Performance Analysis of an Agent Based Architecture using Map-reduce: Using the SABSA Simulator

Benard Ong'era Osero[1], Dr. Elisha Abade[2], Dr. Stephen Mburu[3]
[1]*Department of Computer Science, Chuka University, Kenya*
[2]*School of Computing and Informatics, University of Nairobi, Kenya*
[3]*University of Nairobi, Kenya*

Abstract: **- Increasing performance and decreasing cost of microprocessors are making it feasible to move more processing power to the data source. This allows us to investigate new methods of storage delivery and management of data that were not plausible in the past. Our architecture, inspired by agent-based techniques and active disk technology, promotes an open source agent storage management platform called SPADE that is adopted to implement an agent based simulation model called SABSA. Mobile agent technology and Map-Reduce functionality has been promoted as an emerging technology that makes it much easier to design, implement, and maintain distributed system. In order to Realize the storage technology's full potential requires careful consideration across a wide range of metadata file handling systems and networking issues. This research contrasts four network storage architectures: Store and forward processes(SAF), Object Storage Devices(OSD), Mobile agent Domain Controller (DMC) enhanced with map-reduce function and Mobile agent based Domain Controller with child DMC enhanced with Map-reduce (ABMR): both handling sorted and unsorted metadata. To estimate the potential performance benefits of these architectures, we developed an analytic simulation model and then performed experiments based on the identified storage architectures. Our results suggest that all the agent based storage architectures minimize latencies up to 40 % and OSD architectures and consequently increasing performance in the same margin.**

Keywords: **Store and Forward, Object Storage Devices, Agent, and Map-Reduce.**

I. BACKGROUND

Virtualization is a powerful feature that plays a role in the current success of storage arrays. By design, virtualization manages where data is located and controls access to data for users and applications. The value of storage has moved from disk drives to the array controller as more features and data protection capabilities have been added over time from the array to the point of virtualization(Randy, Fellows and Kerns, 2012).

Applications that can follow mobile users when they change to a different environment, especially with the change of device and location, are in high demand by pervasive computing. Implementation of application mobility also depends on context-awareness and self-adaptation techniques (Yu *et al.*, 2006).

This paper therefore tires to seeks to unearth the deficiencies that impede performance of distibuted Network (Latencies,Scalabity, and throughput) in centralized array based metadata blocks that are either employ physical file storage or virtualized file storage including the store and forward (SAF) file systems and OSD systems, and finally to demonstrate through experimentation how this defiencies can be improved by the use of mobile agents and map reduce functionalities.

II. AGENT BASED DISTRIBUTED NETWORKS

Mobile agents are considered a very interesting technology to develop applications for mobile, pervasive, and distributed computing. Thus, they present a combination of unique features, such as their autonomy and capability to move to remote computers to process data there and save remote communications. Many mobile agent platforms have been developed since the late nineties. While some of them have been outdated, others continue releasing new versions that fix bugs detected or offer new interesting features. Moreover, other new platforms have appeared in the last few years. So, a common problem when one wants to benefit from mobile agent technology to develop distributed applications is the decision about which platform to use (Rajguru, 2011).

A Mobile Agent is an emerging technology that is gaining momentum in the field of distributed computing. The use of mobile agents can bring some interesting advantages when compared with traditional client/server solutions, it can reduce the traffic in the network, it can provide more scalability, it allows the use of disconnected computing and it provides more flexibility in the development and maintenance of the applications. In the latest years, several commercial implementations of mobile agent systems have been presented in the market (Rajguru, 2011).

2.1 Mobile agent-based Map Reduce system

Map Reduce is a computing platform with certain kinds of distributable problems using a cluster consisting of a large number of computers, the original map-reduce consists of

three phases: Map-phase, worker phase and Reduce phase; new models using agents have been suggested where the Mapper agent is a container agent corresponding to the master node in the Map Reduce pattern. It supports multicast coordination and contains at least one worker agent inside it.

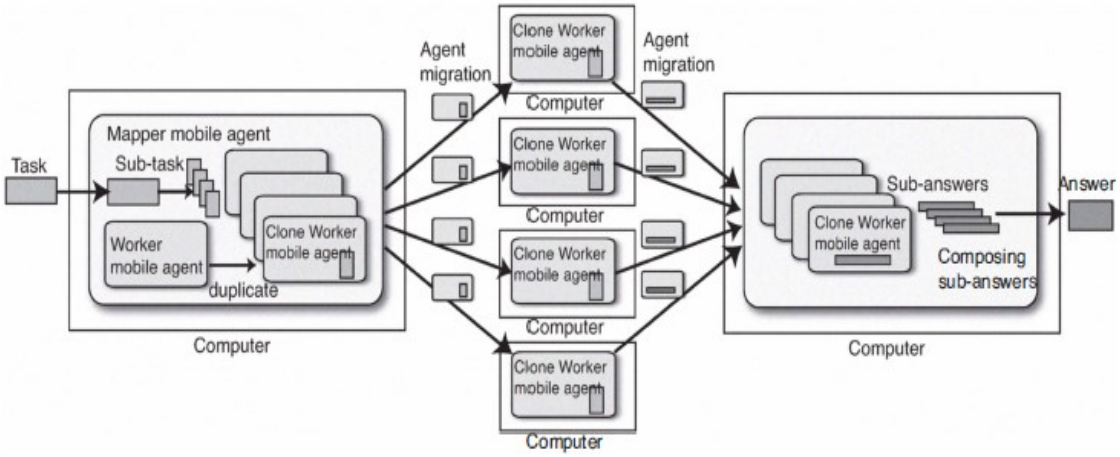


Figure 1: Mobile agent-based Map-Reduce system(Satoh, 2011).

(Satoh, 2014) in Figure 1above, describe a platform for dynamically organizing multiple mobile agents for computing.(Satoh, 2014) has developed a comprehensive model for map-reduce platform employing mobile agents, advancing on the model previously proposed by (Satoh, 2011), also previously demonstrated that agent size has a direct implication on cost.

III. THE VIRTUALIZED SECURE AGENT BASED ARCHITECTURE (SABSA)

To address the gaps that exist between mobile agents and network attached disks that have not yet been fully exploited; a more intelligent, self-managed and secure storage environment has been designed and subsequently used to test the effects of client requests on Scalability, Latencies and throughput as shown in the following model.

