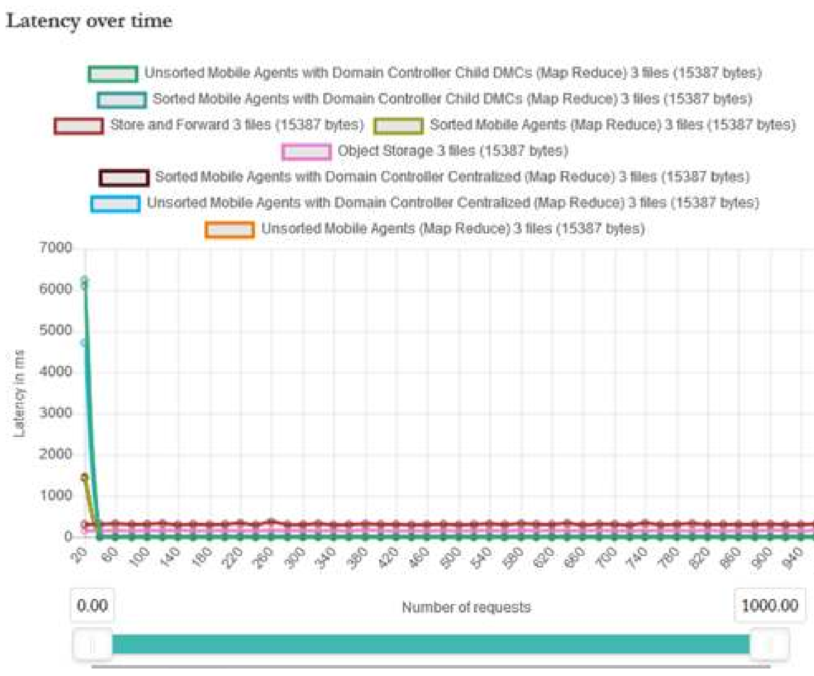
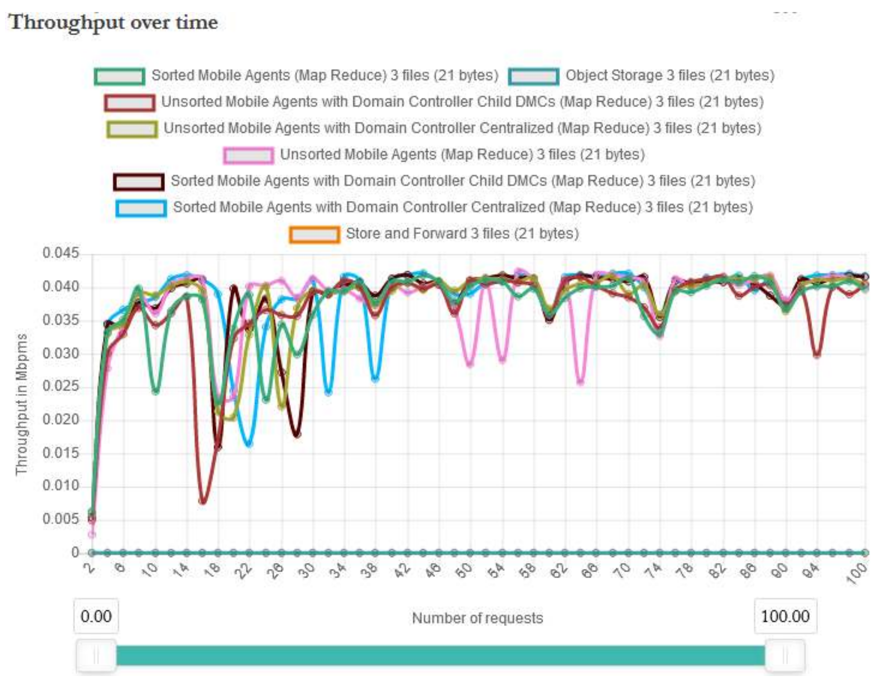
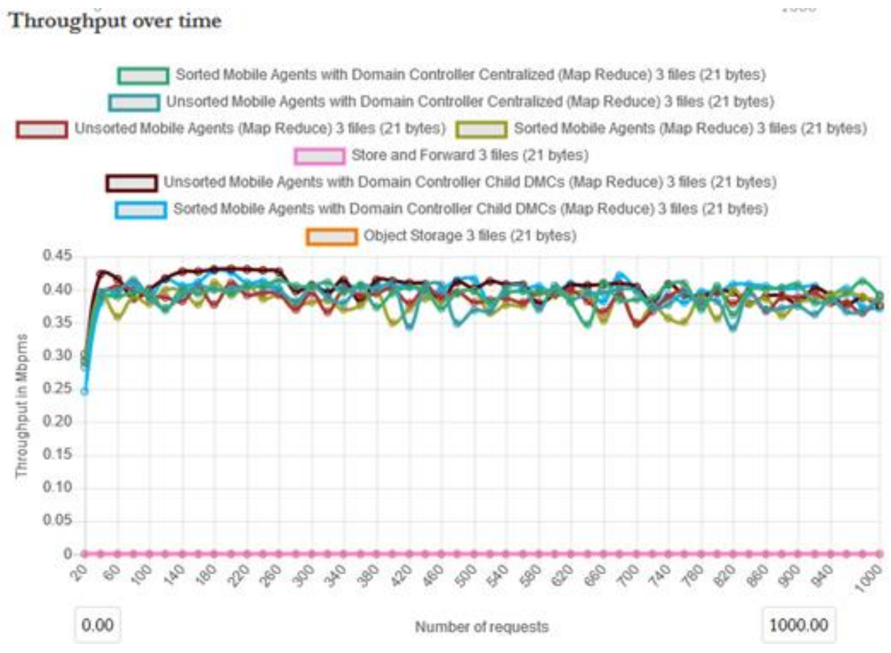
 (a)	CASE 4
 (b)	CASE 4

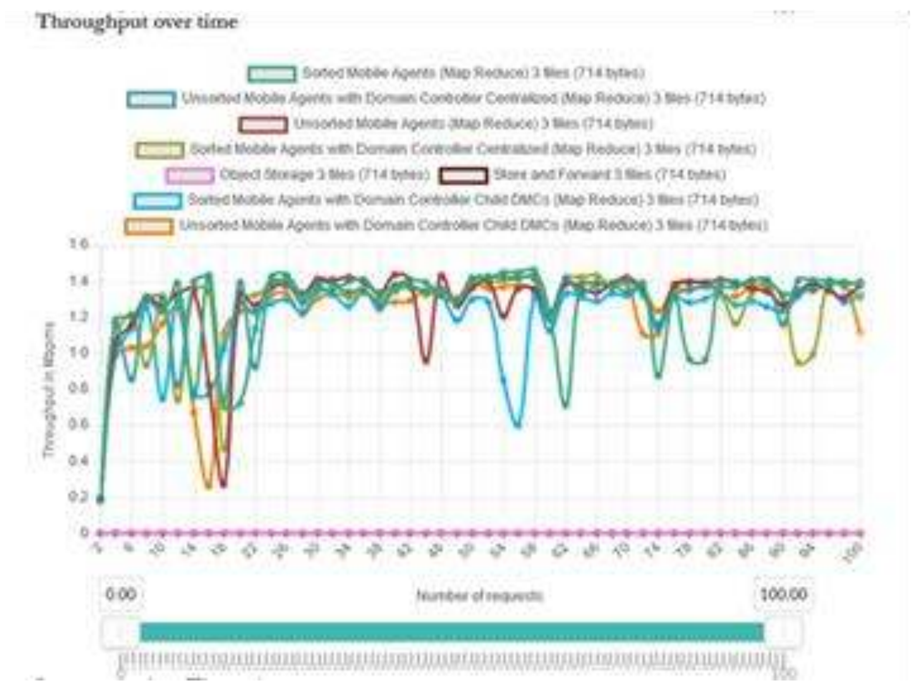
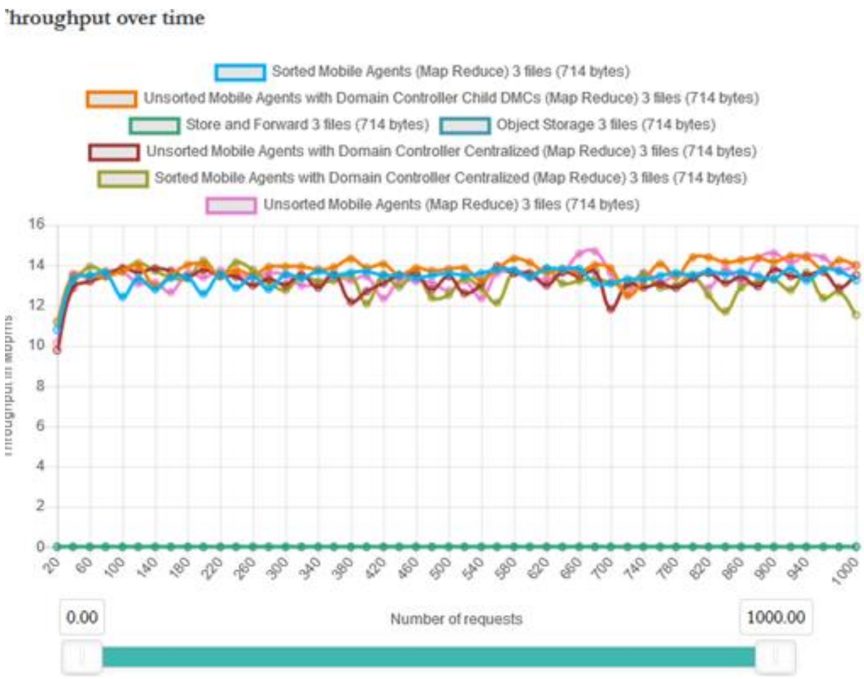
Table 3 shows comparison of various graph outputs under 100 and 1000 client classified under various CASES 1,2,4 and 6 part a) represents 100 client requests and part b) represents 1000 client requests

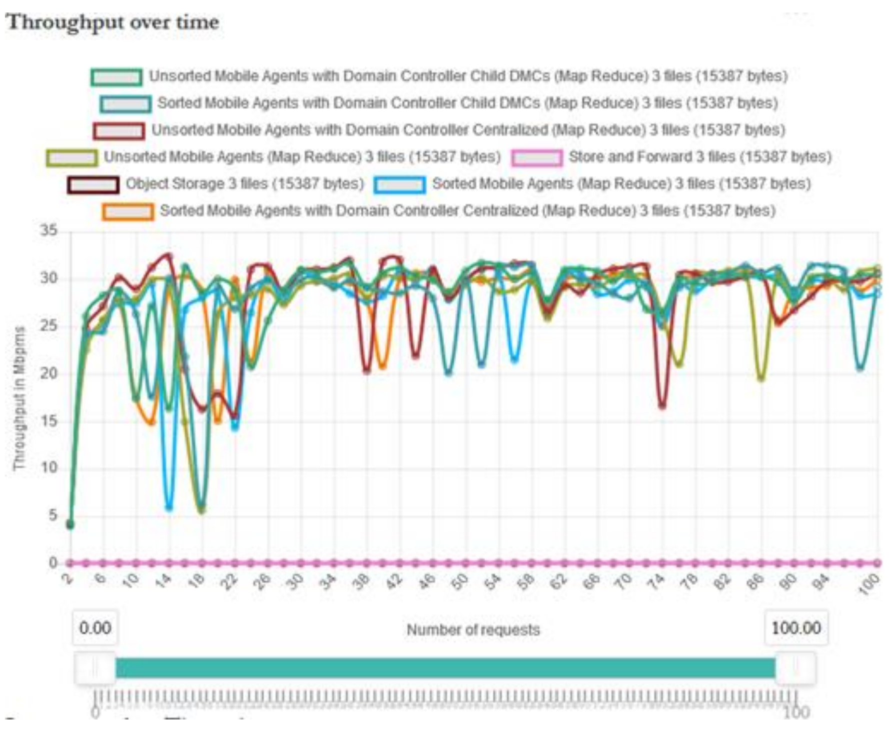
5.2. Throughput over Time

The graph in this table section represents the changes of Throughput over time and it is subdivided into two columns a) and b): column a) represents 100 client requests and column b) represents 1000 client requests.

