NEXT GENERATION INTERNET

Open Call 4c

ARES: A Next generation Erasure Coded, Shared Distributed Storage System

Deliverable 3: Experiment Results and Final Report

Authors	Nicolas Nicolaou, nicolas@algolysis.com, Algolysis Ltd Theophanis Hadjistasi, theo@algolysis.com, Algolysis Ltd Andria Trigeorgi, an.trigeorgi@gmail.com, Algolysis Ltd
Due Date	01/08/2022
Submission Date	01/08/2022
Keywords	distributed storage, atomic memory, shared objects, fault tolerance, consistency, erasure coding

Deliverable 3: Part I

Analysis, results, and wider impact

1 Abstract

Distributed Shared Storage Services, is the building block to yield complex, decentralized, cloud applications in emerging technologies (e.g., IoT, VR/AR), as they may offer a transparent cloud storage space where distributed applications can store, retrieve, and coordinate over shared data. In this project, the EU-US team performed experimental evaluations on the performance of a novel protocol ARES [1], which implements an Atomic Distributed Shared Storage space over asynchronous, failure prone, message passing network nodes, and it ensures data availability and survivability. Armed with extensive prior experience on the field of Distributed Computing, the US team led while the EU team led the development, deployment, and execution of the experiments. The scenarios evaluated ARES's performance under scalability, resource utilisation, and fault-tolerance, and attempted to identify bottlenecks and shortcomings that may prevent ARES from being readily applied in a real-time, data-driven, practical systems. Cross-Atlantic experiments were conducted by reserving network nodes both in the EU as well as in the US through Fed4FIRE+ testbeds, i.e., VirtualWall (EU), CloudLab (US), InstaGENI (US), and Grid5000 (EU), and RPi nodes at the premises of the EU partner. Overall, our experiments demonstrated that ARES performs well, in terms of operation latency, under various environmental conditions, and performs closely to existing commercial solutions, while preserving strong consistency, fault-tolerance, and longevity.

2 Project Vision

Distributed Storage Systems (DSS) store large amounts of data in an affordable manner. Cloud vendors deploy hundreds to thousands of commodity machines, networked together to act as a single giant storage system. Yet, component failures, and network delays are the norm, thus ensuring consistent data-access and availability at the same time is a challenging task. Vendors often solve availability by replicating data across multiple servers. However, keeping these copies consistent, especially when they can be accessed concurrently by different operations, is very difficult and costly. The problem of keeping copies consistent becomes even more challenging when failed servers need to be replaced or new servers are added, without interrupting the service. Any type of service interruption in a heavily used DSS usually translates to immense revenue loss.

Commercial DSS avoid providing strong consistency guarantees as they are considered costly and difficult to implement in an asynchronous, fail prone, message passing environment. Indeed, initial implementations of Atomic DSS had high demands in communication, storage, and sometimes computation. Recent works however, invest in algorithms that may reduce the overheads on the aforementioned parameters. In particular, there exist algorithms that reduce the communication cost by trading (cheaper) computation resources. Others propose the use of erasure coding techniques to reduce both communication overheads and storage demands in the replica hosts, trading however the number of faults tolerated.

ARES goes one step further and attempts to harvest the efficiency of the proposed algorithms by incorporating an adaptive approach that allows algorithm switching based on the application needs. ARES offers *strong consistency* guarantees (atomicity), providing the illusion that data are accessed sequentially when, in reality, multiple processes may read and write the same data object concurrently. In addition, ARES provides the capability of dynamically changing the membership of the replica host, enabling the service to stay alive even in the presence of server failures or server replacement. Atomicity is the most intuitive semantic, which if implemented correctly and efficiently, may relief developers from a major headache of writing complex code or communication protocols and will provide a transparent shared memory service alleviating low level synchronization tasks for distributed applications. By utilizing communication lightweight algorithms via its adaptive nature, together with erasure coding techniques, ARES promises a communication and storage efficient service, which may attract the attention of potential stakeholders.

The ultimate vision of this project was to provide clear indications for the possibility of deploying a global-scale shared memory (storage) space, introducing the new concept of Memory-as-a-Service (MaaS). Through our experiments we deployed nodes in both sides of the Atlantic and we examined the performance of the ARES protocol in both intra- and inter-continental experiments. We have put the algorithm under various tests, and we have compared its performance with existing commercial solutions providing a clear reference on the potential use of ARES in real setups. Overall, our experiments demonstrated that ARES is a promising technology that may compete head-to-head with existing solutions. Moreover, they helped us identify shortcomings and bottlenecks of the ARES protocol which will help us to rethink and redesign aspects of the protocol that will improve its overall performance.