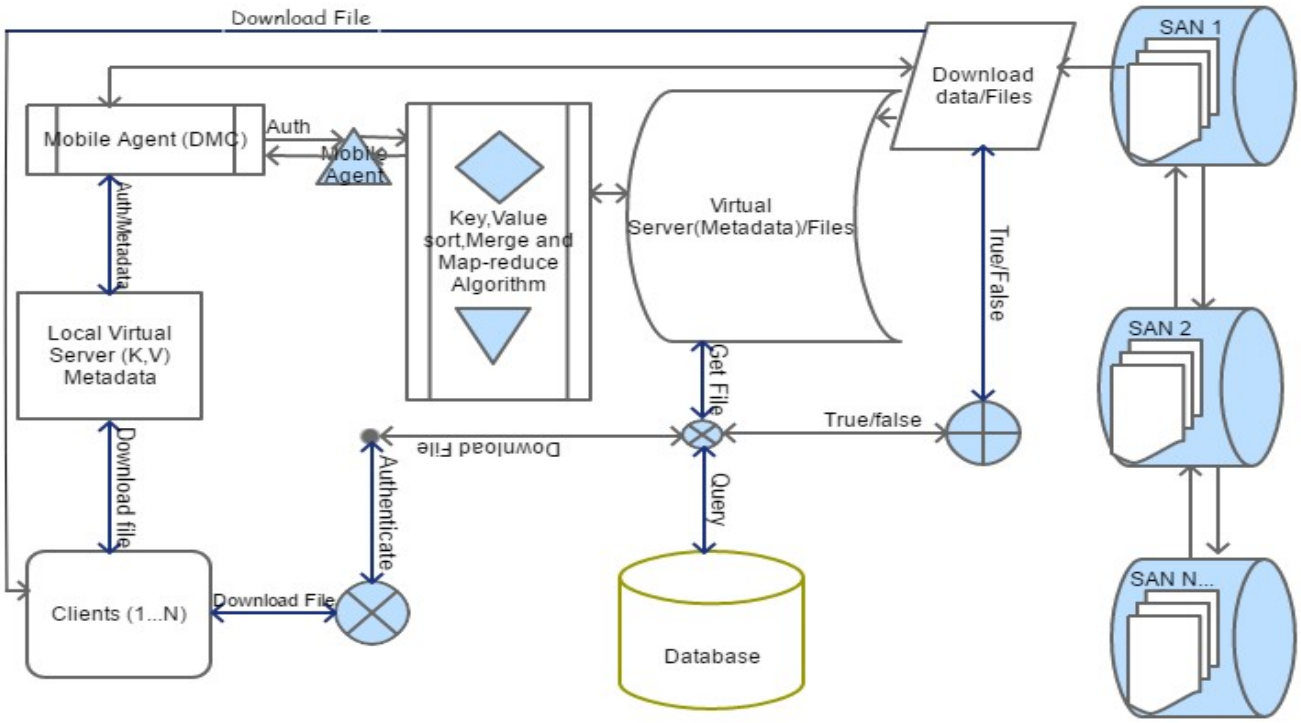


Figure 2: A Conceptual Architecture for intelligent objects using agents and Map-reduce.

Figure 2 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 serve as the middle tier and storage SAN as the backend, it also includes the following functionalities:

i. Storage Area network (SAN)-It is responsible for the storage of the physical files it is implemented as a storage container that has a global IP address to identify the container; included is also the port number and individual internal IP address to identify each internal individual container.
ii. Virtual Server(VS)-It contains the logical implementation of the switching of networks to enable the clients access the metadata. It is also responsible for the authentication of the clients by providing a tokenization mechanism whose capabilities are stored in the database and later mapped onto the storage to allow clients download files.
iii. Client-It is an important aspect of this distributed architecture; It is responsible for requesting for the files and then allowing the clients to view the files through the console or preferred browser interface.
iv. Map-reduce Functions-Responsible for sorting and reducing metadata functions which can then be transported to client side for further processing.
v. Mobile Agent-It is responsible for migrating sorted metadata values from the virtual resource server to the client side.
vi. Local Client-Functions hand in hand with the domain controller, which manages the local switching of clients and keeps a registry of the requested and served metadata requests for each client, it also caches the requests for future access.

This research employed search mechanism to the existing metadata resource storage pool enhanced by the map reduce algorithm that sorted the metadata blocks according to the client IP address domains before mapping them to a mobile agent and eventually migrated to a Domain Controller (DMC).

The mobile agent fetched the sorted metadata (using map-reduce function) pool and migrated them to the one of the selected local servers where they were executed henceforth, this would be terminated if this particular local server terminated normally or it is terminated by the parent server in case the local server used the resource not allocated to it or issues instructions beyond its allocated mandate or a critical unrecoverable event happened.

The clients within a particular domain were then given the resource path indicating where a certain physical resource is located in the storage area network physical disks as long as the requests were valid.

The local server had the potential of enforcing their local security mechanisms to be able to protect the clients within a particular domain. Various experiments were carried out in various phases in order to test performance (Latencies, throughput) and Scalability.

3.1 Simulation Environment Design

The research in this paper was carried out in five phases which included varying of workloads using the SABSA (Secure Agent Based System Architecture) Engine that was designed from scratch using the Docker containers designed within the Python development environment employing the SPADE framework defined within the python Environment to implement mobile agents. The experiments were first categorized into Six Cases as indicated in table 4 and the jobs were then classified as small, medium and large for ALL Phases as shown in the table 1 below. But the object based metadata schemes including; OSD, Mobile agent Domain Controller (DMC) enhanced with map-reduce function, Mobile agent based Domain Controller with child DMC controllers enhanced with Map-reduce **(ABMR)** all their workloads were also further, except store and forward processes, classified into either sorted or unsorted metadata groups.

For easy handling of the files and client requests; the client requests were classified as indicated in, table 1 and 2 below and file load sizes were classified as indicated in table 3:

Table 1: Workload Requests Classifications for OSD.

WORKLOAD TYPE (NO.OF REQUESTS)	BASE NO OF REQUEST(S)	MAX NO. OF REQUESTS
SMALL	1	100
MEDIUM	500	1000
LARGE	5000	10000

Table 2: Workload Requests Classifications for Object based and agent based metadata based models.