Table 4. Throughput over time

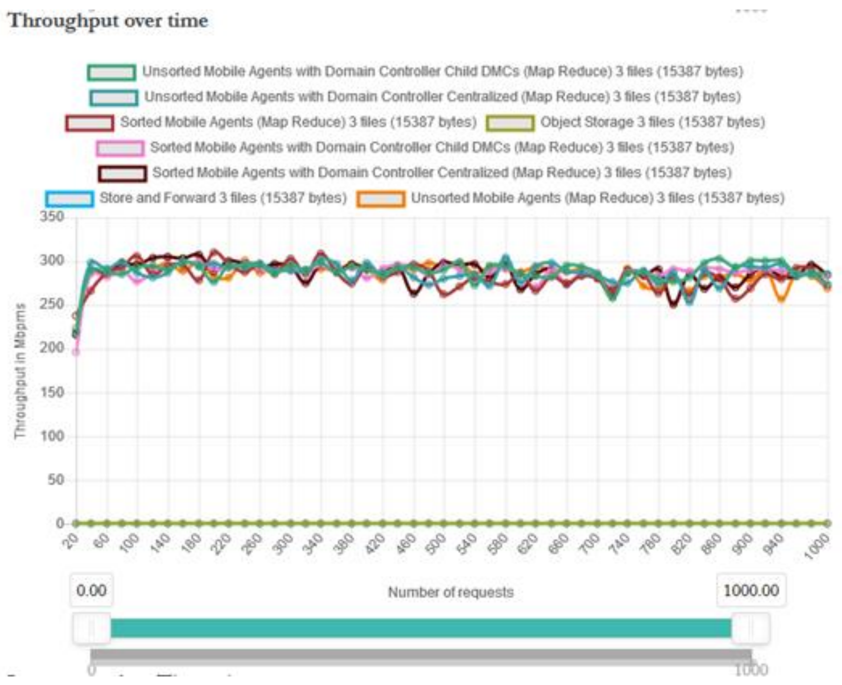
1000 CLIENT FILE REQ.	CASE NO
 (a)	CASE 1
 (b)	CASE 1

 (a)	CASE 2
 (b)	CASE 2



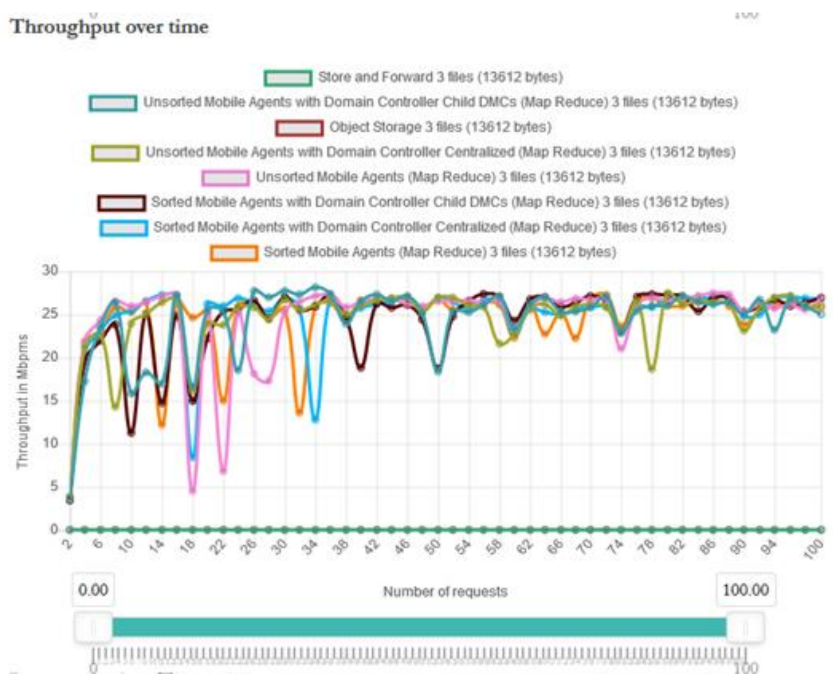
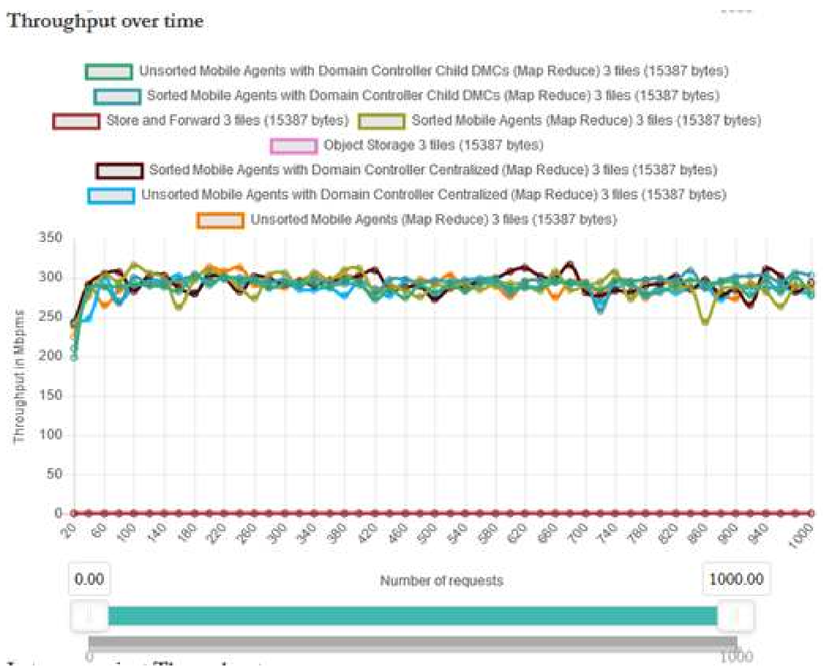
(a)

CASE 3



(b)

CASE 3

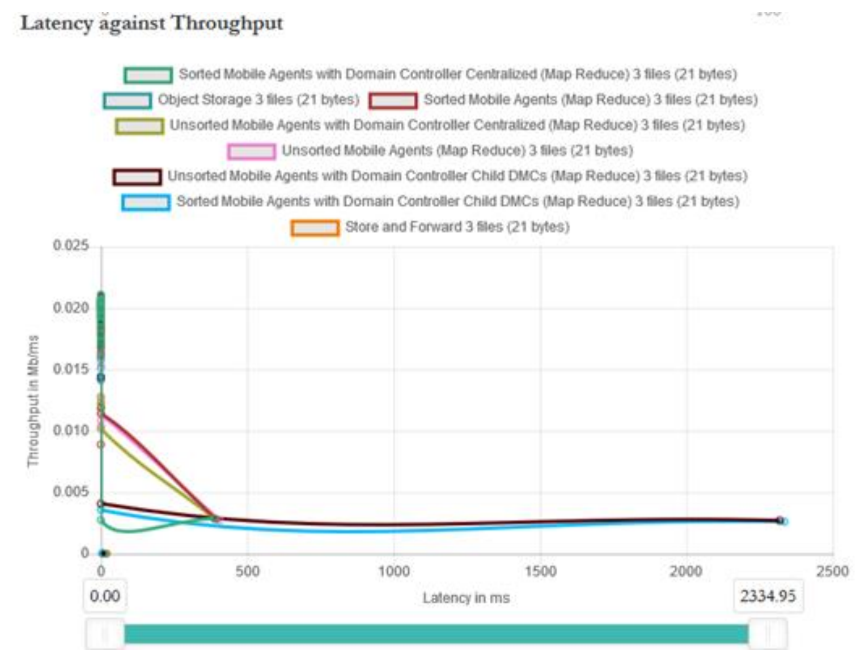
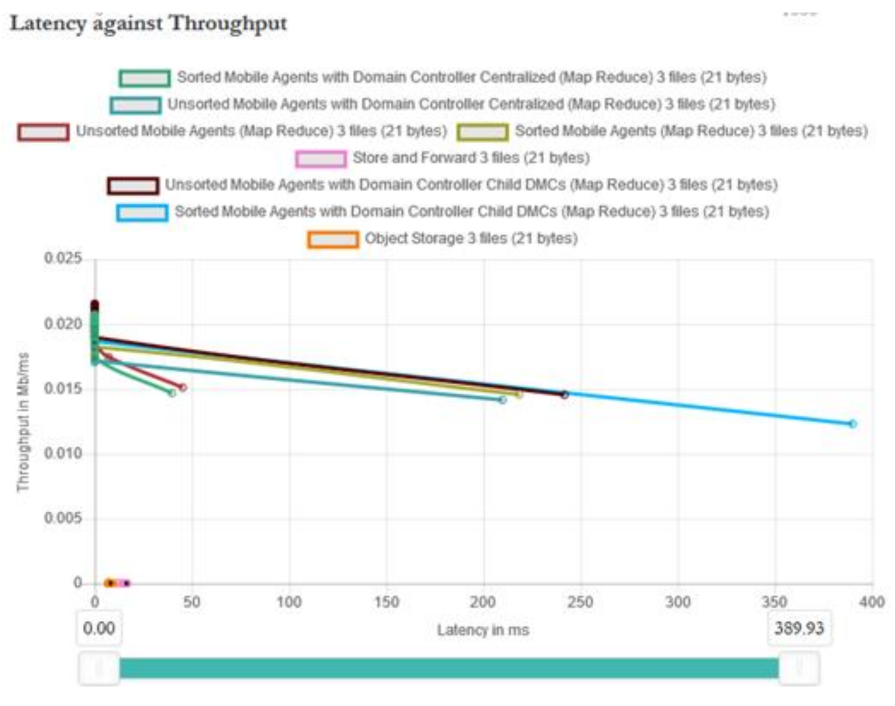
 (a)	CASE 4
 (b)	CASE 4

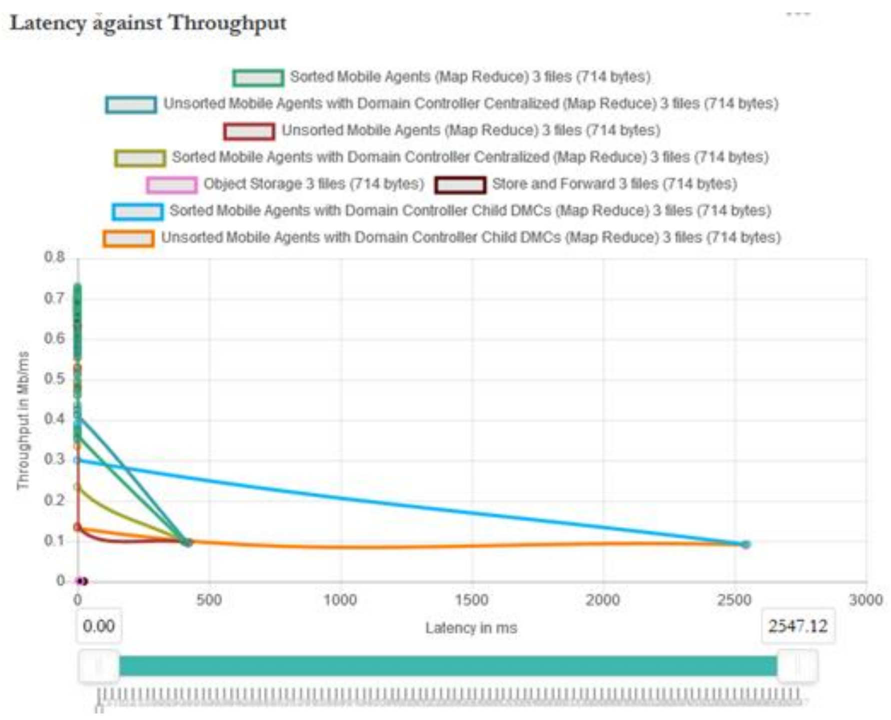
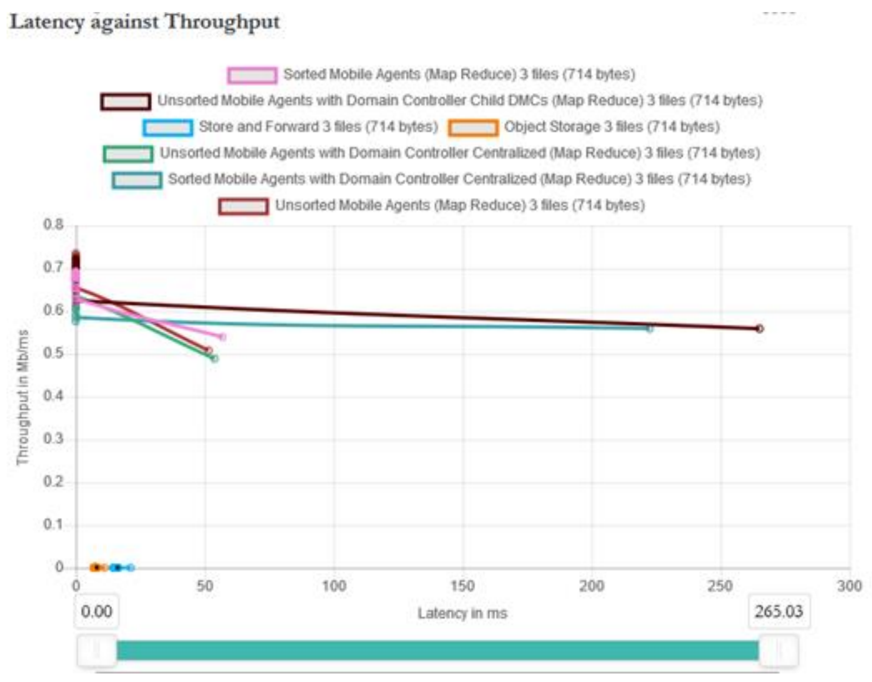
In table 4 case for instance indicates that in both case a and b store and forward has the minimum throughput at Zero and map reduce and sorting by mobile agents also has a significant improvement in throughput in all the cases observed, store and forward has a worst throughput in all cases and object based storage has an average throughput in all the cases. The maximum observed through for case 4a is 25 Mbpms, which drastically increases tenfold to 250 Mbpms. In all cases agents and map reduce combined show a very good improvement in throughput.