Co-funded by the Horizon 2020 Framework Programme of the European Union

3 Details on participants

Algolysis Ltd – Cyprus (EU) – SME

Dr. Nicolas Nicolaou [Algolysis LTD] is the co-founder and a senior scientist and algorithms engineer at Algolysis. Dr. Nicolaou participated in the project utilizing his long expertise on the formalization and analysis of Distributed Algorithms for Atomic R/W Storage Systems. He is one of the main co-authors of ARES, the algorithm which we developed and examined in this project, and he has been the main architect of multiple efficient algorithms in the field.

Role: Project Coordinator (PC) and *Design of Experiments, Implementation* of the PoC and the *Analysis* of the experimental outcomes.

Dr. Efstathios Stavrakis [Algolysis LTD] is a co-founder and a senior scientist and algorithms engineer at Algolysis Ltd and has been conducting research in academia and the industry for 15 years. His research is in the areas of computer graphics, animation, virtual reality and games.

Role: Design, Analysis, and Data Visualization of the experiments.

Dr. Theophanis Hadjistasi [Algolysis LTD] is conducting research in Fault-tolerant Parallel and Distributed Computing, with emphasis on Distributed Atomic Storage Implementations. His research interests span “Theory” and “Practice” with a focus on Algorithms and Complexity.

Role: Development and Deployment of the Prototype, and Analysis of the experiments.

Ms Andria Trigeorgi [Algolysis LTD], PhD candidate in Computer Science at the University of Cyprus and researcher at Algolysis since the beginning of the project.

Role: Development and Deployment of the Prototype, and Analysis of the experiments.

Penn State University – Pennsylvania (USA) – Academic

Dr. Viveck Cadambe [PSU], is an expert on distributed algorithms and erasure coding, and has previously participated in development of linearizable shared memory emulation algorithms via erasure coding, as well as provably consistent non-blocking reconfigurable algorithms.

Role: Design & Analysis of the experiments and provide equipment to deploy the experiments.

Dr. Bhuvan Urgaonkar [PSU], brings expertise in systems software, distributed computing, and performance modelling. He has made contributions to resource management in clouds, power/cost management of DCs and storage systems.

Role: Design & Analysis of the experiments and provide equipment to deploy the experiments.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4 Experiment Description and Implementation Details

The team from Algolysis and the PSU collaborated to develop and test a prototype of a next generation dynamic, atomic, distributed shared memory (ADSM) service, called ARES. During the past decade, several ADSM algorithms have been proposed in the literature. Some consider environments where the set of replica hosts remains the same (static), and others allowing that set to change over time (dynamic). All the proposed algorithms are theoretically proven to satisfy strong consistency guarantees, however only a few have presented experimental findings, and none has been deployed for real use. Compared with previous approaches ARES offers three unique innovations:

- It is *highly fault tolerant*: the set of data hosts may change dynamically to mask transient or permanent failures without any service interruptions
- It is *modular*: generic modular primitives allow the use of different ADSS algorithms and data replication policies (including erasure coding) based on the application's demands
- It ensures *strong consistency*: despite operation concurrency and adaptive behaviour, the shared storage appears as if it is accessed sequentially, imposing a total order on the read and write operations.

4.1 Experiments Description

Experiments allow to test the performance of ARES across transatlantic deployments, and under various traffic, environmental, and node failure conditions. More precisely, we performed the following classes of experiments:

SCALABILITY TESTS: Scalability experiments aim to test the ability of the service to maintain performance while the set of service participants becomes larger. In our scalability scenarios we modify the number of **writers** (end users), **readers** (end users), **reconfigurers** (end users) and **replica servers**.

STRESS TESTS: Stress tests include experiments that test the performance of ARES under various concurrency patterns, by allowing multiple participants to access shared resources concurrently. Additionally, these experiments test the performance of the service under various resource loads, by modifying the size of the data shared among the participants.

FAULT TOLERANCE TESTS: ARES service is fault-tolerant, i.e., it may stay alive despite the existence of failures among the nodes that implement it. In fault-tolerance experiments, we introduce crash-failures in the system by terminating the operation of a set of server replicas on specific points in the execution. This helps us verify the fault-tolerance guarantees and the responsiveness of the service (i.e., measure the time it takes for the service to reconfigure for preserving liveness in the case of failures).