WORKLOAD TYPE(NO.OF REQUESTS)	BASE NO OF REQUEST(S)	MAX NO. OF REQUESTS	METADATA SCHEME
SMALL	1	100	Sorted
SMALL	1	100	Unsorted
MEDIUM	500	1000	Sorted
MEDIUM	500	1000	Unsorted
LARGE	5000	10000	Sorted
LARGE	5000	10000	Unsorted

Table 3: Workload classifications for all Phases.

File Size Type	Base File Size(Bytes)	Max File Size (Bytes)
Small	1	100
Medium	101	10000
Large	10001	>=10001

3.2 SAN File Selection Options

Two cases were employed in the workload selection as follows:

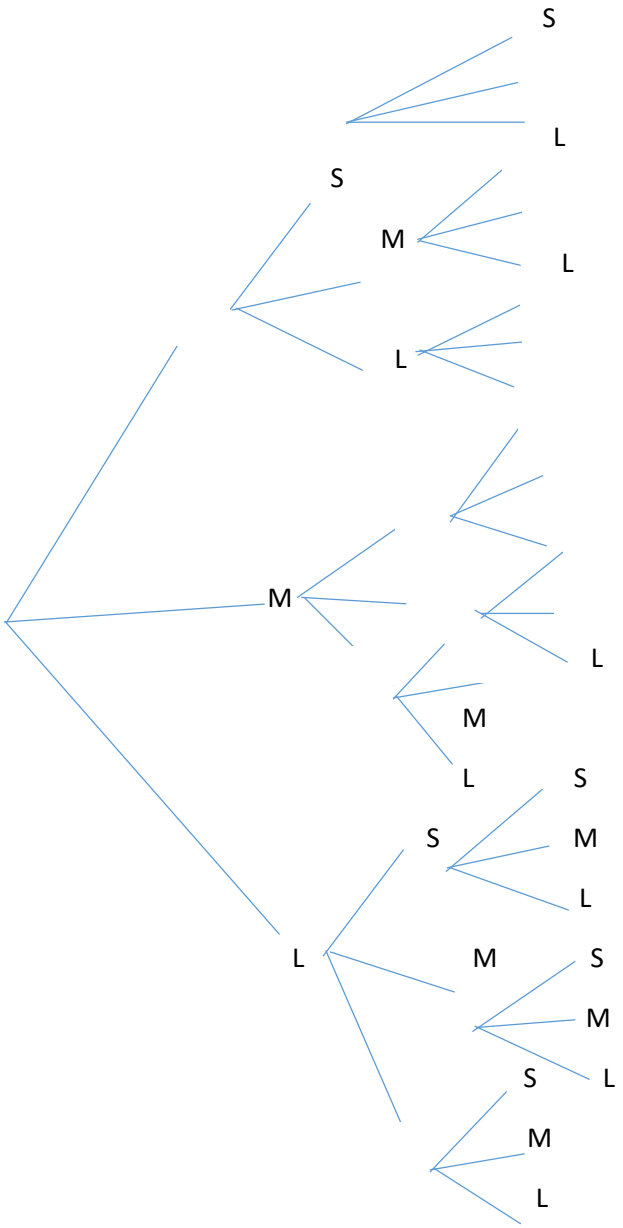
I. Using the individual SAN options

II. Using the file sizes from the individual SANs

Both Case I and II were used in our experiments as follows:

For case I one SAN could be selected and be run with any file(s) to identify the performance and scalability.

For case II above A probability tree shown in figure 3below was used to generate some possible file selection options from the three SANs (SAN 1, SAN2 and SAN 3) supported by the SABSA simulator and a few sample workloads were then randomly selected for testing as indicated in table 4 below:



KEY:S-SMALL, M-MEDIUM, L-LARGE.

Figure 3: File distribution tree

Figure 3 above indicates a decision tree that has been applied to select Jobs from each of the three SANs that were used to impliment the SABSA Engine. Each SAN contains three sets of files classified as S(Small), M(Medium) and L(Large).

Table 4: File size and selection options

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

Key:

x –indicates option not selected.

✓ -Option was selected.

Once the requests had been made the time taken for each of the above file request variations was then generated as a CSV file sliced and output on the Microsoft Excel Sheet and corresponding charts were generated by the simulator, which were then be used to automatically calculate latencies, throughput and scalability (i.e. size of file in bytes against time in ms).The importance of the metrics that were used as basic performance measures in the SABSA Simulator is shown in (Pedro Jos´e Marr ´on, Stamatis Karnouskos, 2011) and (Andrei *et al.*, 2014)who demonstrated the importance of such metrics as shown in figure 4 below:

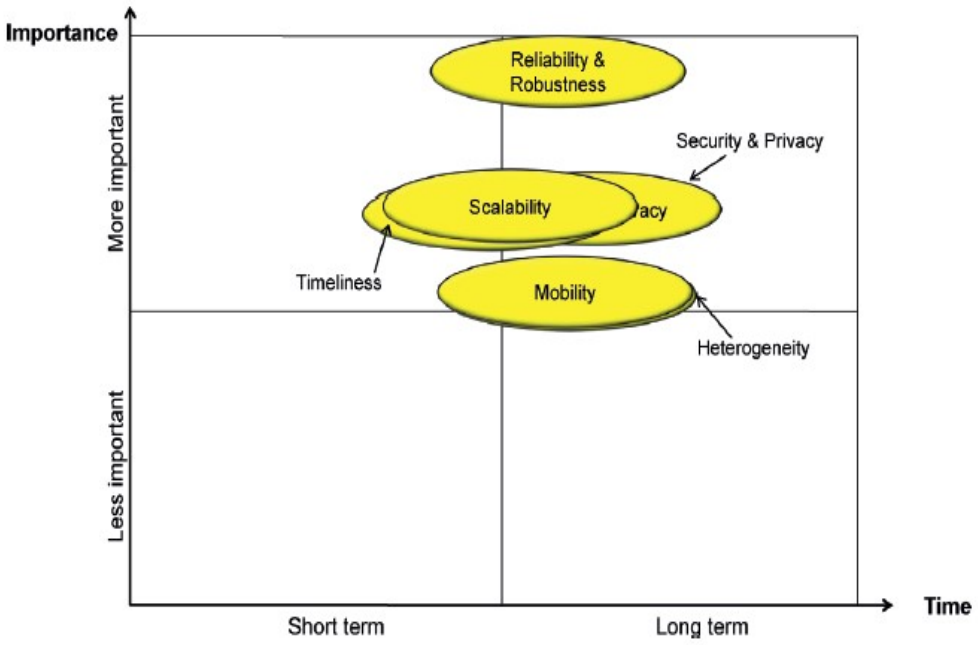


Figure 4:A matrix showing the order of importance of Non-functional system properties (Pedro Jos´e Marr ´on, Stamatis Karnouskos, 2011).