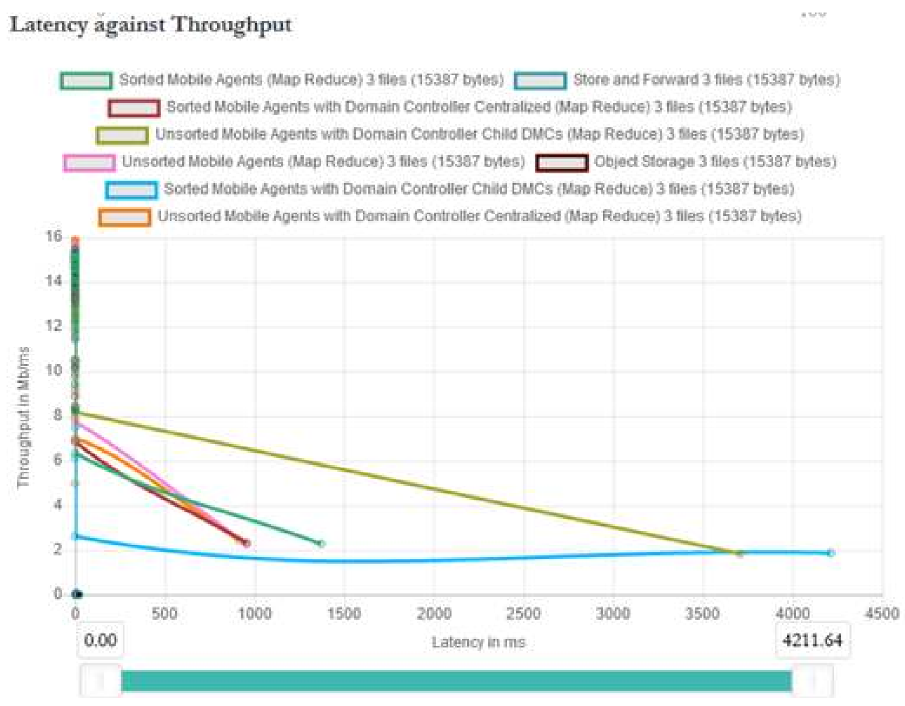
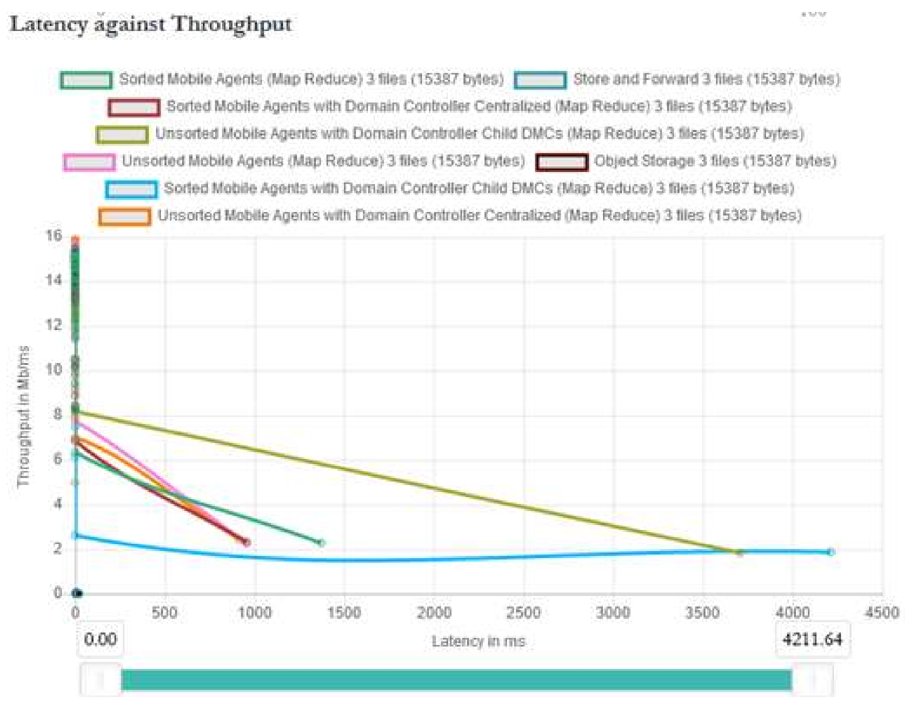
5.3. Latency against Throughput

The graph in this table section represents the effects of latencies on throughput and it is subdivided into two column Sections a) and b): column a represents 100 client requests and column b) represents 1000 client requests.

Table 5. Latency against throughput

1000 CLIENT FILE REQ.	CASE NO
 (a)	CASE 1
 (b)	CASE 1

 (a)	CASE 2
 (b)	CASE 2

 (a)	CASE 3
 (b)	CASE 3

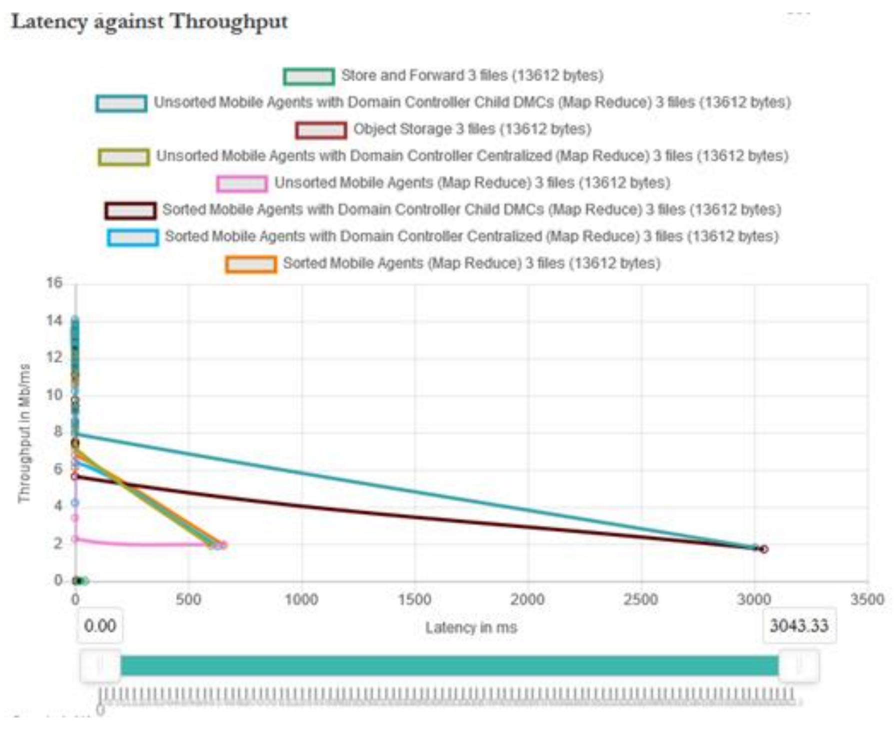
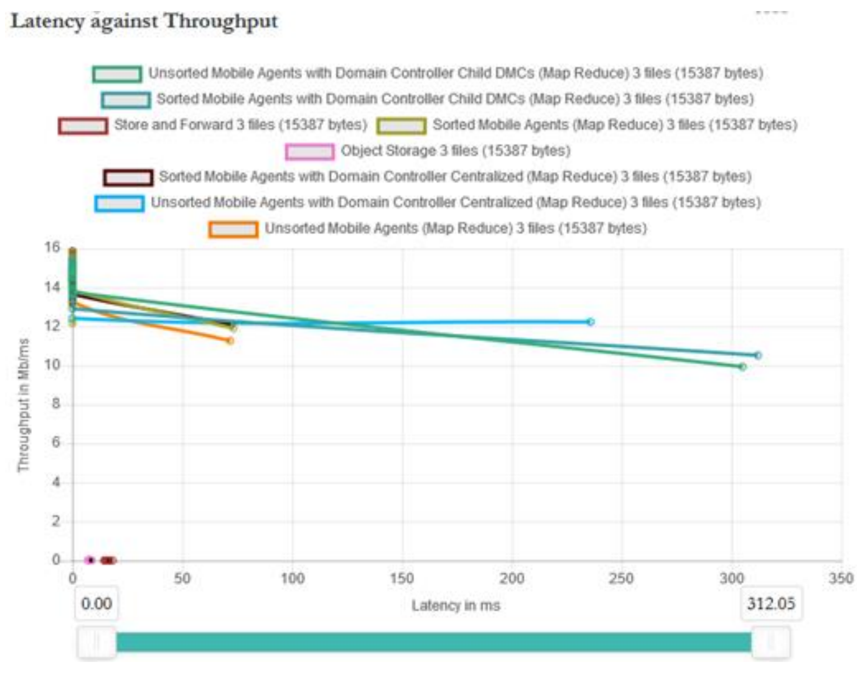
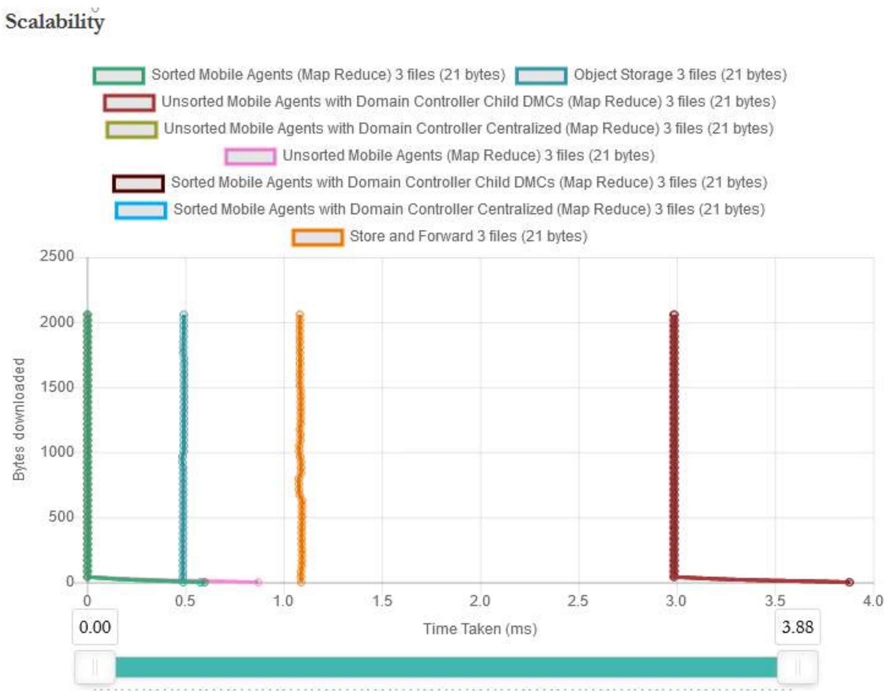
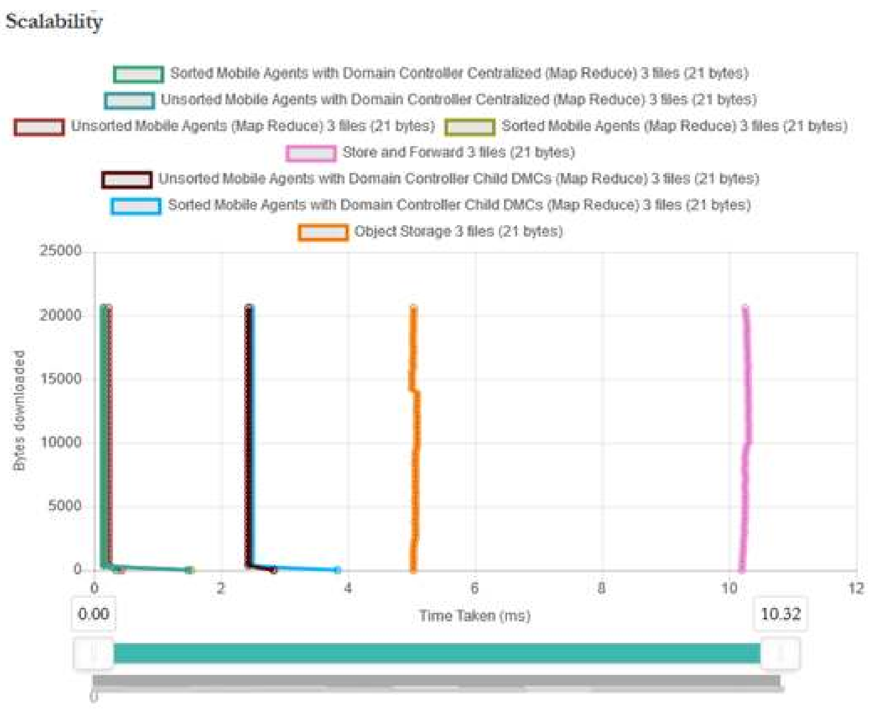
 (a)	CASE 4
 (b)	CASE 4