PERFORMANCE COMPARISON: We examined how the underlying distributed shared storage algorithm affects the overall performance of the service. We developed and deployed (i) ABD [2,3] a baseline static algorithm, (ii) Cassandra [7], an open source, distributed, NoSQL database, and (iii) Redis[4], an open-source, distributed key-value store. Comparison with those implementations provide valuable observations on how ARES competes with existing solutions. Additional details on Cassandra and Redis algorithms, are given bellow.

CASSANDRA [7] is a NoSQL distributed database offering continuous availability, high performance, horizontal scalability, and a flexible approach with tuneable parameters. It was initially

Co-funded by the Horizon 2020 Framework Programme of the European Union

developed by Facebook for Facebook's inbox search feature. Today, it is an open-source application of Apache Hadoop. Cassandra uses peer-to-peer communication where each node is connected to all other nodes, forming a logical ring topology. The protocol used to achieve this communication is gossip, in which nodes periodically exchange state information about themselves. All the nodes in a cluster can serve read and write requests. Thus, when a request is sent to any node, this node acts as the coordinator. The coordinator distributes execution around the cluster, gathers the responses from the replicas, and responds back to the client. By default, Cassandra guarantees *eventual consistency*, which implies that all updates reach all replicas eventually. However, Cassandra offers tuneable consistency for read and write operations, guaranteeing weaker or stronger consistency, as required by the client application. The required consistency can be achieved by tuning the consistency level (CL) and replication factor (RF) parameters. RF specifies how many copies of a store object (i.e., a row in Cassandra's DB) is kept among the participants. Given RF, the CL controls how many responses the coordinator waits for before the operation is considered complete. Cassandra is dynamic, allowing the removal and addition of a single node at a time, in contrast to ARES that allows a complete modification of the configuration (reconfiguration) in a single operation.

REDIS [4] is an open source, in-memory key-value store. The read/write response time for Redis is extremely fast since all the data is in memory. Redis is based on Master-Slave Architecture, i.e., it enables replication of master Redis instances in replica Redis instances. The use of Redis is rather easy; Redis will internally store the key and value when users execute commands like set key value. Redis returns the value with a simple get key command from the user. The data size cannot exceed the main memory limit because all the data are in main memory; Redis starts to reply with an error to write commands when the max memory limit is reached. Redis has two persistence mechanisms: RDB and AOF. RDB persistence provides point-in-time snapshots of the database at specified intervals. AOF persistence logs every write operation. When the database server starts, Redis reads the AOF log to reconstruct the database. RDB is perfect for backup, but if the RDB stops working all data changes since the last snapshot are lost. In comparison, AOF has better durability, although adopting AOF persistence may result in performance loss. Redis has a command called "WAIT" in order to implement synchronous replication. This command blocks a writer until all the previous write commands are successfully transferred and acknowledged by at least the specified number of replicas. Even with "WAIT" in place Redis can only guarantee *eventual consistency* as reads do not wait other than the master node. By [4], usage of "WAIT" ensures *atomicity* in many cases.

4.2 ARES Implementation

The architecture of our Distributed Storage System (DSS) can be seen in Figure 1. This includes the modules composing the infrastructure as well as the communication layer between them. There are two main modules: (i) a Manager, and (ii) a Distributed Shared Memory Module (DSMM). The Manager module handles the client commands (read and write operations through the Command Line Interface - CLI) and establishes the gateway to access the memory. The memory objects are maintained by servers through the DSMM. Notice that, the Manager uses the DSMM as an external service to write and read objects to the shared memory. To this respect, our architecture is flexible to utilize any underlying DSM algorithm in the DSMM.

Co-funded by the Horizon 2020 Framework Programme of the European Union

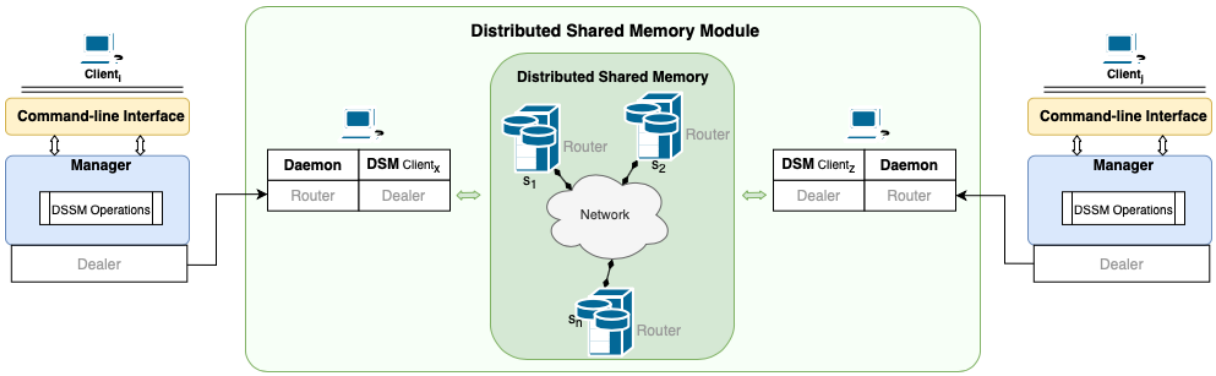


Figure 1: The basic architecture of our Distributed Storage System.