In our research Scalability and timeliness were our biggest focus.

3.3 Setting timings for the simulator

There were three parameters were used to measure (Throughput, Latency and Scalability) the performance SABSA ENGINE:

3.3.1 Throughput

Although the theoretical peak bandwidth of a network connectionis fixed according to the technology used, the actual amount of data that flows over a connection (called *throughput*) varies over time and is affected by higher and lower latencies. Excessive latency creates bottlenecks that prevent data from filling the network pipe, thus decreasing throughput and limiting the maximum effective bandwidth of a connection. The impact of latency on network throughput can be temporary (lasting a few seconds) or persistent (constant) depending on the source of the delays. Throughput was calculated as a function of latency as shown in the code below

3.3.2 Latency

Latency refers to any of several kinds of delays typically incurred in processing of network data. A so-called **low latency** network connection is one that experiences small delay times, while a **high latency** connection suffers from long delays. In our SABSA Engine; Latency was calculated as **Propagation delay+Serialisation time** as shown in the code-section below. Propagation delay is the length of time taken for the quantity of interest to reach its destination and in the context of data storage, **serialization** (or **serialization**) is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later. Latency and throughput were implemented in the following code section:

```
def get_time ():
    return time. perf_counter ()
def calculate throughput (latency, file_size_in_bytes):
    totalFileSizeMb = (file_size_in_bytes / (1024 * 2)) * 8 // convert bytes to bits
    throughput = totalFileSizeMb / latency
    return throughput
def calculate latency (start_time, time_taken, prev_end_time):
    propagation_delay = (start_time - prev_end_time) * 1000
    serialization_delay = time_taken
    latency = propagation_delay + serialization_delay
    return latency
def get_time_values (start_time, file_size = 0, prev_end_time = 0):
    end_time = get_time ()
    time_taken_sec = end_time - start_time
    time_taken_ms = time_taken_sec * 1000
    latency = calculate_latency (start_time, time_taken_ms, prev_end_time or start_time)
    return end_time, (start_time, end_time, time_taken_ms, latency, calculate_throughput (latency, file_size))
```

3.3.3 Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system. For instance a routing protocol is considered scalable with respect to network size, if the size of the necessary routing table on each node grows as O(log *N*), where *N* is the number of nodes in the network. Scalability was calculated as shown in the Python code-section below:

```
const fileSizeTimeComparisonChartLine = Ranged Chart ({
    elId: 'chart-root-bytes-tt',
    data,
    sliderOptionsFn,
    dataFn (data, [min, max]) {
      return {
        datasets: data.map (({storage_type, intervals, metadata, overall}) => {
          return ({
            label: `${storage_type} ${metadata. path} (${metadata. size} bytes) `,
            fill: false,
            // backgroundColor: myColors.next().value,
            borderColor: myColors.next().value,
            data: chunk50AndMapChunks (intervals, (chunk, i) => {
              if (chunk. length < 1) return
              const [{starting_time: _starting_time}, {ending_time}] = [chunk[0], chunk[chunk.length - 1]];
              // const timeTaken = ending_time - overall. starting_time;
              const timeTaken = ending_time - _starting_time;
              const totalFileSize = metadata. size * chunk. length
              return {
                y: totalFileSize * i,
                x: timeTaken
              }
            }). filter(({x}) => min <= x && x <= max)
          })
        })
      }
    },
    chartOptions: {
      type: 'line',
      options: {
        scales: {
          xAxes: [{
            type: 'linear',
            scaleLabel: {
              display: true,
              labelString: 'Time Taken (ms)'
            }
          }],
          yAxes: [{
            scaleLabel: {
              display: true,
              labelString: 'Bytes downloaded'
            }
          }],
        }
      }
    }
  });
```

3.4 CSV data generator

This is the function that is tasked with capturing the start time, ending time, latency and throughput outputs as shown in the code-section below. Python has inbuilt tools to assist in the graph generation as demonstrated in the Python code-section below.

```
import itertools
import operator
from. constants import METRICS
def gen_csv_text_multi_column (headers, rows):
  return '\n'. join (itertools. chain (
    [','. join(headers)],
    (','. join (map (str, itertools. chain. from_iterable(row))) for row in zip(*rows))
  ))
def gen_csvs_multi (all_intervals_with_headers, columns = []):
  if not columns:
    return ''
  headers = []
  csv_rows = []
```

```
    is_one_column = len(columns) == 1
    column_getter = operator. itemgetter (*map (METRICS. Index, columns))
    for (type_, _error, metadata, _overall), intervals in all_intervals_with_headers:
        headers. extend(f'{metric} {type_} {metadata["name"]} ({metadata["size"]} bytes)' for metric in columns)
        csv_cols = []
        csv_rows. append(csv_cols)
        for interval in intervals:
            column = column_getter(interval)
            csv_cols. append((column,) if is_one_column else column)
    return gen_csv_text_multi_column (headers, csv_rows)
```