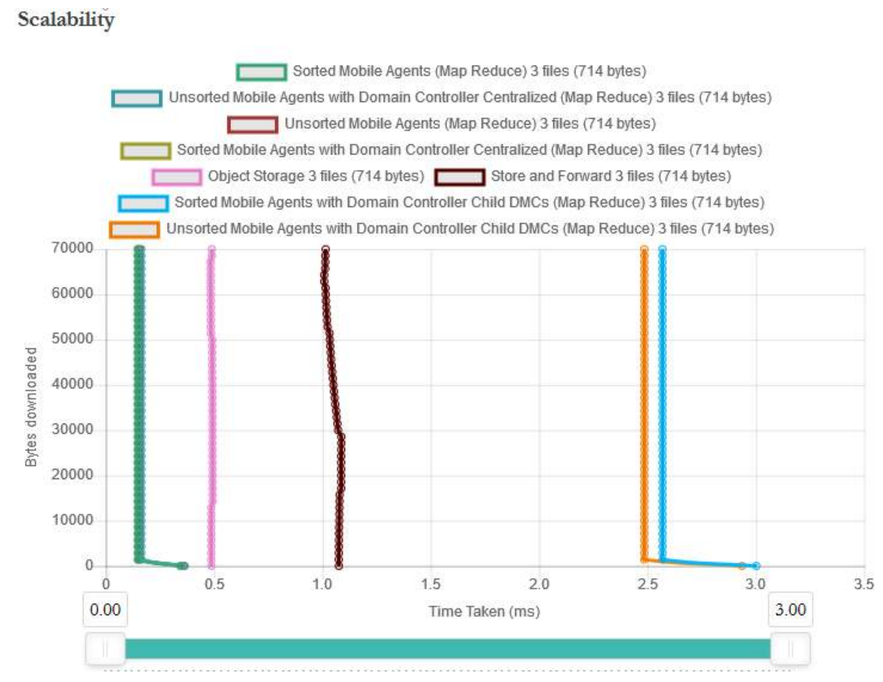
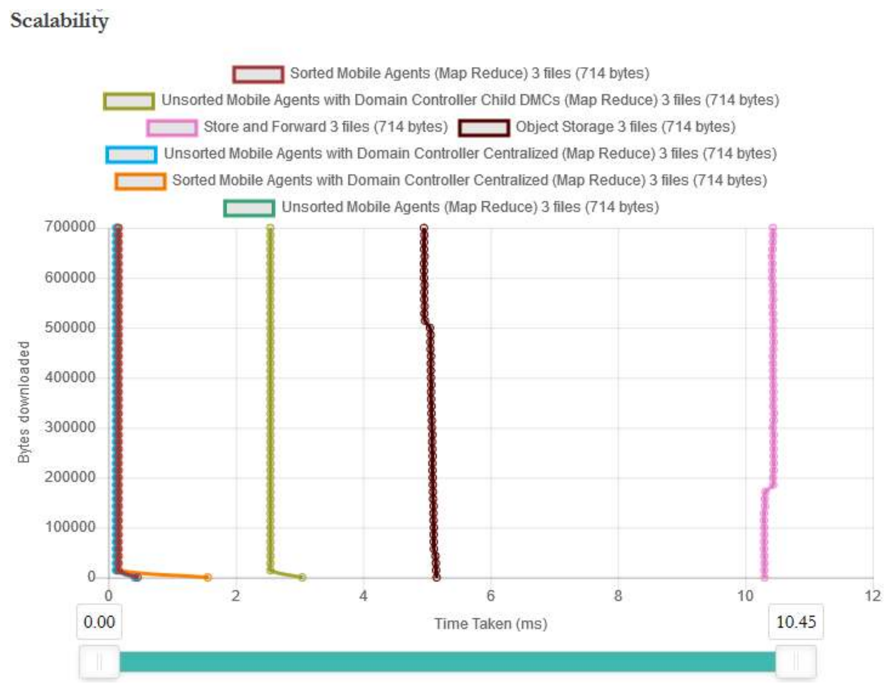
Table 5 above shows that latency decreases with increase in load for the agent based cases but remains at zero remains at the minimum for both OSD and SAF. Case 4a for instance indicates that at Zero Latency sorted mobile agents with domain controllers have the highest throughput and at 3000 ms latency throughput for sorted mobile agents with domain controller and sorted mobile agents with a centralized controller their throughput drastically reduced to 4 Mb/ms from the initial 8 Mb/ms. In 4b at Zero Latency All mobile agents based metadata sorting with map reduce exhibit highest throughput at averagely 14 Mb/ms unlike for 4a where the cases averaged around 7 Mb/ms and also the average latency moves drastically from the high of 3000 to an average high of 300 ms a tenfold decrease in Latency. All the other cases exhibit similar characteristics. This shows that agents are better in terms of improving latencies and consequently increasing throughput even at high load capacities.

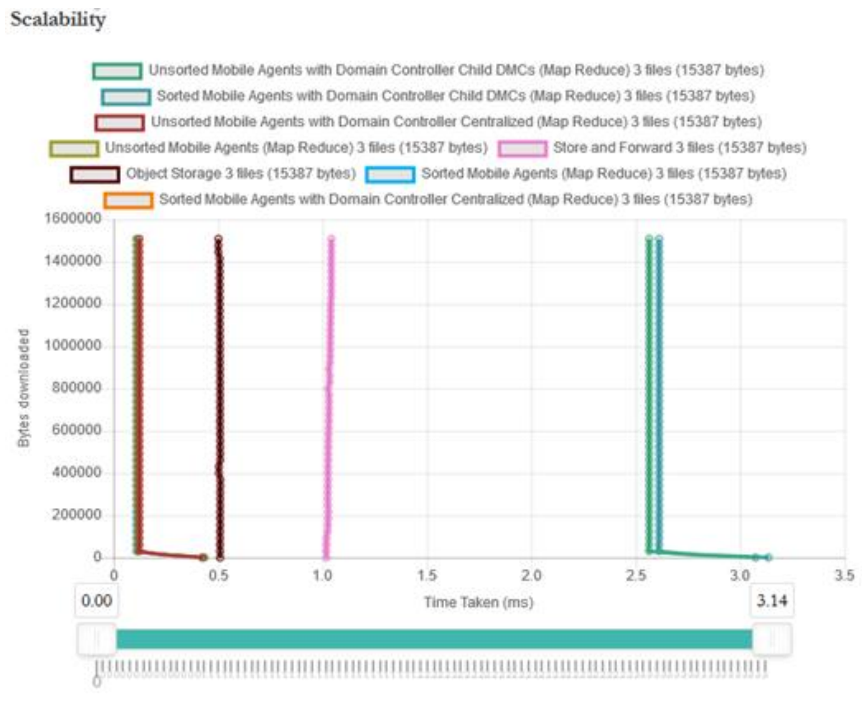
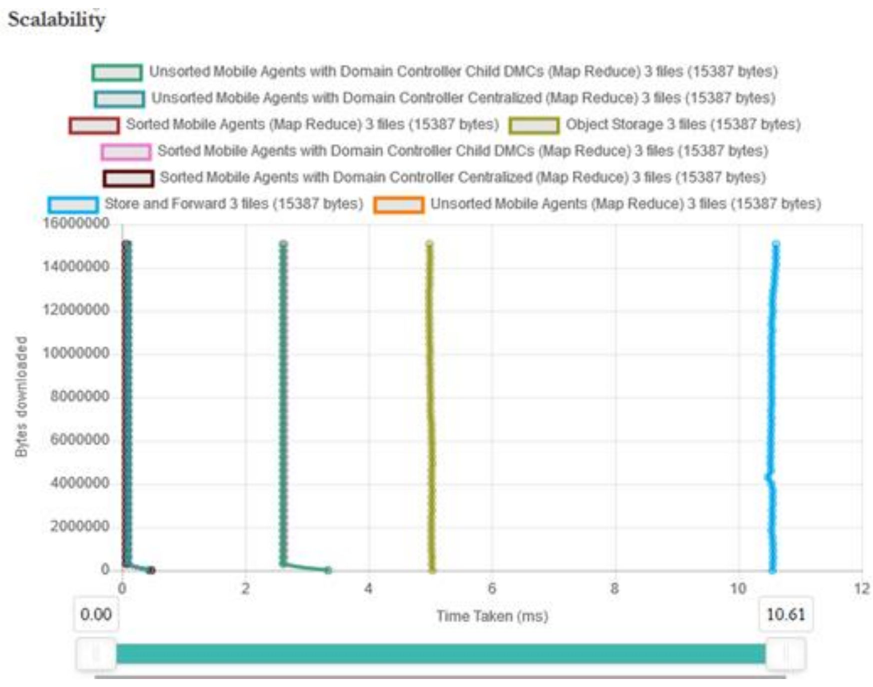
5.4. Scalability over Time

The graphs in this table section represents how the system scales over time and it is subdivided into two columns sections a) and b): column a represents 100 client requests and column b) represents 1000 client requests.

Table 6. Scalability over time

1000 CLIENT FILE REQ.	CASE NO
 (a)	CASE 1
 (b)	CASE 1

 (a)	CASE 2
 (b)	CASE 2

 (a)	CASE 3
 (b)	CASE 3

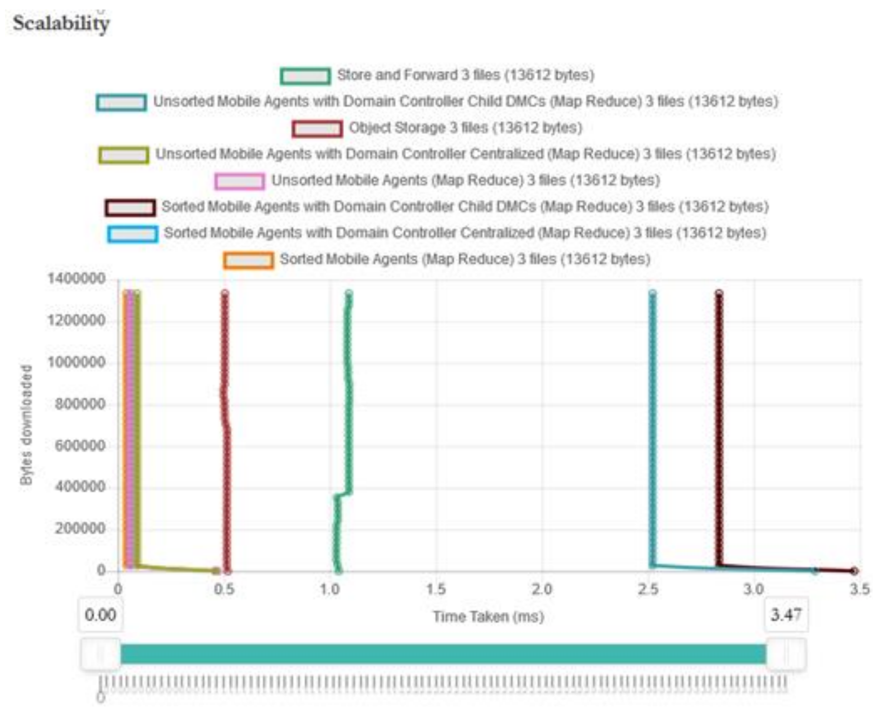
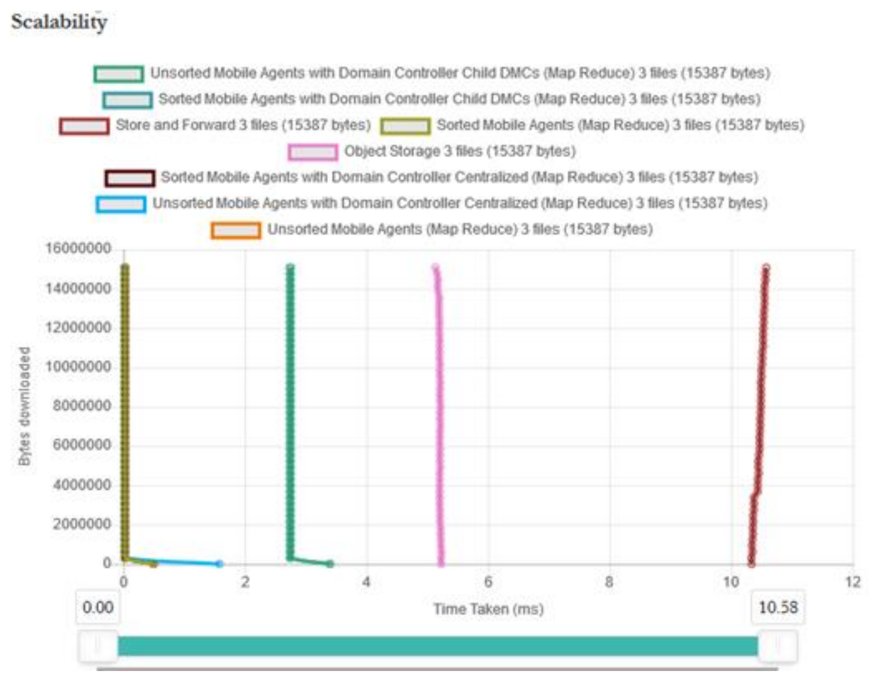
 (a)	CASE 4
 (b)	CASE 4

Table 6 above shows scalability of the SABSA engine with both 100 and 1000 client requests with files from three different SANs.

Table 6 above shows scalability over time, for instance CASE 4 shows that sorted agents with map reduce are the fastest in hundred client requests and sorted mobile agents with domain controllers don't perform well at this point on the other hand when the requests are increased to 1000 the sorted mobile agents with domain controllers have the best scalability, this shows that sorting of metadata by use of map reduce and further catching them improves not only performance but such systems are also highly scalable. Case 4a for instance shows that at an average of 1300000 bytes sorted mobile with child DMCs agents do not scale well at an average of 2.8 ms while unsorted mobile agents with map reduce without child DMCs perform well with the same load at 0 ms. For case 4b the at 15000000 bytes the unsorted mobile agents with map reduce still have the best scalability at 0 ms and store and forward has the worst scalability at averagely 10.5 ms. But overall the salability sorted mobile with child DMCs agents improves significantly to average at 2.5 ms.