An illustration of the implementation components along with the component interconnectivity for a DSM instance appears in Figure 2. We have implemented two DSM algorithms. First, we integrated algorithm ABD to our DSM Module. Next, we implemented algorithm ARES with two different DAPs (ABD and EC) and then we integrated that implementation to the DSM Module. Notice that the implementation of ARES requires a consensus algorithm to be implemented as well. So, we implemented the RAFT [8] consensus algorithm. For the algorithms and all the modules implementation Python was chosen as the programming language. For the needs of the underlying communication protocol, we used the ZeroMQ [5] messaging library written in Python. Our implementation utilizes an open-source implementation of RAFT, PySybcObj also written in Python [9].

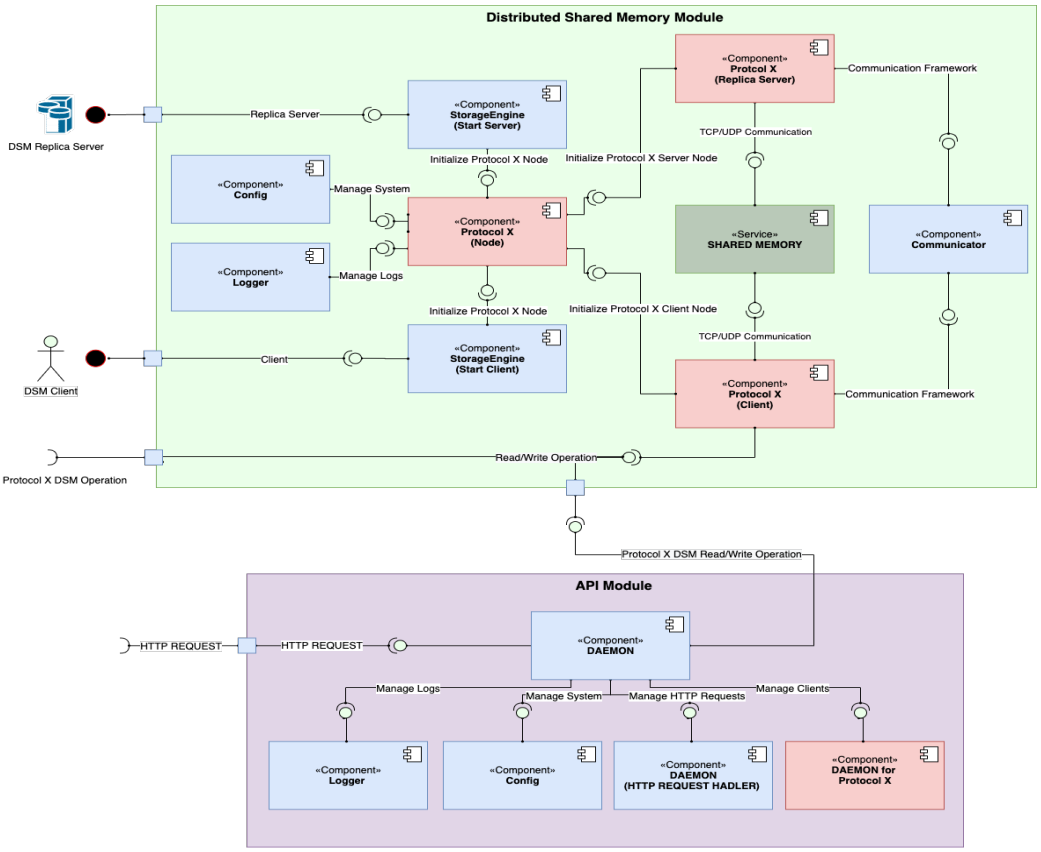


Figure 2: The implementation components of a DSM Protocol.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.3 Deployment and Testbed Setup

NODE PROVISIONING

To set up our network topology we used the jFed [16] GUI tool which was developed within the Fed4FIRE+ [11] project. We draw a topology with several physical and virtual nodes (e.g., Figure 3 shows an instantiation of server and client nodes, and one controller). We configured each node, selecting its testbed, and disk image. Thus, based on the resources defined in the RSpec (a description of the requested resources), the JFed deploys the nodes and executes the selected testbed.