3.5 Graph Interface generator

This function allows the graphical outputs to be generated from the above identified variables, it makes interpretation of the data easy to understand shown in the Python code section below the request methods and authentication procedures are in bold:

```
from aiohttp_jinja2 import template
from aiohttp import web
import functools
import json
from concurrent import futures
from client import (
    actions, status, storage_types, addr_to_url, client_addr,
    storage_client, server_client, Server, local_hostname,
    storage_addr
)
from .shared import middlewares
# from .shared. file_downloading import file_request_futures
from .shared. downloaders import file_request_futures
from .shared. utilities. parsers. metadata import flatten_file_metadata, format_times_taken_as_json
from .shared. utilities. Formatting import format_interval_data, format_interval_data_files_combined
from .shared. utilities. encoding import str_to_b64
from .shared. utilities. constants import METRICS
from .shared. validation import validate_file_index, get_number_of_repetitions
def validate_login(form):
    return all (key in form for key in ('username', 'password'))
@template ('index. jinja')
async def index(request):
    """
    This is the view handler for the "/" url.
    : param request: the request object see http://aiohttp.readthedocs.io/en/stable/web_reference.html#request
    : return: context for the template.
    """
    # Note: we return a dict not a response because of the @template decorator
    login_error = None
    form = await request.post ()
    if validate_login(form):
        print ("============================")
        print(form)
        print ("============================")
        client_req = await server_client.post (actions. AUTH_REQUEST, {
            "username": form['username'],
            "password": form['password']
        })
        auth = client_req.get_header('Authorization')
        if auth:
            location = request.query.get ('referrer', '/reports')
            res = web. HTTPFound(location=location)
            res.set_cookie ('Auth', auth, max_age=60)
            raise res
        else:
            login_error = client_req. json_data ()
    return dict (title='Sign In', form=form, login_error=login_error)
HANDLED_TYPES = [
    storage_types. STORE_AND_FORWARD,
    storage_types.OSD,
    storage_types. UNSORTED_AGENTS,
    storage_types. SORTED_AGENTS,
```

```
    storage_types. UNSORTED_AGENTS_WITH_DMC,
    storage_types. SORTED_AGENTS_WITH_DMC,
    storage_types.
UNSORTED_AGENTS_WITH_DMC_MAP_RED,
    storage_types.
SORTED_AGENTS_WITH_DMC_MAP_RED
]
@template ('reports. jinja')
@middlewares. check_token
async def reports_handler (request, auth_token):
    was_processed, csv_file, interval_data, files = await
reports_data_handler (request, True, auth_token)
    context = {
        "title": 'Reports',
        "files": files,
        "storage_types": HANDLED_TYPES,
        "metrics": METRICS,
        "metrics_json": json. dumps(METRICS),
        "query": request. query
    }
    if was_processed:
        context["csv_file"] = str_to_b64(csv_file)
        context["all_intervals"] = interval_data
        context["json_intervals"] = json. dumps(interval_data)
    return context
async def reports_data_handler (request, is_local=False,
auth_token=None):
    metadata = (await server_client.get (actions.GET_FILES,
")). get_header ('Metadata', decode=True)
    files = list(flatten_file_metadata(metadata))
    file_indices_arg = request. query. getall ('files', [])
    file_indices = [file_index for is_index_valid, file_index in
(validate_file_index (index_, files) for index_ in
file_indices_arg) if is_index_valid]
    selected_storage_types = [type_ for type_ in request. query.
getall ('storage_type', []) if type_ in HANDLED_TYPES]
    if file_indices and selected_storage_types:
        num_times =
get_number_of_repetitions(request.query.get('n'))
        with futures.
ThreadPoolExecutor(max_workers=len(selected_storage_type
s) *len(file_indices)) as future_pool:
            all_intervals_iter = await file_request_futures (request,
num_times, auth_token, future_pool, files, file_indices,
selected_storage_types)
            csv_file, interval_data =
format_interval_data_files_combined(all_intervals_iter)
            return web. json_response(interval_data) if not
is_local else (True, csv_file, interval_data, files)
    return web. json_response ({}) if not is_local else (False, '',
[], files)
```

IV. RESULTS

Six cases were randomly selected from a set of probable workloads from three different SANs:

Subdivided into two column sections a) and b): column a) represents 100 client requests and column b) represents 1000 client requests.

Table 4: 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 5 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 6 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 7 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 8 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 9 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	$6.45X\ 10^4$

Table 4 to 9 above indicates a summary of the CSV outputs that were analyzed under workload and 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.

4.1 Analysis of Results

A summary of the CSV averaged summary outputs was captured for each client for Time Taken, Latencies and throughputs as shown in table 10,11 and 12 below. CASE 1to CASE 6 represents our sample test cases for the 100 and 1000 client requests. The Difference abbreviated as "DIFF" represents the difference between the value obtained in the 1000 client requests for each case in (The Time Taken, Latencies and throughputs)-Y. And the value obtained in the 100 client requests for each case in (The Time Taken, Latencies and throughputs)-X. for demonstration purposes this values have been assigned X and Y. The percentage change is the Difference(DIFF)/original value (X) * 100.Which can also be illustrated mathematically as: ((Y-X)/X) *100. This formula applies for all the cases for time taken, latencies and throughput.