5.5. Case Summaries for Csv File Data

The data in this section that was captured Table 7 to 12 below indicates a summary of the CSV outputs that were analyzed under various workload then Average time in Millisecond (ms), Throughput and latencies were captured and the compared under this predefined conditions.

It important to note that the files were run as batch files including multiple client requests for the required file in this experiment we have considered small and medium range workload requests.

A summarized table for the output of the time variance for downloading 1000 client requests from 100 client requests.

Table 7. CASE 1 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(9Bytes) +SAN2(6Bytes) +SAN3(6 Bytes): SMALL-SMALL-SMALL 100/1000 FILE REQUESTS

Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+Child DMCs	Unsorted De-Centralized MA-MR+DMC+ Child DMCs	Total File Size(SAN1+SAN2+SAN3)(Bytes)
AV.TT(ms) 1000	10.17	5.00	0.23	0.15	0.16	0.16	2.47	2.43	$2.1x10^1$
Av.TT(ms) 100	1.02	0.49	0.15	0.14	0.15	0.15	2.38	2.37	$2.1x10^1$
Throughput MB/s 1000	$1.07x10^{-5}$	$2.21X10^{-5}$	0.02	0.02	0.02	0.02	0.02	0.02	$2.1x10^1$
Throughput MB/s 100	$1.06x10^{-5}$	$2.21X10^{-5}$	0.02	0.02	0.02	0.12	0.02	0.02	$2.1x10^1$
AV.Latency(ms) 1000	15.17	7.48	0.23	0.15	0.81	4.21	7.81	4.84	$2.1x10^1$
AV.Latency(ms)100	15.21	7.37	8.12	7.88	7.76	7.77	46.71	46.39	$2.1x10^1$

Table 8. CASE 2 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(15 Bytes) +SAN2(684) +SAN3(15 Bytes): SMALL-MEDIUM-SMALL 100/1000 FILE REQUESTS

Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+Child DMCs	Unsorted De-Centralized MA-MR+DMC+ Child DMCs	Total File Size (Bytes)
AV.TT(ms) 1000	10.02	4.92	0.17	0.17	0.17	0.18	2.15	2.18	$7.14X 10^2$
Av.TT(ms) 100	1.04	0.49	0.16	0.15	0.15	2.49	2.57	2.49	$7.14X 10^2$
Throughput MB/s 1000	0.00	0.00	0.65	0.66	0.66	0.67	0.66	0.67	$7.14X 10^2$
Throughput MB/s 100	0.00	0.00	0.64	0.63	0.63	0.64	0.60	0.62	$7.14X 10^2$
AV.Latency(ms) 1000	15.12	7.37	0.89	0.88	0.92	0.88	4.08	4.12	$7.14X 10^2$
AV.Latency(ms)100	15.67	7.24	8.52	8.13	8.29	8.42	50.96	50.79	$7.14X 10^2$

Table 9. CASE 3 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(15Bytes) +SAN2(15360 Bytes) +SAN3(12 Bytes): SMALL-LARGE-SMALL 100/1000 FILE REQUESTS

Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+Child DMCs	Unsorted De-Centralized MA-MR+DMC+Child DMCs	Total File Size (Bytes)
AV.TT(ms) 1000	10.45	4.96	0.10	0.10	0.07	0.10	2.63	2.61	$1.54X 10^4$
Av.TT(ms) 100	1.02	0.51	0.11	0.11	0.12	0.13	2.62	2.57	$1.54X 10^4$
Throughput MB/s 1000	0.01	0.02	14.28	14.13	14.31	14.13	14.32	14.42	$1.54X 10^4$
Throughput MB/s 100	0.01	0.02	13.73	13.73	13.81	13.92	13.70	14.24	$1.54X 10^4$
AV.Latency(ms) 1000	15.64	7.43	1.27	1.16	1.33	1.16	6.08	6.12	$1.54X 10^4$
AV.Latency(ms)100	15.22	7.48	11.12	11.20	10.88	10.74	54.83	54.93	$1.54X 10^4$

Table 10. CASE 4 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(331 Bytes) +SAN2(993) +SAN3(12288 Bytes): MEDIUM-MEDIUM-LARGE 100/1000 FILE REQUESTS

Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+Child DMCs	Unsorted De-Centralized MA-MR+DMC+ Child DMCs	Total File Size (Bytes)
AV.TT(ms) 1000	10.37	5.16	0.02	0.02	0.03	0.02	2.74	2.76	$1.36X\ 10^4$
Av.TT(ms) 100	1.07	0.51	0.06	0.05	0.07	0.10	2.84	2.53	$1.36X\ 10^4$
Throughput MB/s 1000	0.01	0.02	14.50	14.51	14.64	14.34	14.65	14.41	$1.36X\ 10^4$
Throughput MB/s 100	0.01	0.01	12.26	12.21	12.41	12.28	12.13	12.20	$1.36X\ 10^4$
AV.Latency(ms) 1000	15.46	7.67	1.44	1.48	1.45	4.73	6.25	6.11	$1.36X\ 10^4$
AV.Latency(ms)100	15.98	7.58	13.11	13.12	12.62	12.04	60.88	60.08	$1.36X\ 10^4$

Table 11. CASE 5 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(234 Bytes) +SAN2(33312) +SAN3(662 Bytes): MEDIUM-LARGE-MEDIUM 100/1000 FILE REQUESTS

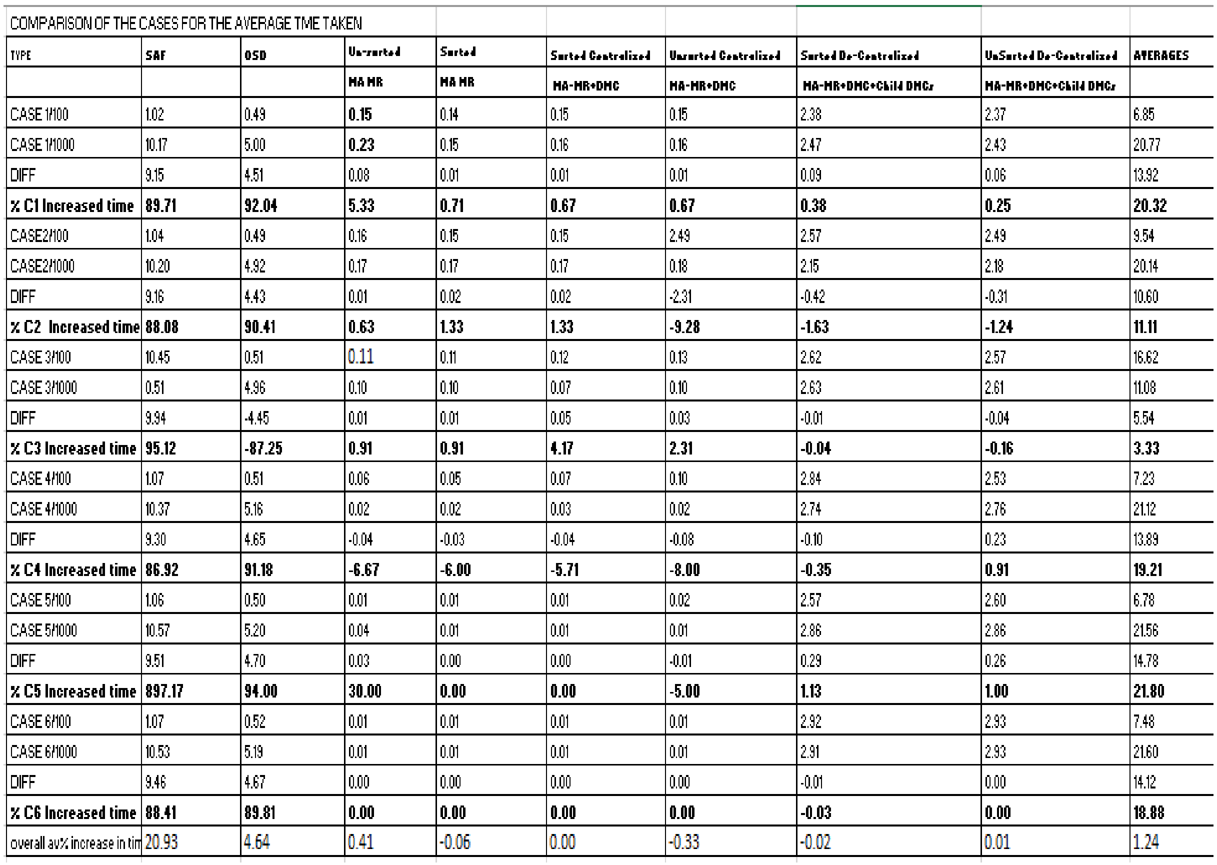
Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+Child DMCs	Unsorted De-Centralized MA-MR+DMC+ Child DMCs	Total File Size (Bytes)
AV.TT(ms) 1000	10.57	5.20	0.01	0.01	0.01	0.01	2.86	2.86	$3.42X\ 10^4$
Av.TT(ms) 100	1.06	0.50	0.04	0.01	0.01	0.02	2.57	2.60	$3.42X\ 10^4$
Throughput MB/s 1000	0.01	0.01	13.41	13.63	13.37	13.53	13.39	13.69	$3.42X\ 10^4$
Throughput MB/s 100	0.01	0.01	12.77	12.93	12.96	12.95	12.48	12.84	$3.42X\ 10^4$
AV.Latency(ms) 1000	15.72	7.72	1.61	1.62	1.67	1.62	6.92	6.99	$3.42X\ 10^4$
AV.Latency(ms)100	15.68	7.39	14.81	15.14	15.48	15.04	64.26	64.63	$3.42X\ 10^4$

Table 12. CASE 6 CSV Summary: AV. Throughput, Latency and Performance (Single file per SAN Request)-SAN1(20480 Bytes) +SAN2(25600) +SAN3(18432 Bytes): MEDIUM-MEDIUM-LARGE 100/1000 FILE REQUESTS

Parameters	SAF	OSD	Un-sorted MA-MR	Sorted MA-MR	Sorted Centralized MA-MR+DMC	Unsorted Centralized MA-MR+DMC	Sorted De-Centralized MA-MR+DMC+ Child DMCs	Unsorted De-Centralized MA-MR+DMC+Child DMCs	Total File Size (Bytes)
AV.TT(ms) 1000	10.53	5.19	0.01	0.01	0.01	0.01	2.91	2.93	$6.45X\ 10^4$
Av.TT(ms) 100	1.07	0.52	0.01	0.01	0.01	0.01	2.92	2.93	$6.45X\ 10^4$
Throughput MB/s 1000	0.03	0.07	60.85	60.98	61.19	6.72	61.60	61.34	$6.45X\ 10^4$
Throughput MB/s 100	0.03	0.07	58.32	58.20	56.91	57.38	57.29	57.47	$6.45X\ 10^4$
AV.Latency(ms) 1000	15.92	7.73	1.85	1.88	1.83	5.10	7.14	7.32	$6.45X\ 10^4$
AV.Latency(ms)100	16.01	7.72	17.67	26.85	16.88	17.56	68.37	67.69	$6.45X\ 10^4$

5.5.1. Effect on Time Taken the Csv File Analysis

Table 13. Comparison of all the cases for the average time taken



COMPARISON OF THE CASES FOR THE AVERAGE TME TAKEN

TYPE	SAF	OSD	Un-sorted	Sorted	Sorted Centralized	Unsorted Centralized	Sorted De-Centralized	UnSorted De-Centralized	AVERAGES
			MA MR	MA MR	MA-MR+DMC	MA-MR+DMC	MA-MR+DMC+Child DMCs	MA-MR+DMC+Child DMCs	
CASE 1/100	1.02	0.49	**0.15**	0.14	0.15	0.15	2.38	2.37	6.85
CASE 1/1000	10.17	5.00	**0.23**	0.15	0.16	0.16	2.47	2.43	20.77
DIFF	9.15	4.51	0.08	0.01	0.01	0.01	0.09	0.06	13.92
% C1 Increased time	**89.71**	**92.04**	**5.33**	**0.71**	**0.67**	**0.67**	**0.38**	**0.25**	**20.32**
CASE2/100	1.04	0.49	0.16	0.15	0.15	2.49	2.57	2.49	9.54
CASE2/1000	10.20	4.92	0.17	0.17	0.17	0.18	2.15	2.18	20.14
DIFF	9.16	4.43	0.01	0.02	0.02	-2.31	-0.42	-0.31	10.60
% C2 Increased time	**88.08**	**90.41**	**0.63**	**1.33**	**1.33**	**-9.28**	**-1.63**	**-1.24**	**11.11**
CASE 3/100	10.45	0.51	0.11	0.11	0.12	0.13	2.62	2.57	16.62
CASE 3/1000	0.51	4.96	0.10	0.10	0.07	0.10	2.63	2.61	11.08
DIFF	9.94	-4.45	0.01	0.01	0.05	0.03	-0.01	-0.04	5.54
% C3 Increased time	**95.12**	**-87.25**	**0.91**	**0.91**	**4.17**	**2.31**	**-0.04**	**-0.16**	**3.33**
CASE 4/100	1.07	0.51	0.06	0.05	0.07	0.10	2.84	2.53	7.23
CASE 4/1000	10.37	5.16	0.02	0.02	0.03	0.02	2.74	2.76	21.12
DIFF	9.30	4.65	-0.04	-0.03	-0.04	-0.08	-0.10	0.23	13.89
% C4 Increased time	**86.92**	**91.18**	**-6.67**	**-6.00**	**-5.71**	**-8.00**	**-0.35**	**0.91**	**19.21**
CASE 5/100	1.06	0.50	0.01	0.01	0.01	0.02	2.57	2.60	6.78
CASE 5/1000	10.57	5.20	0.04	0.01	0.01	0.01	2.86	2.86	21.56
DIFF	9.51	4.70	0.03	0.00	0.00	-0.01	0.29	0.26	14.78
% C5 Increased time	**897.17**	**94.00**	**30.00**	**0.00**	**0.00**	**-5.00**	**1.13**	**1.00**	**21.80**
CASE 6/100	1.07	0.52	0.01	0.01	0.01	0.01	2.92	2.93	7.48
CASE 6/1000	10.53	5.19	0.01	0.01	0.01	0.01	2.91	2.93	21.60
DIFF	9.46	4.67	0.00	0.00	0.00	0.00	-0.01	0.00	14.12
% C6 Increased time	**88.41**	**89.81**	**0.00**	**0.00**	**0.00**	**0.00**	**-0.03**	**0.00**	**18.88**
overall av% increase in tim	20.93	4.64	0.41	-0.06	0.00	-0.33	-0.02	0.01	1.24