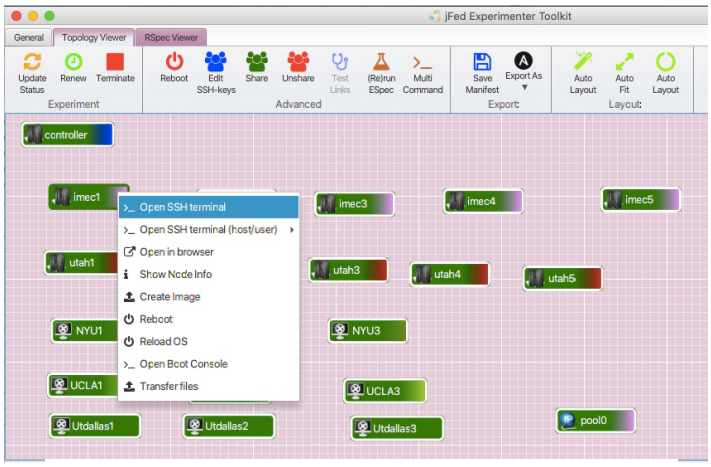


Figure 3: JFed GUI showing an instantiation of server and client nodes, and one controller node.

Platforms & Geographic Locations: We used nodes located both in the EU and the US. In particular from Europe, we use the **GRID'5000** [12] testbed that spans over 8 sites located in France and Luxembourg; the **Virtual Wall** [13] composed of >550 servers that can be used as bare metal hardware and is located in Belgium; **InstaGENI** [14] of NYU, UCLA, UT Dallas; **Cloudlab Utah** [15], which has a large number of servers, each with relatively modest specifications; and RPi's (small nodes) by Algolysis LTD, geo-distributed in Cyprus for demonstrating the applicability of the protocols in cheap off-the-shelf hardware. All the above testbeds are wide area deployments that can be readily accessed and used through the FED4FIRE+ (JFed) platform.

REQUIRMENTS INSTALLATION

Once the nodes are defined and initialized the next step was to install the modules required by our experiments in each node. This is a two-step procedure: (i) we installed the git repository with the project code, and (ii) we installed all the required python libraries. To automate the procedure, we created a script to install the requirements on the nodes. This script accepts a number of different arguments based on the deployment needs. Additional usage details are given bellow:

```
$ python3 install-requirements.py -testbed <fed4fire|aws|emulab > -proxy <false|true>
```

- The **testbed** name. Available options are Emulab, Fed4fire, and AWS EC2 testbeds. In our experiments we used the Fed4fire testbed.
- The **proxy** parameter is used to state if we need a proxy node to provide a gateway between nodes that do not have routable ips and the internet. For example, for the installation of requirements on Fed4Fire, we set this parameter to **true**, as not all the nodes have public ips. The default value is false.

Co-funded by the Horizon 2020 Framework Programme of the European Union

TECHNICAL DIFFICULTIES

Deploying the code and running the experiments was not an obstacle-free procedure. We managed to overcome any issues we faced during the deployment and execution of the experiments and luckily, they did not affect the final outcomes. The list of the main difficulties we faced along with their solution is the following:

- **Difficulty:** The nodes of VirtualWall 1 or 2 testbeds (a testbed of Fed4Fire) could not be accessed directly from external machines through the internet.

  Solution: To overcome this obstacle we enabled NATting for both VirtualWall 1 or VirtualWall 2 using a script we created. For example:

  ```
  $ ansible-playbook -i start-nodes/config.ini enable_NATted_IPv4_imecVitualWall.yml
  ```

- **Difficulty:** The imec Virtual Wall 1/2 nodes that are used as servers need to be reachable over IPv4.

  Solution: We used an Address Pool on jFed and a script in order to configure the public IPv4-addresses of the Address Pool to the raw-pcs.

  ```
  $ python3 configureIPV4ips.py -manifest ../manifest.mrspec -ssh_config ../ssh-config
  ```

- **Difficulty:** The nodes of the Grid5000 testbed could not be accessed directly from external machines through the internet.

  Solution: To overcome this problem, we had two possible solutions: (i) we could connect to Grid5000 nodes through a VPN (using Tunelblick), or (ii) use of a reconfigurable firewall. Solution (i) is not ideal as it will route the traffic through a single VPN machine, and this may generate a performance bottleneck. The second solution provides only routable ipv6 addresses which were not supported by all the other testbeds. Thus, we decided to use Grid5000 nodes only as clients that only require outgoing communication.