4.1.1 Summary Time Taken Analysis

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

Table 10: 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

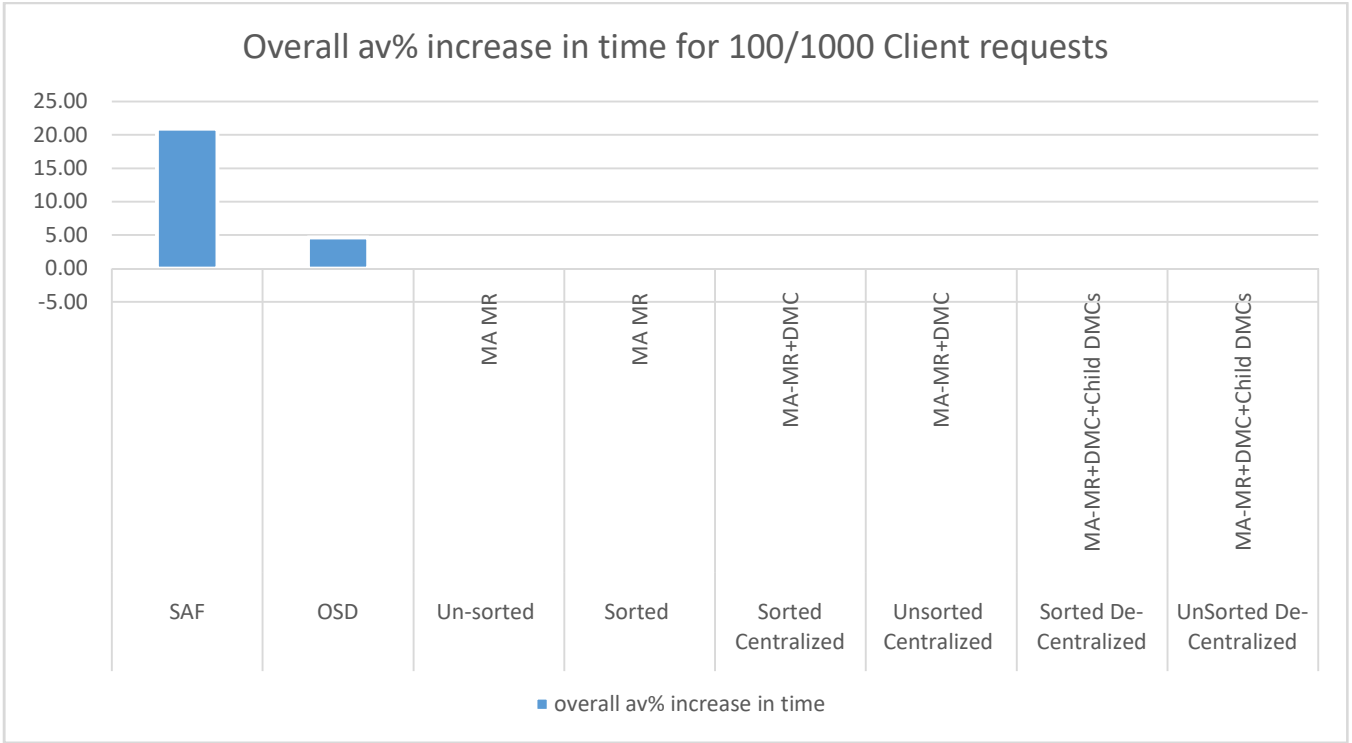


Figure 5: BAR CHARTS SHOWING AVERAGE INCREASE IN TIME FOR 100 AND 1000 CLIENT REQUESTS

Figure 5 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.

4.3.2 Summary Latencies Analysis

Table 11: COMPARISON OF ALL THE CASES FOR THE LATENCIES.

TYPE	SAF	OSD	Un-sorte	Sorted	Sorted C	Unsorted	Sorted D	UnSorte	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 Latencie	-1.64	1.37	-44.78	-45.09	-44.67	-36.87	-45.27	-45.21	-39.53

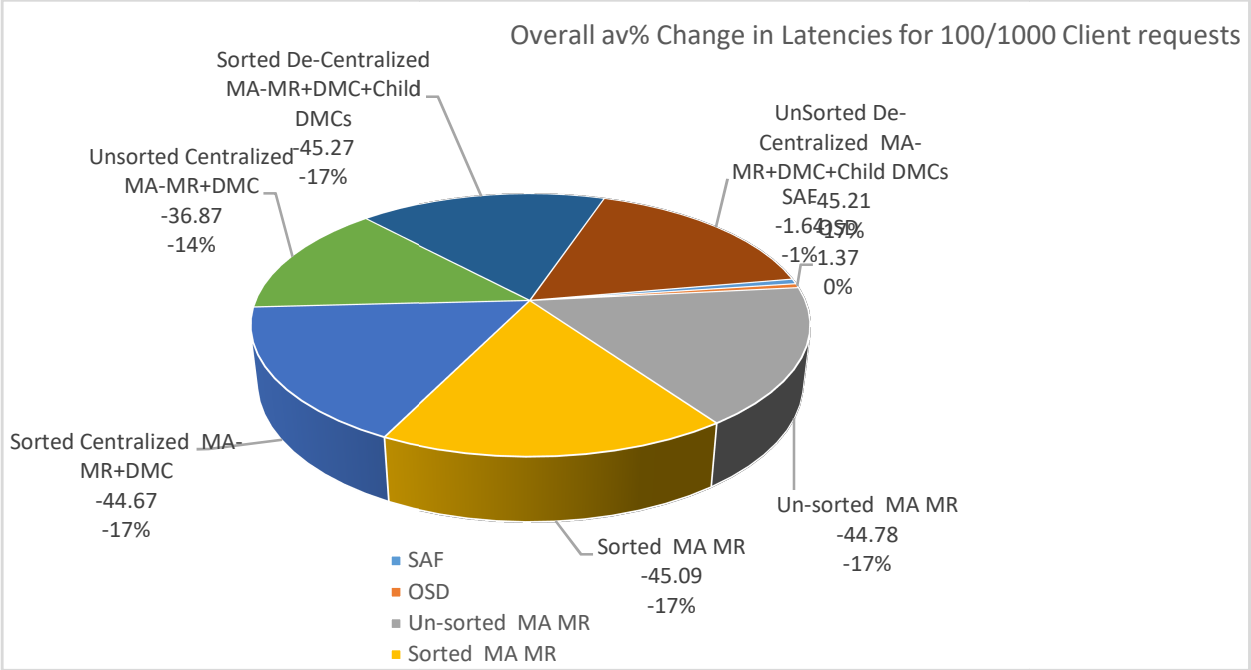


Figure 6 PIE CHARTS SHOWING AVERAGE CHANGE IN TIME FOR 100 AND 1000 CLIENT REQUESTS.

In Figure 6 shows the Negative % shows overall decrease in Latencies to execute while the Positive % indicate increase in Latencies. In the above chart MA-MR and Chid DMcs,Sorted MA-MR,Centralized Sorted MA-MR,Unsorted MA-MR and uncentalized MA-MR ALL contain -17% , implying reduction in latencies by 17% for each of them when the client requests have increased to 1000 from 100. For OSD at 0% , no change in latencies and SAF at -1% decrease in Latencies by 1% margin.