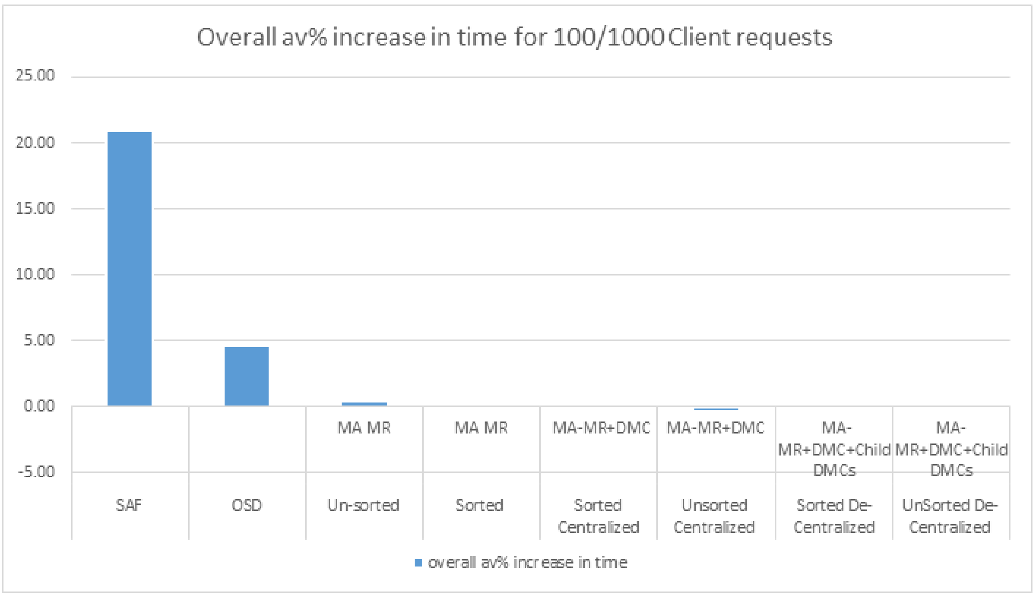


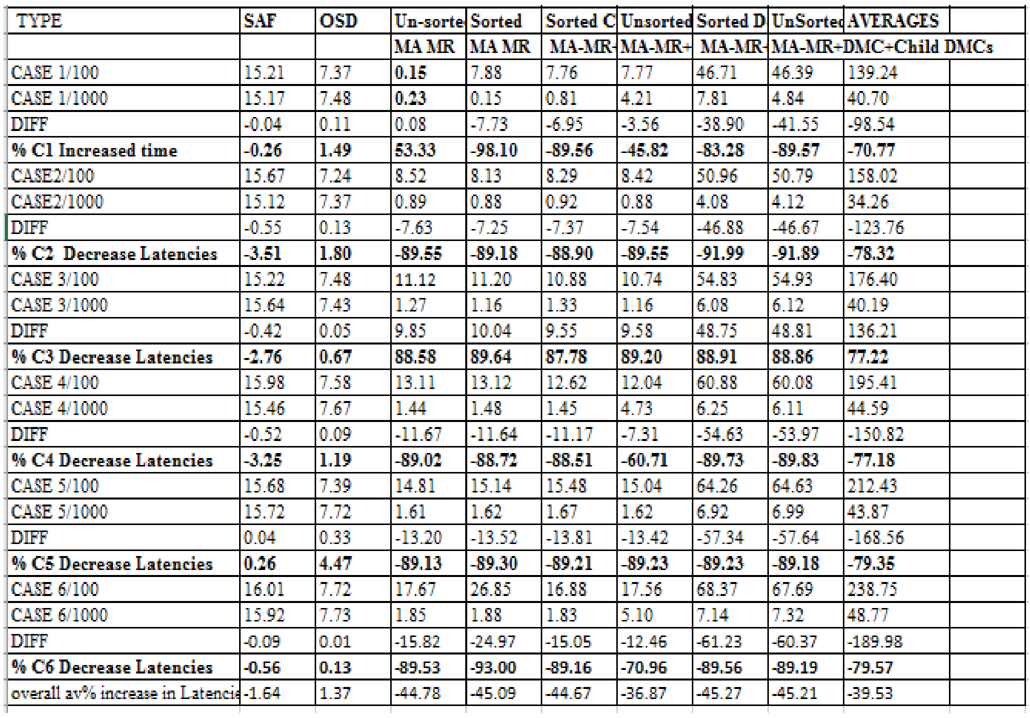
Figure 4. Bar charts showing average increase in time for 100 and 1000 client requests

Figure 4 above shows a summary of the overall individual percentage averages,for Case 1 to Case 6, of how the time is affected when the client requests increased from 100 the initial number of requests to 1000 requests for each identified method in the SABSA engine. SAF has the largest time difference at at 20.9% more time followed by OSD at 4.64 % ,but the agent based and map reduce based objects have insgnificant change in time in servicing this request.

5.5.2. Effect on Latencies for the Csv File Analysis

5.5.2.1. Summary Latencies Analysis

Table 14. Comparison of all the cases for the latencies



TYPE	SAF	OSD	Un-sorte	Sorted	Sorted C	Unsorted	Sorted D	UnSorted	AVERAGES	
			MA MR	MA MR	MA-MR-	MA-MR+	MA-MR-	MA-MR+DMC+Child DMCs		
CASE 1/100	15.21	7.37	**0.15**	7.88	7.76	7.77	46.71	46.39	139.24	
CASE 1/1000	15.17	7.48	**0.23**	0.15	0.81	4.21	7.81	4.84	40.70	
DIFF	-0.04	0.11	0.08	-7.73	-6.95	-3.56	-38.90	-41.55	-98.54	
% C1 Increased time	**-0.26**	**1.49**	**53.33**	**-98.10**	**-89.56**	**-45.82**	**-83.28**	**-89.57**	**-70.77**	
CASE2/100	15.67	7.24	8.52	8.13	8.29	8.42	50.96	50.79	158.02	
CASE2/1000	15.12	7.37	0.89	0.88	0.92	0.88	4.08	4.12	34.26	
DIFF	-0.55	0.13	-7.63	-7.25	-7.37	-7.54	-46.88	-46.67	-123.76	
% C2 Decrease Latencies	**-3.51**	**1.80**	**-89.55**	**-89.18**	**-88.90**	**-89.55**	**-91.99**	**-91.89**	**-78.32**	
CASE 3/100	15.22	7.48	11.12	11.20	10.88	10.74	54.83	54.93	176.40	
CASE 3/1000	15.64	7.43	1.27	1.16	1.33	1.16	6.08	6.12	40.19	
DIFF	-0.42	0.05	9.85	10.04	9.55	9.58	48.75	48.81	136.21	
% C3 Decrease Latencies	**-2.76**	**0.67**	**88.58**	**89.64**	**87.78**	**89.20**	**88.91**	**88.86**	**77.22**	
CASE 4/100	15.98	7.58	13.11	13.12	12.62	12.04	60.88	60.08	195.41	
CASE 4/1000	15.46	7.67	1.44	1.48	1.45	4.73	6.25	6.11	44.59	
DIFF	-0.52	0.09	-11.67	-11.64	-11.17	-7.31	-54.63	-53.97	-150.82	
% C4 Decrease Latencies	**-3.25**	**1.19**	**-89.02**	**-88.72**	**-88.51**	**-60.71**	**-89.73**	**-89.83**	**-77.18**	
CASE 5/100	15.68	7.39	14.81	15.14	15.48	15.04	64.26	64.63	212.43	
CASE 5/1000	15.72	7.72	1.61	1.62	1.67	1.62	6.92	6.99	43.87	
DIFF	0.04	0.33	-13.20	-13.52	-13.81	-13.42	-57.34	-57.64	-168.56	
% C5 Decrease Latencies	**0.26**	**4.47**	**-89.13**	**-89.30**	**-89.21**	**-89.23**	**-89.23**	**-89.18**	**-79.35**	
CASE 6/100	16.01	7.72	17.67	26.85	16.88	17.56	68.37	67.69	238.75	
CASE 6/1000	15.92	7.73	1.85	1.88	1.83	5.10	7.14	7.32	48.77	
DIFF	-0.09	0.01	-15.82	-24.97	-15.05	-12.46	-61.23	-60.37	-189.98	
% C6 Decrease Latencies	**-0.56**	**0.13**	**-89.53**	**-93.00**	**-89.16**	**-70.96**	**-89.56**	**-89.19**	**-79.57**	
overall av% increase in Latenci	-1.64	1.37	-44.78	-45.09	-44.67	-36.87	-45.27	-45.21	-39.53	

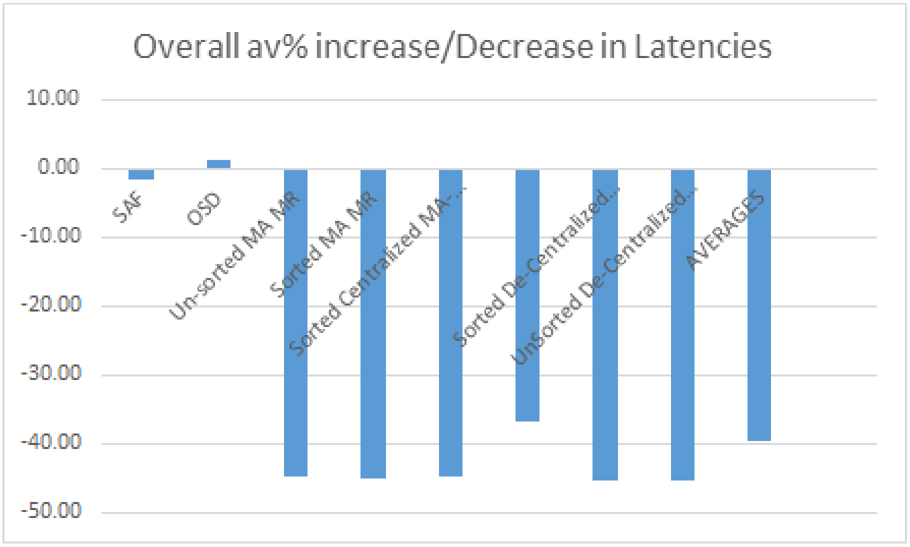


Figure 5. Bar charts showing average increase/decrease in overall latencies for 100/1000 client requests