- **Difficulty:** Nodes in PSU could not be acquired as the procedure to get access to the internal VPN of the university was tedious.

  Solution: We did not use machines from the internal network of the PSU but rather we utilized the Cloudlab and InstaGENI testbeds to establish the US network. This was a change in the initial plans which did not affect the proper execution of the experiments. In contrast, it was rather beneficial as it allowed us to utilize devices in a wider geographical area that span throughout the US region.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.4 Running the Experiments

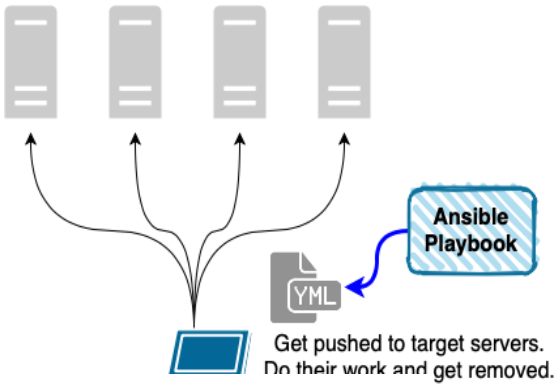


Figure 4: Run tasks using Ansible.

We used two main tools to execute our initial experiments: (i) jFed [16], and (ii) Ansible [6]. Ansible is a tool to automate different IT tasks, such as cloud provisioning, configuration management, application deployment, intra-service orchestration. There are two main steps to run an experiment: (i) booting up the Client Nodes (either writer or reader) and the Server Nodes in the testbed, and (ii) executing each scenario using *Ansible Playbook* scripts written in YAML language. As shown in Figure 4, the scripts get pushed to target servers from a control machine, do their work there and then get removed. In our experiments, one instance node was dedicated as a controller to orchestrate the experiments.

For each round of our experiments, we had to execute 6 Ansible Playbooks in series:

- **Playbook 1:** we terminate all nodes removing their previously created log files. In addition, we create a new config file including the current ip addresses of servers and start the nodes again.
- **Playbook 2:** a writer invokes a write operation of a specific data object (e.g., file) and the other writers invoke a read operation in order to retrieve the initial copy of this object.
- **Playbook 3:** all the readers and writers start invoking operations on the object.
- **Playbook 4:** acts as a sync mechanism by monitoring all the processes created from Playbook 3 and it waits until all of them terminate before signalling Playbook 5 to start.
- **Playbook 5:** we execute a read operation in order to read the final file that the servers have. To save space, we only save the stats of this file, and not the whole file.
- **Playbook 6:** we fetch on the control machine all log files that are created from the current round.

Ansible allows you to choose how you want to control a play's execution (this parameter is called Strategy). By default, it plays a run with a linear strategy, in which all hosts will run each task before any host starts the next task, using the number of forks (default=5) to parallelize. For the Playbook 3 we use the free strategy which allows each host to run until the end of the play as fast as it can. In our experiments we specify the number of forks to be the total number of clients and servers.

Cassandra Experimentation: For each experimental round we had to execute the Ansible Playbooks as mentioned above. We install Apache Cassandra 4. For the needs of Cassandra, the first two Playbooks were customized as follows:

- **Playbook 1:** It shuts down the cluster and removes any previously created log files. In addition, it creates a new configuration file including the current ip addresses of servers. The next task is to (i) edit the replicas nodes by setting the following parameters: seeds, listen address, rpc address, (ii) restart Cassandra instance, and then (iii) pause for 30 seconds for discovery to work.
- **Playbook 2:** We use blobs (binary large objects) in Cassandra table. A writer invokes a write operation of a specific blob (create a KEYSPACE, a TABLE, and send WRITE). The

Co-funded by the Horizon 2020
Framework Programme of the European Union

other writers invoke a read request to get this blob. All these requests are made using the Cassandra-driver Python library. We set the CL to quorum, which means that a majority of (n/2 +1) nodes of the replicas must respond. Also, we set the replication factor (RF) to the majority, where the RF is the number of copies of each row.