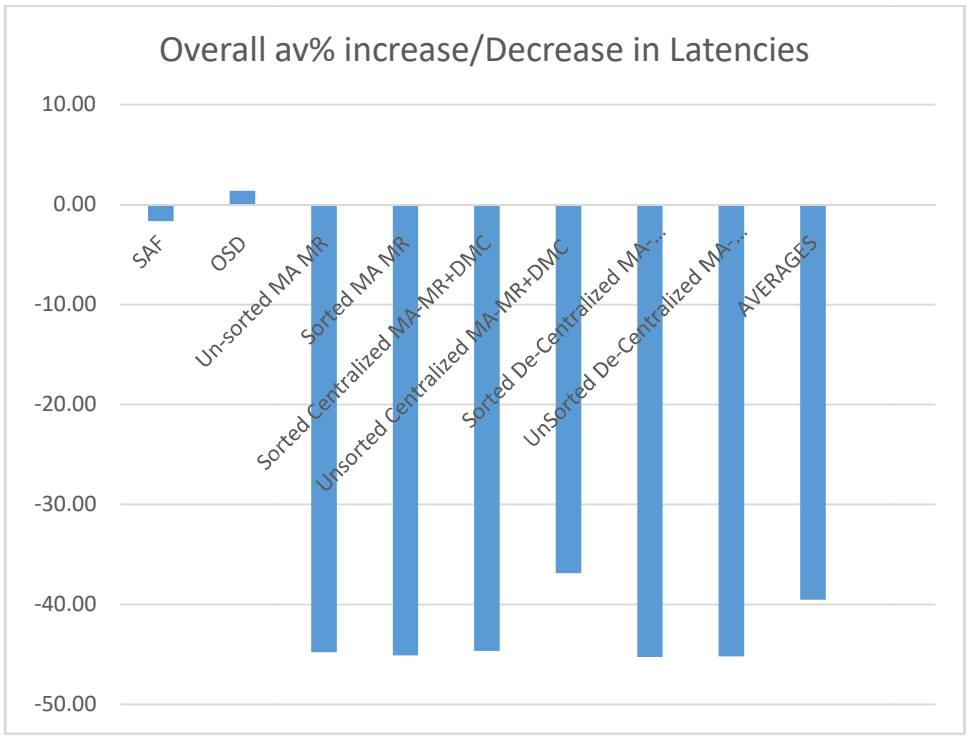


Figure 7 BAR CHARTS SHOWING AVERAGE INCREASE/DECREASE IN OVERALL LATENCIES FOR 100/1000 CLIENT REQUESTS.

Figure 7 above show 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 6 above.

4.2 Summary Throughput Analysis

Table 12: COMPARISON OF ALL THE CASES FOR THE THROUHPUT.

COMPARISON OF THE CASES FOR THE AVERAGE THROUGHPUTS									
TYPE	SAF	OSD	Un-sorted	Sorted	Sorted Cent	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 Latenc	0.0000000	16.6666667	4.1962256	5.1459990	4.9658244	-10.6426190	6.8440309	6.3781852	2.8145519

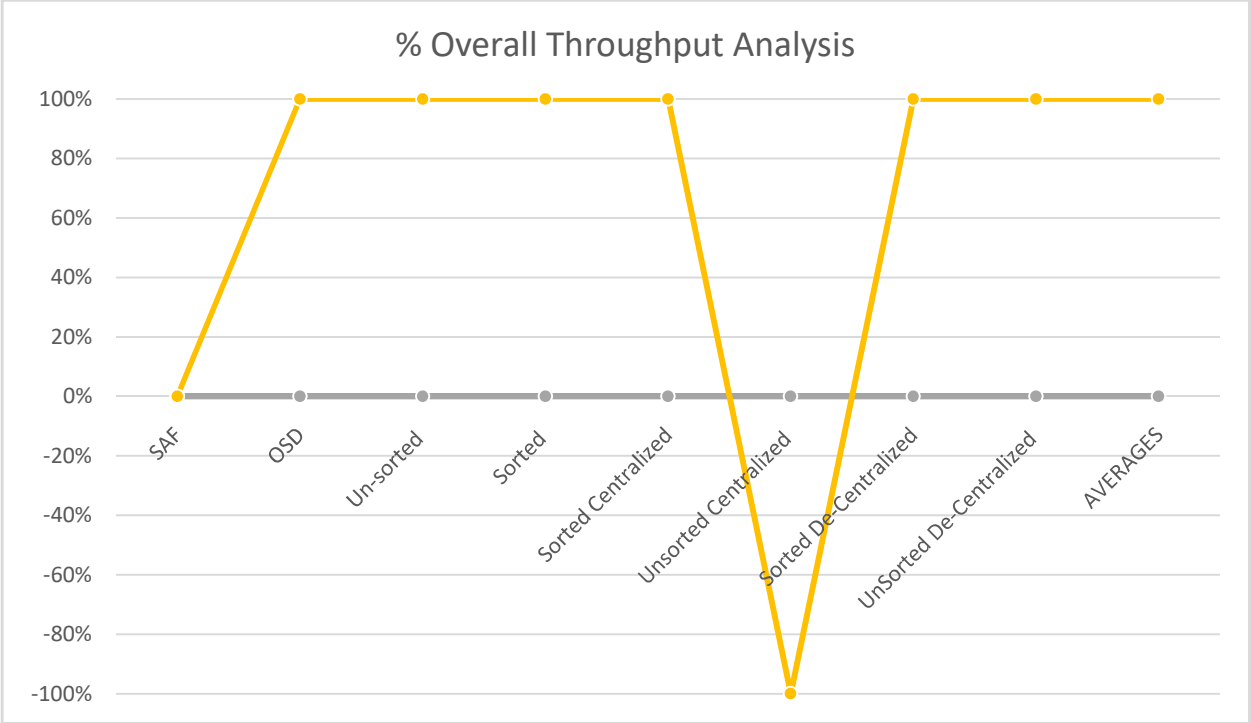


Figure 8 A LINE GRAPH SHOWING AVERAGE OVERALL THROUPUT 100-1000 CLIENT REQUESTS.

Figure 8 above 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.

V. CONCLUSION

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.

Mobile agents furtherplay a key role in contributing to the performance improvement of the distributed system as witnessed in our experimental analysis. Mobile agents can autonomously move from one place to another with metadata and security being guaranteed.

This paper has also 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.

Finally, it is therefore evident that mobile agents can play a major role in improvement of the overall performance of a clustered distributed network, especially if the clusters are sorted into cluster sets using Map reduce.