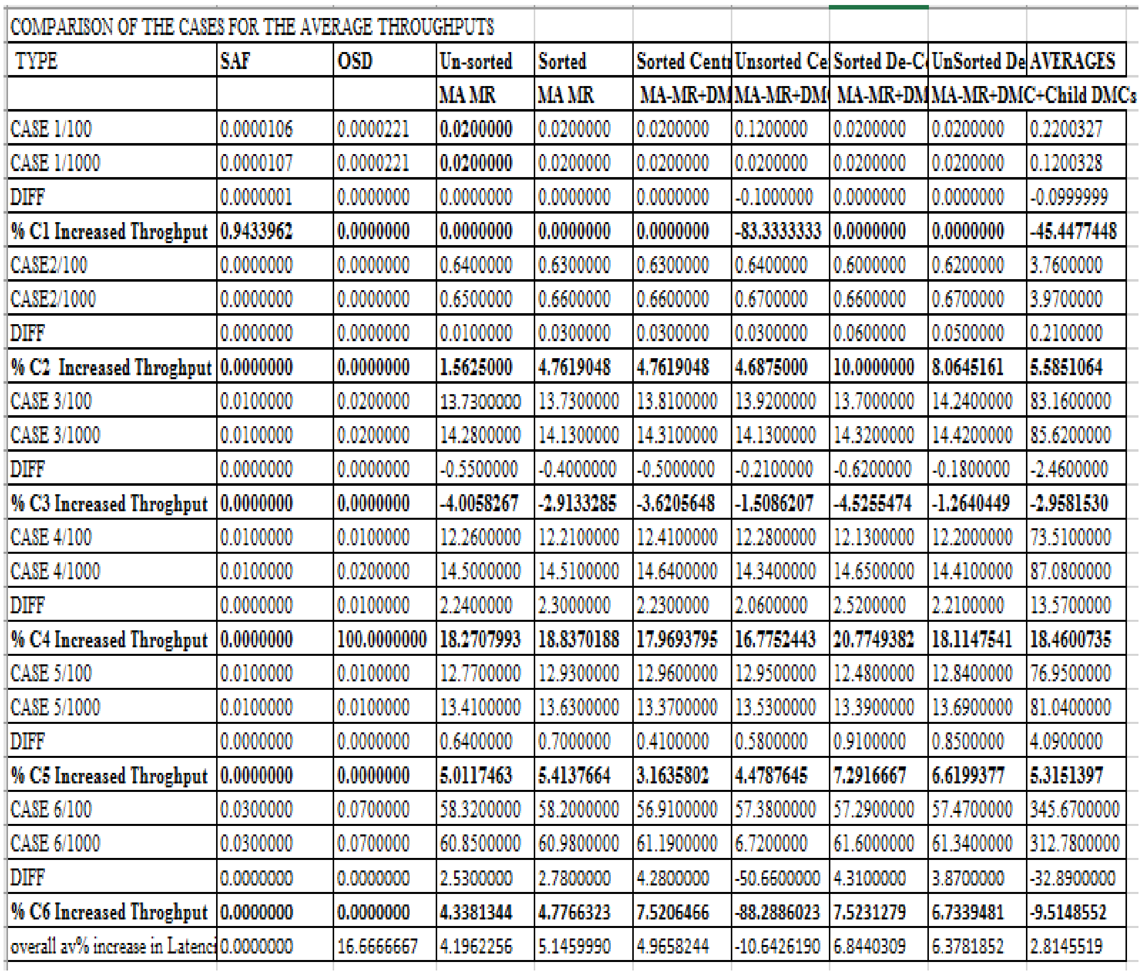
Figure 5 above shows a bar chart for the latencies. The outputs show both positive and negative outputs. The negative ouputs indicate better utilization of the system by minimizing latencies which are a penalty to system performance. The positive values signify increase in latencies which impedes system perfomance. The margins above are also represented as percentage reduction of the overall considered methods in Figure 5 above.

This section has demonstrated various scenarios of viewing the data generated by the SABSA engine using the bar graphs and pie charts and line graphs; various cases generated by the decision tree were generated and captured into the decision matrix where a few random cases were chosen, indicated as Case 1-Case 6 to demonstrate the performance of the SABSA Engine under various load capacities.

5.5.3. Effect on Throughput

5.5.3.1. Summary of Throughput Analysis

Table 15. Comparison of all the cases for the throughput



COMPARISON OF THE CASES FOR THE AVERAGE THROUGHPUTS

TYPE	SAF	OSD	Un-sorted	Sorted	Sorted Centr	Unsorted Ce	Sorted De-C	UnSorted De	AVERAGES
			MA MR	MA MR	MA-MR+DM	MA-MR+DM(	MA-MR+DM	MA-MR+DMC+Child DMCs	
CASE 1/100	0.0000106	0.0000221	**0.0200000**	0.0200000	0.0200000	0.1200000	0.0200000	0.0200000	0.2200327
CASE 1/1000	0.0000107	0.0000221	**0.0200000**	0.0200000	0.0200000	0.0200000	0.0200000	0.0200000	0.1200328
DIFF	0.0000001	0.0000000	0.0000000	0.0000000	0.0000000	-0.1000000	0.0000000	0.0000000	-0.0999999
% C1 Increased Throghput	**0.9433962**	**0.0000000**	**0.0000000**	**0.0000000**	**0.0000000**	**-83.3333333**	**0.0000000**	**0.0000000**	**-45.4477448**
CASE2/100	0.0000000	0.0000000	0.6400000	0.6300000	0.6300000	0.6400000	0.6000000	0.6200000	3.7600000
CASE2/1000	0.0000000	0.0000000	0.6500000	0.6600000	0.6600000	0.6700000	0.6600000	0.6700000	3.9700000
DIFF	0.0000000	0.0000000	0.0100000	0.0300000	0.0300000	0.0300000	0.0600000	0.0500000	0.2100000
% C2 Increased Throghput	**0.0000000**	**0.0000000**	**1.5625000**	**4.7619048**	**4.7619048**	**4.6875000**	**10.0000000**	**8.0645161**	**5.5851064**
CASE 3/100	0.0100000	0.0200000	13.7300000	13.7300000	13.8100000	13.9200000	13.7000000	14.2400000	83.1600000
CASE 3/1000	0.0100000	0.0200000	14.2800000	14.1300000	14.3100000	14.1300000	14.3200000	14.4200000	85.6200000
DIFF	0.0000000	0.0000000	-0.5500000	-0.4000000	-0.5000000	-0.2100000	-0.6200000	-0.1800000	-2.4600000
% C3 Increased Throghput	**0.0000000**	**0.0000000**	**-4.0058267**	**-2.9133285**	**-3.6205648**	**-1.5086207**	**-4.5255474**	**-1.2640449**	**-2.9581530**
CASE 4/100	0.0100000	0.0100000	12.2600000	12.2100000	12.4100000	12.2800000	12.1300000	12.2000000	73.5100000
CASE 4/1000	0.0100000	0.0200000	14.5000000	14.5100000	14.6400000	14.3400000	14.6500000	14.4100000	87.0800000
DIFF	0.0000000	0.0100000	2.2400000	2.3000000	2.2300000	2.0600000	2.5200000	2.2100000	13.5700000
% C4 Increased Throghput	**0.0000000**	**100.0000000**	**18.2707993**	**18.8370188**	**17.9693795**	**16.7752443**	**20.7749382**	**18.1147541**	**18.4600735**
CASE 5/100	0.0100000	0.0100000	12.7700000	12.9300000	12.9600000	12.9500000	12.4800000	12.8400000	76.9500000
CASE 5/1000	0.0100000	0.0100000	13.4100000	13.6300000	13.3700000	13.5300000	13.3900000	13.6900000	81.0400000
DIFF	0.0000000	0.0000000	0.6400000	0.7000000	0.4100000	0.5800000	0.9100000	0.8500000	4.0900000
% C5 Increased Throghput	**0.0000000**	**0.0000000**	**5.0117463**	**5.4137664**	**3.1635802**	**4.4787645**	**7.2916667**	**6.6199377**	**5.3151397**
CASE 6/100	0.0300000	0.0700000	58.3200000	58.2000000	56.9100000	57.3800000	57.2900000	57.4700000	345.6700000
CASE 6/1000	0.0300000	0.0700000	60.8500000	60.9800000	61.1900000	6.7200000	61.6000000	61.3400000	312.7800000
DIFF	0.0000000	0.0000000	2.5300000	2.7800000	4.2800000	-50.6600000	4.3100000	3.8700000	-32.8900000
% C6 Increased Throghput	**0.0000000**	**0.0000000**	**4.3381344**	**4.7766323**	**7.5206466**	**-88.2886023**	**7.5231279**	**6.7339481**	**-9.5148552**
overall av% increase in Latenci	0.0000000	16.6666667	4.1962256	5.1459990	4.9658244	-10.6426190	6.8440309	6.3781852	2.8145519

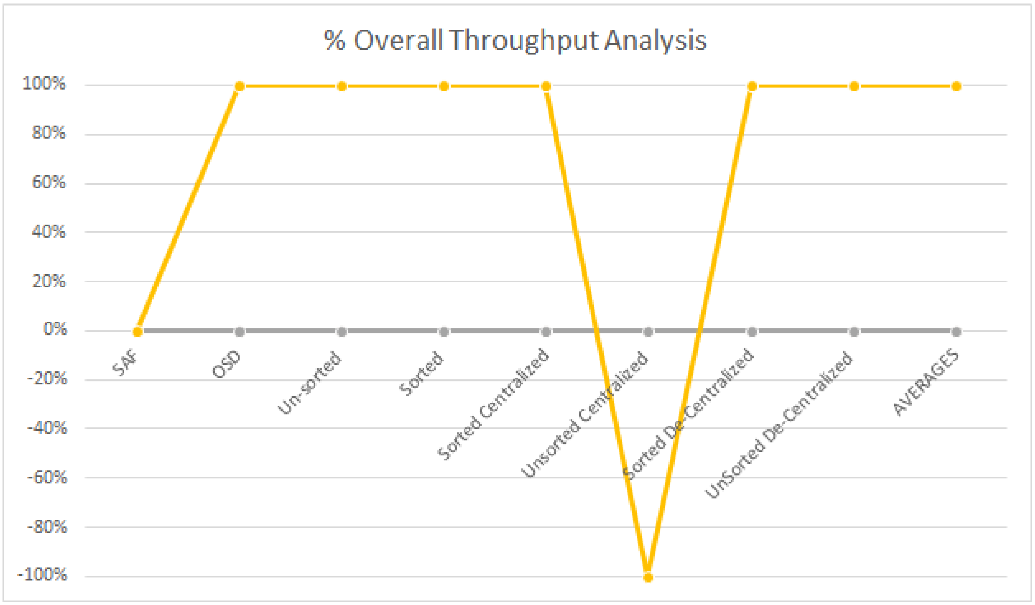


Figure 6. A Line graph showing average overall throuput 100-1000 client requests

Figure 6 is a summary Line graph of the Comparison of all the cases for the throughput in table 15 above ; it shows that store and forward (SAF) has the lowest throughput at 0% and sorting of metadata sorting of metadata and introducing an agent also has a positive impact in increasing throughput of a system performing at maximum throughput of 100% for OSD and all agent based methods except the sorted decentralized whose throughput drastically drops and then resume back to maximum throughput after some time.

6. Conclusions and Further Work

As observed from the previous analysis, all cases indicate that sorting of metadata and caching it will make this system faster than their counter-parts with centralized metadata. In conclusion our experiments tend to concur with Amidal's law that splitting a system into sub systems improves the performance of such a system up to a certain limit.

Mobile agents play a key role in contributing to the performance improvement of the distributed system environment as has been indicated in the previous observations. Mobile agents can autonomously move from one place to another with metadata and security being guaranteed. This research question led to discovery of new tools like SPADE framework for agent design within the python programming environment this greatly contributed to implementation of mobile agents in with the Docker containers.

Since mobile agents is a new concept, and virtualization and big data are emerging trends, this technology will be important in defining and re-defining such research directions. Mobile agents can also be applied in the study of Internet of things that greatly relies on virtualization technologies and therefore offload the virtual server from the mundane work of security, load balancing and job tracking.

Map reduce help improve locality of reference of the metadata functions with Key, value pairs for target storage domains being shuffled together and then the agent caches this sorted metadata domains which consequently leads to shorter access paths and thus minimization of latencies and consequently increasing performance of such systems.

Further work will be carried as follows:

1) Extend the concept of Mobile Agent (MA) based virtualization to the field of Artificial Intelligence (AI) and Artificial Neural networks (ANNs); in order to solve the exponential data requirements in the IOT systems. ANNs will provide multiple distributed nodes that will communicate with other peers within a given domain and eventually transfer the processed data the distributed virtual child nodes or domains to the parent Nodes/Domains and eventually to the parent nodes/domain for final storage or processing.
2) To improve on our SABSA Test-Bed simulator to a more advanced simulator with an adaptable API-to allow for testing of applications-; including organizations for testing their storage requirements in regards to: scalability, Latencies and throughput.
3) To study and implement advanced security fencing systems within our agent-based storage architecture.

REFERENCES

[1] Al-shishtawy, A. (2012) Self-Management for Large-Scale Distributed Systems.

[2] Alberola, J. M. et al. (2010) 'A performance evaluation of three Multiagent Platforms', Artificial Intelligence Review, 34(2), pp. 145–176. doi: 10.1007/s10462-010-9167-9.

[3] Amazon (2019) 10-Minute Tutorials. Available at: https://aws.amazon.com/getting-started/tutorials/.

[4] Andrei, P. S. et al. (2014) 'Evolution towards Distributed Storage in a Nutshell', pp. 1267–1274.

[5] Anon (2016) Concordia White paper. Available at: https://www.cis.upenn.edu/bcpierce/629/papers/Concordia-Whitepaper/ (Accessed: 17 March 2016).

[6] Arias (2018) Introduction to Redis:Installation,CLI commands and Data-Types.

[7] Avilés-González, A., Piernas, J. and González-Férez, P. (2014) 'Scalable metadata management through OSD+ devices', International Journal of Parallel Programming, 42(1), pp. 4–29. doi: 10.1007/s10766-012-0207-8.

[8] Banks, C. M. (no date) 'Principles of Modeling and Simulation: A Multidisciplinary Approach'.

[9] Caidi, M. et al. (2008) 'The Google File System Sanjay', Journal de Chirurgie, 145(3), pp. 298–299. doi: 10.1016/S0021-7697(08)73776-1.

[10] Ceph (2016) Welcome to Ceph. Available at: http://docs.ceph.com/docs/master/# (Accessed: 30 April 2019).

[11] Chaturvedi, V. (no date) Deep Dive into Docker. Available at: https://www.edureka.co/blog/what-is-docker-container (Accessed: 26 March 2019).