Redis Experimentation: For each experimental round we had to execute the Ansible Playbooks as mentioned above. In our set-up we install Redis 5. For the needs of Redis, the first two Playbooks were customized as follows:

- **Playbook 1:** It shuts down the cluster and removes any previously created log files. In addition, it creates a new configuration file including the current ip addresses of servers. The next task is to (i) set a server as the MASTER, (ii) starting Redis Server Service in each replica server.
- **Playbook 2:** We use binary strings in Redis implementation. A writer invokes a write operation of a specific string (using the set function). The other writers invoke a read request (using the get function) to get this string. All these requests are made using the Redis-driver Python library. We run two variants of Redis algorithm, with and without the WAIT command during a write operation. The WAIT command in Redis blocks the current writer until all the previous write commands are successfully transferred and acknowledged by at least a pre-specified number of replicas. We specified this number with a majority, to match the ABD algorithm.

Experimentation Process Automation: Having to monitor and run the ansible scripts one by one is a time-consuming process. To automate the process, for each experiment we created its own script written in python that will take care the process. As an example, we created an experiment that increases the file size while keeping the readers, writers and servers fixed to 5. Assume the script name is set to **execute_file_size_exp.py** then its call will be:

```
$ python3 execute_file_size_exp.py -fast_op <False|True> -exp_name <experiment name> -protocol <ARES|ABD|Cassandra> -dap_protocol <ABD|EC> -parity <parity number for EC> -testbed <emu-lab|aws|fed4fire> -load_policy <growingsize|fixedsize>
```

With the following arguments:

- Set the **fast_op** argument to True if you want to enable the fast read mechanism.
- Set the **exp_name** argument to the name of emulab experiment name, in case you use the https://www.emulab.net/.
- Set the **protocol** argument to the algorithm name that we want to run [ARES, ABD, Cassandra].
- Choose the **dap_protocol** argument [EC, ABD] only if the protocol is ARES.
- Set the **parity** number only if the protocol is any variant of ARES with EC chosen as DAP.
- Set **testbed** type. We currently support Emulab, AWS EC2 and Fed4Fire. In the experiments of this work, we set this value to Fed4Fire.
- Argument **load_policy** has two possible values: growingsize and fixedsize. In the first, writers write at a random file that its size keeps growing and, in the latter, they write at a random file of fixed size. In these experiments, we set this value to fixedsize.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4.5 Logging and Data Collection

GENERETING LOGS

We use python-json-logger library to configure our implementations to log DEBUG and higher-level messages from clients and servers to .log files on disk. The logs follow a format that includes the following standard attributes like the *asctime*, *levelname*, *message*, and extra customised attributes according to the message type.

COLLECTING LOGS

As previously explained, in each experimental round we use Playbook 6 to collect the log files from each node to the control machine. Then, at the end of all the experiments, we download the log files in our workspace.

PLOTTING DATA

To plot our results, we use Grafana, an open-source visualization platform that lets you illustrate data in a variety of graphs and charts (time series, bar chart, histogram etc). To be able to utilize Grafana we parsed the collected logs and we import the data in InfluxDB, a time series database. The structured storage of data in InfluxDB, allowed the composition of complex queries for the generation of a large set of statistical data. A dashboard was created for each scenario that allowed the researchers to visualize quickly and easily various plots by dynamically changing the value of a number of variables. The dashboards were then embedded in the project's website so any visitor may explore freely the data we collected in our experiments.

5 Results

Our analytical results aim to expose how a strongly consistent, reconfigurable service like ARES, compares in performance with the two commercial storages of our choice, namely Cassandra and Redis. Moreover, it helps us identify bottlenecks and shortcomings of ARES for future optimizations, and, in some scenarios, we demonstrate the ability of ARES to utilize erasure-coding and to cope with failures and dynamic reconfiguration.

To summarize, below we show the expected Key Performance Indicators (KPIs) that were proposed in deliverable D1, and what was achieved through this experimental evaluation. One additional KPI was achieved. Additional details are given in the tables bellow.

KPI	Measure	Target	Achieved
1	**Scalability Test 1:** Number of processes to access a single shared object concurrently	Allow more than 250 (read/write) concurrent processes in the service	Scalability experiments as the number of **readers** increases from **5 to 250**. Scalability experiments as the number of **writers** increases from **5 to 20**. Thus, the service indeed allows **more that 250 processes concurrently**. (Scalability scenarios)
2	**Scalability Test 2:** Examine the performance of r/w operations when modifying	Measure performance in seconds and expect linear increment as the # of replica hosts grows.	The experiments of KPI-1 were repeated for **3 and 11 servers**. The linear incrementation is not true for ARES with EC DAP due to the parity selection.