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]. 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.
[9]. Ceph (2016) *Welcome to Ceph*. Available at: http://docs.ceph.com/docs/master/# (Accessed: 30 April 2019).
[10]. Chaturvedi, V. (no date) *Deep Dive into Docker*. Available at: https://www.edureka.co/blog/what-is-docker-container (Accessed: 26 March 2019).
[11]. CORP (2016) *Content addressed storage systems*, *EMC*. Available at: http:www.emc.com/products/systems/centera.jsp?openfolder=platform (Accessed: 26 June 2016).
[12]. Docker (2019) *Docker Docs*. Available at: https://docs.docker.com/v17.09/compose/install/ (Accessed: 27 March 2019).
[13]. EMC2 (2008) *Where information lives:current benefit and future potential technology concepts and business considerations*.
[14]. Escriv, M., C, J. P. and Bada, G. A. (2014) 'A Jabber-based Multi-Agent System Platform ∗', (January 2006). doi:

10.1145/1160633.1160866.
[15]. 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-B9E4-F52C5E1BAE79.
[16]. 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.
[17]. Feng, D. *et al.* (2004) 'Enlarge Bandwidth of Multimedia Server with Network Attached Storage System 3 The Redirection of Data Transfer', pp. 489–492.
[18]. Finin, T. (1992) *An Overview of KQML : A Knowledge Query and Manipulation Language*.
[19]. FIPA (2000) 'Foundation for Intelligent Physical Agents', *Inform*.
[20]. FIPA (2002) 'FIPA Abstract Architecture Specification (SC00001L)', p. 75.
[21]. FullStack (no date) *Full Stack Python,Redis.* Available at: https://www.fullstackpython.com/redis.html (Accessed: 28 March 2019).
[22]. 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.
[23]. Griffit (2018) *how-build-hello-redis-with-python*, *Opensource.com*.
[24]. Hendricks, J. *et al.* (2006) 'Improving small file performance in object-based storage', (May).
[25]. Hitachi (2016) *Storage virtualisation:How to capitalize on its economic benefits*.
[26]. Iii, W. B. L. and Ross, R. B. (2000) '4th Annual Linux Showcase & Conference , Atlanta PVFS : A Parallel File System for Linux Clusters £'.
[27]. James (2006) 'Improving small file performance in object based storage.', *CMU-PDL-06-104.*
[28]. James, J. (no date) '"Cassandra"', *Notes and Queries*, s2-X(241), p. 111. doi: 10.1093/nq/s2-X.241.111-a.
[29]. Karakoyunlu, C. *et al.* (2013) 'Toward a Unified Object Storage Foundation for Scalable Storage Systems'.
[30]. Kasireddy, P. (2016) *A Beginner-Friendly Introduction to Containers, VMs and Docker*. Available at: https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b?gi=a26c3acc92c1 (Accessed: 26 March 2018).
[31]. 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.
[32]. Lehner, W. (2013) *Web-Scale Data Management for the Cloud*.
[33]. Li, G. *et al.* (2006) 'Researches on Performance Optimization of Distributed Integrated System Based on Mobile Agent *', pp. 4038–4041.
[34]. Liancheng, X. U. (2014) 'Research on Distributed Data Stream Mining in Internet of Things', (Lemcs).
[35]. 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/.
[36]. Maitrey, S. (2015) 'Handling Big Data Efficiently by using Map Reduce Technique'. doi: 10.1109/CICT.2015.140.
[37]. Mark et al. (2000) *Storage Virtualisation, What is it all about?*
[38]. 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: 10.1109/49.611154.
[39]. Mesnier, M. *et al.* (2003) '01222722', (August), pp. 84–90.
[40]. Microsystems, S. (2007) 'LUSTRE ™ FILE SYSTEM', (December).
[41]. 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.
[42]. Mishra, A. (2012) 'Application of Mobile Agent in Distributed Network Management'. doi: 10.1109/CSNT.2012.198.
[43]. 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.
[44]. 'MSST-Cabrera' (1991).
[45]. 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.
[46]. Oracle, S. (2011) 'Lustre Software Release 2. x Operation Manual'.
[47]. Osero, B. O. (2010) *Storage virtualisation and management*. University of Nairobi.
[48]. 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.
[49]. Palanca, J. (2018) 'SPADE Documentation'.
[50]. Panasas, I. (2016) 'Panasas', *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Panasas.
[51]. Patel, A. B., Birla, M. and Nair, U. (2012) 'Addressing Big Data Problem Using Hadoop and Map Reduce', pp. 6–8.
[52]. Pedro Jos´e Marr ´on, Stamatis Karnouskos, D. M. A. O. and the C. consortium (2011) *No Title*.
[53]. Permabit (2015) 'Permabit', *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Permabit.
[54]. 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.
[55]. Randy, Fellows, A. R. and Kerns, R. (2012) 'SAN Virtualization Evaluation Guide', p. 2.
[56]. 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.
[57]. 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.
[58]. Rugg, G. and Petre, M. (2004) 'The Unwritten Rules of PhD research', *Open University Press*, p. 241.
[59]. Sargent, R. G. (2011) 'Advanced Tutorials: Verification and Validation of Simulation Models', *Proceedings of the 2011 Winter Simulation Conference*, pp. 183–198.
[60]. Satoh, I. (2011) 'Mobile Agent Middleware for Dependable Distributed Systems'.
[61]. Satoh, I. (2014) 'MapReduce-based Data Processing on IoT', (iThings). doi: 10.1109/iThings.2014.32.
[62]. Silva, L. M. (1999) 'Optimizing the Migration of Mobile Agents'.
[63]. Singavarapu, S. and Hariri, S. (2001) 'S ELF-MANAGING STORAGE SYSTEM – D ESIGN AND EVALUATION 2 . Self Managing Storage System (SMSS) Architecture – Overview'.
[64]. Sowmya, N., Aparna, M. and Tijare, P. (2015) 'An Adaptive Load Balancing Strategy in Cloud Computing based on Map Reduce', (September), pp. 4–5.
[65]. Tate, J. *et al.* (2017) 'Introduction to Storage Area'.
[66]. Tekniska, K., Ögskolan, H. and Simsarian, K. T. (2000) 'VETENSKAP OCH KONST Dissertation, March 2000 Computational Vision and Active Perception Laboratory (CVAP)', (March).
[67]. Tutorialpoint (no date) *REDIS - QUICK GUIDE REDIS - ENVIRONMENT REDIS - DATA TYPES*.
[68]. 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'.

[69]. Weber, R. O. (2004) 'Information technology—SCSI object-based storage device commands (OSD)', *Technical Council Proposal Document*, 10, pp. 2003–2031.
[70]. Weil, S. A., Brandt, S. A. and Miller, E. L. (2006) 'CRUSH : Controlled , Scalable , Decentralized Placement of Replicated Data', (November).
[71]. Welch, B. *et al.* (2008) 'White Paper Scalable Performance of the Panasas Parallel File System', *Fast 2008*, (May), pp. 1–22.
[72]. Wu, S. A. I. (2014) 'Distributed Data Management Using MapReduce', 46(3).
[73]. 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.
[74]. 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.
[75]. Yu, P. *et al.* (2006) 'Mobile Agent Enabled Application Mobility for', pp. 648–657.