[12] CORP (2016) Content addressed storage systems, EMC. Available at: http:www.emc.com/products/systems/centera.jsp?openfolder=platform (Accessed: 26 June 2016).

[13] Docker (2019) Docker Docs. Available at: https://docs.docker.com/v17.09/compose/install/ (Accessed: 27 March 2019).

[14] EMC2 (2008) Where information lives:current benefit and future potential technology concepts and business considerations.

[15] Escriv, M., C, J. P. and Bada, G. A. (2014) 'A Jabber-based Multi-Agent System Platform ∗', (January 2006). doi: 10.1145/1160633.1160866.

[16] Escriva, R. and Wong, B. (no date) 'Http://Hyperdex.Org/Papers/Hyperdex.Pdf', Hyperdex. Org. Available at: http://hyperdex.org/papers/hyperdex.pdf %5Cnpapers2://publication/uuid/7E524955-B159-492D-B 9E4-F52C5E1BAE79.

[17] Factor, M. et al. (2006) 'Object Storage: The Future Building Block for Storage Systems A Position Paper', pp. 119–123. doi: 10.1109/lgdi.2005.1612479.

[18] Feng, D. et al. (2004) 'Enlarge Bandwidth of Multimedia Server with Network Attached Storage System 3 The Redirection of Data Transfer', pp. 489–492.

[19] Finin, T. (1992) An Overview of KQML : A Knowledge Query and Manipulation Language.

[20] FIPA (2000) 'Foundation for Intelligent Physical Agents', Inform.

[21] FIPA (2002) 'FIPA Abstract Architecture Specification (SC00001L)', p. 75.

[22] FullStack (no date) Full Stack Python,Redis. Available at: https://www.fullstackpython.com/redis.html (Accessed: 28 March 2019).

[23] Gibson, G. A. et al. (2001) 'A cost-effective, high-bandwidth storage architecture', High Performance Mass Storage and Parallel I/O: Technologies and Applications, (May 2014), pp. 431–444. doi: 10.1109/9780470544839.ch28.

[24] Griffit (2018) how-build-hello-redis-with-python, Opensource.com.

[25] Hanemann, A. et al. (2006) 'A study on network performance metrics and their composition', Campus-Wide Information Systems, 23(4), pp. 268–282. doi: 10.1108/10650740610704135.

[26] Hendricks, J. et al. (2006) 'Improving small file performance in object-based storage', (May).

[27] Hitachi (2016) Storage virtualisation:How to capitalize on its economic benefits.

[28] Iii, W. B. L. and Ross, R. B. (2000) '4th Annual Linux Showcase & Conference, Atlanta PVFS : A Parallel File System for Linux Clusters £'.

[29] James (2006) 'Improving small file performance in object based storage.', CMU-PDL-06-104.

[30] James, J. (no date) '"Cassandra"', Notes and Queries, s2-X(241), p. 111. doi: 10.1093/nq/s2-X.241.111-a.

[31] Kabir, M. H. et al. (2014) 'Detail Comparison of Network Simulators', (November). doi: 10.13140/RG.2.1.3040.912 8.

[32] Kang, S. J., Lee, S. Y. and Lee, K. M. (2015) 'Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems', Advances in Multimedia, 2015, pp. 1–9. doi: 10.1155/2015/575687.

[33] Karakoyunlu, C. et al. (2013) 'Toward a Unified Object Storage Foundation for Scalable Storage Systems'.

[34] Kasireddy, P. (2016) A Beginner-Friendly Introduction to Containers, VMs and Docker. Available at: https://medium .freecodecamp.org/a-beginner-friendly-introduction-to-con tainers-vms-and-docker-79a9e3e119b?gi=a26c3acc92c1 (Accessed: 26 March 2018).

[35] Kaur, K. and Rai, A. K. (2014) 'A Comparative Analysis : Grid, Cluster and Cloud Computing', 3(3), pp. 5730–5734.

[36] Lange, D. B. (1998) 'Mobile objects and mobile agents: The future of distributed computing?', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1445, pp. 1–12. doi: 10.1007/BFb0054084.

[37] Lehner, W. (2013) Web-Scale Data Management for the Cloud.

[38] Li, G. et al. (2006) 'Researches on Performance Optimization of Distributed Integrated System Based on Mobile Agent *', pp. 4038–4041.

[39] Liancheng, X. U. (2014) 'Research on Distributed Data Stream Mining in Internet of Things', (Lemcs).

[40] Liu, X. et al. (2015) 'Meta-MapReduce for scalable data mining', Journal of Big Data. Journal of Big Data. doi: 10.1186/s40537-015-0021-4.

[41] Luck, M., McBurney, P. and Preist, C. (2003) 'Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)'. Available at: http://eprints.soton.ac.uk/257309/.

[42] Maitrey, S. (2015) 'Handling Big Data Efficiently by using Map Reduce Technique'. doi: 10.1109/CICT.2015.140.

[43] Mark et al. (2000) Storage Virtualisation, What is it all about?

[44] McCanne, S., Vetterli, M. and Jacobson, V. (1997) 'Low-complexity video coding for receiver-driven layered multicast', IEEE Journal on Selected Areas in Communications, 15(6), pp. 983–1001. doi: 10.1109/49.61 1154.

[45] Mesnier, M. et al. (2003) '01222722', (August), pp. 84–90.

[46] Microsystems, S. (2007) 'LUSTRE TM FILE SYSTEM', (December).

[47] Miller, E. L., Freeman, W. E. and Reed, B. C. (2002) 'Proceedings of the FAST 2002 Conference on File and Storage Technologies Strong Security for Network-Attached Storage', Access. Available at: http://www.usenix.org.

[48] Mishra, A. (2012) 'Application of Mobile Agent in Distributed Network Management'. doi: 10.1109/CSNT.20 12.198.

[49] Mohammed, E. A., Far, B. H. and Naugler, C. (2014) 'Applications of the MapReduce programming framework to clinical big data analysis : current landscape and future trends', 7(1), pp. 1–23. doi: 10.1186/1756-0381-7-22.

[50] Moniem, H. A. and Ammar, H. H. (2015) 'A framework for Performance Prediction of Service-Oriented Architecture', International Journal of Computer Applications Technology and Research, 4(11), pp. 865–870. doi: 10.7753/ijcatr0411.1013.

[51] 'MSST-Cabrera' (1991) 'A storage Architecture for large objects'.

[52] Mwathi, D. G. (2018) 'A model based approach for implimenting Authentication and access control in public WLANS: A CASE OF UNIVERSITIES IN KENYA', Director CSI, 15(2), pp. 2017–2019. doi:10.22201/fq.18708404e.2004.3.66178.

[53] Oracle, S. (2011) 'Lustre Software Release 2. x Operation Manual'.

[54] Osero, B. O. (2010) Storage virtualisation and management. University of Nairobi.

[55] Osero, B. O. (2013) 'NETWORK STORAGE VIRTUALISATION AND MANAGEMENT BENARD ONG ' ERA OSERO LECTURER Network Attached Devices , Storage virtualization , Security .', International Journal of Education and Research, 1(12), pp. 1–10.

[56] Oussous, A. et al. (2009) 'Comparison and Classification of NoSQL Databases for Big Data'.

[57] Outcomes, L. (no date) 'Understanding research philosophies and approaches', pp. 2–30.

[58] Palanca, J. (2018) 'SPADE Documentation'.

[59] Panasas, I. (2016) 'Panasas', Wikipedia. Available at: https://en.wikipedia.org/wiki/Panasas.

[60] Patel, A. B., Birla, M. and Nair, U. (2012) 'Addressing Big Data Problem Using Hadoop and Map Reduce', pp. 6–8.

[61] Pedro Jos´e Marr ´on, Stamatis Karnouskos, D. M. A. O. and the C. consortium (2011) No Title.

[62] Permabit 'Permabit', (2015), Wikipedia. Available at: https://en.wikipedia.org/wiki/.

[63] Rajguru, P. (2011) 'Available Online at www.jgrcs.info ANALYSIS OF MOBILE AGENT', Journal of Global Research in Computer Science, 2(11), pp. 6–10. Available at: www.jgrcs.info.

[64] Randy, Fellows, A. R. and Kerns, R. (2012) 'SAN Virtualization Evaluation Guide', p. 2.

[65] Rfc, T., Rfc, T., et al. (no date) 'Ethernet RFC-2544 expained', pp. 1–9.

[66] Rfc, T., Engineering, I., et al. (no date) 'The RFC 2544 Application – Performance Benchmarking for the HaulPass V60s Link', pp. 1–11.

[67] Riedel, E. and Nagle, D. (1999) 'Active Disks - Remote Execution for Network-Attached Storage Thesis Committee':, Science, (December). Available at: https://pdfs.semanticscholar.org/74ac/0dd0a14ea27f016b170a1254c14fe8c73b37.pdf.

[68] Rodríguez-enríquez, L. R. C. et al. (2015) 'A general perspective of Big Data : applications, tools ', The Journal of Supercomputing. Springer US. doi:10.1007/s11227-015-1501-1.

[69] Rugg, G. and Petre, M. (2004) 'The Unwritten Rules of PhD research', Open University Press, p. 241.

[70] Sargent, R. G. (2011) 'Advanced Tutorials: Verification and Validation of Simulation Models', Proceedings of the 2011 Winter Simulation Conference, pp. 183–198.

[71] Sarkar, N. I., Member, S. and Halim, S. A. (2011) 'A Review of Simulation of Telecommunication Networks : Simulators, Classification, Comparison, Methodologies, and Recommendations'.

[72] Satoh, I. (2003) 'Building reusable mobile agents for network management', IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 33(3), pp. 350–357. doi: 10.1109/TSMCC.2003.818944.

[73] Satoh, I. (2004) 'Dynamic Federation of Partitioned Applications', pp. 2–6.

[74] Satoh, I. (2011) 'Mobile Agent Middleware for Dependable Distributed Systems'.

[75] Satoh, I. (2014) 'MapReduce-based Data Processing on IoT', (iThings). doi: 10.1109/iThings.2014.32.

[76] Satoh, I. (2016) 'Agent-based MapReduce Processing in IoT', 1(Icaart), pp. 250–257. doi: 10.5220/0005802102500257.

[77] Satoh, I. and Society, I. C. (2003) 'A Testing Framework for Mobile Computing Software', 29(12), pp. 1112–1121.

[78] Silva, L. M. (1999) 'Optimizing the Migration of Mobile Agents'.

[79] Singavarapu, S. and Hariri, S. (2001) 'S ELF-MANAGING STORAGE SYSTEM – D ESIGN AND EVALUATION 2. Self Managing Storage System (SMSS) Architecture – Overview'.

[80] Smith, R. (1999) '3ULQFLSOHV RI 0RGHOLQJ Fundamental Principles of Model Abstraction', pp. 1–28.

[81] Sowmya, N., Aparna, M. and Tijare, P. (2015) 'An Adaptive Load Balancing Strategy in Cloud Computing based on Map Reduce', (September), pp. 4–5.

[82] Tate, J. et al. (2017) 'Introduction to Storage Area'.

[83] Tekniska, K., Ögskolan, H. and Simsarian, K. T. (2000) 'VETENSKAP OCH KONST Dissertation, March 2000 Computational Vision and Active Perception Laboratory (CVAP)', (March).

[84] Tutorialpoint (no date) REDIS - QUICK GUIDE REDIS - ENVIRONMENT REDIS - DATA TYPES.

[85] Wakefield, R. (2007) 'An Analysis of Quality of Service Metrics and Frameworks in a Grid Computing Environment'.

[86] Wang, J. et al. (2010) 'A Novel Weighted-Graph-Based Grouping Algorithm for Metadata Prefetching A Novel Weighted-Graph-Based Grouping Algorithm for Metadata Prefetching'.

[87] Weil, S. A., Brandt, S. A. and Miller, E. L. (2006) 'CRUSH : Controlled, Scalable, Decentralized Placement of Replicated Data', (November).

[88] Welch, B. et al. (2008) 'White Paper Scalable Performance of the Panasas Parallel File System', Fast 2008, (May), pp. 1–22.

[89] Wetoyi, A. O. (2014) 'UNIVERSITY OF NAIROBI Dynamic Subset Difference Revocation using One Binary Tree AUTHOR Austin Owino Wetoyi Prof William Okelo-Odongo', (December).

[90] Wikipedia (2019) 'Network Performance', Wikipedia. Available at: https://en.wikipedia.org/wiki/Network_performance.

[91] Wu, S. A. I. (2014) 'Distributed Data Management Using MapReduce', 46(3).

[92] Xu, H. and Shatz, S. M. (2001) 'A Design Model for Intelligent Mobile Agent Software Systems', pp. 1–23. Available at: file:///C:/Users/ltturche/Downloads/32bfe51224d170bc42.pdf.

[93] Yazdi, H. T., Fard, A. M. and Akbarzadeh-T, M. R. (2008) 'Cooperative criminal face recognition in distributed web environment', AICCSA 08 - 6th IEEE/ACS International Conference on Computer Systems and Applications, (March), pp. 524–529. doi: 10.1109/AICCSA.2008.4493582.

[94] Yu, P. et al. (2006) 'Mobile Agent Enabled Application Mobility for', pp. 648–657.