Co-funded by the Horizon 2020 Framework Programme of the European Union

	the number of replica hosts		
3	**Stress-Test 1:** Number of read operations completed in a second (throughput) for objects under 1MB and under different concurrency and congestion loads	Strict concurrency to affect negatively the performance of the algorithm while more stochastic (more realistic) loads will perform better.	We created two types of scenarios: (1) To examine the throughput under **different topologies**, i.e., different combinations of continents (EU, US) for clients and servers. (2) To examine the throughput, while we varied the **number of servers from 3 to 15**. Indeed, the throughput changes depending on the topology.
4	**Stress-Test 2:** Time to take a read/write to complete under different object sizes	Keep operations latency close to the expected delay of transferring an object of a predefined size over a dedicated connection bandwidth.	We evaluated how the latencies are affected by the size of the shared object. The **object size** kept increasing by doubling it from **64 KB to 8 MB**.
5	**Fault-Tolerance Test 1:** Service Interruption	We expect the service to experience no interruptions while allowing replica failure crashes to happen concurrently with reconfigurations.	We introduced **two replica server failures** in the ARES algorithm while allowing reconfigurations in the service. No service interruptions were recorded.
6	**Fault-Tolerance Test 2:** Time it takes for the service to reconfigure on replica failure or on replica removal	Measured in seconds and targeted to be linear with respect to the number of concurrent reconfigurations.	We evaluated the latency it takes for the service to **reconfigure**: (i) while the reconfiguration switches between **different algorithms** (i.e., the ABD and EC DAPs); and (ii) while the **number of reconfigurers** increases from 1 to 3.
7	**Performance Comparison Test:** Compare the algorithm performance with other ADSS	We expect that ARES may have additional performance overhead over simpler solutions that do not support dynamicity or use centralized control.	Except from the ARES algorithm with the two DAPs, we also set-up commercial algorithms, **Cassandra** and **Redis**.

EXTRA KPI:

KPI	Measure	Achieved
8	**Stress-Test 3:** Examine the read and write latencies with different numbers of the fragmentation parameter k, in the Reed-Solomon algorithm.	We increased the k of the EC algorithm from 2 to 10.

Co-funded by the Horizon 2020 Framework Programme of the European Union

5.1 Experimental Evaluation

The table below provides a comprehensive list of the variables we used in our scenarios (as discussed in Sec. 4.1). Experiments were conducted for a selection of those parameters.

Variable	Description	Possible Values
topology	The distribution of servers in EU and US. For the scenarios with more than 3 servers we use two servers in US for every server in EU.	{0E+3U, 1E+2U, 2E+1U, 3E+0U}
Client Continent	The location of the clients for the throughput scenario.	{EU, US}
S	The number of servers.	{3, 5, 7, 9, 11}
W	The number of writers.	{0, 1, 5, 10, 15, 20}
R	The number of readers.	{0, 1, 5, 15, 50, 100, 150, 250}
G	The number of reconfigurers in the service.	{0, 1, 3, 5}
k	The erasure-coding data fragments.	{1, 2, 3, 4, 5, 6, 7, 8, 9}
fsize	The size of the file object.	{64 Kb, 128 Kb, 256 Kb, 512 Kb, 1 Mb, 2 Mb, 4 Mb, 8 Mb}
recontype	The way the reconfigurers work: (i) reconfiguring to the same DAP, (ii) reconfiguring the DAP alternately, (iii) reconfiguring the DAP alternatively and servers randomly.	{ sameDAP, switchingDAP, switchingDAP&randomServers }

For all the experiments, we used a stochastic invocation scheme in which read operations are scheduled randomly in the interval [1...*rInt*] and writes in the interval [1..*wInt*], where *rInt*, *wInt* = 3sec. In total, each writer performs 50 writes and each reader 50 reads. The reconfigurer invokes its next operation every 15sec and performs a total of 15 reconfigurations. In throughput experiments, there is no delay between invokes, and the clients perform 1000 operations each.

Performance of the algorithms is measured in terms of the time it takes for their operations to terminate. Thus, for each algorithm we measure the average operation latency, starting at the invocation to the response and taking in account both the communication as well as the computation overhead. The operation latency is computed as the average of all clients' average operation latencies. It is worth mentioning that in the case of Cassandra we omitted to account some unsuccessful operations, where clients did not receive replies from a majority.

Next, we highlight the most informational outcomes in each scenario. More results may be found in the website of the project (https://projects.algolysis.com/ares-ngi/results) presented in interactive plots where the user may choose the parameters to apply.

Co-funded by the Horizon 2020 Framework Programme of the European Union

5.1.1 Scalability Tests

This scenario is constructed to compare the read and write latencies of the algorithms, as the number of service participants increases. We varied the number of readers |R| from 5 to 250 and the number of writers |W| from 5 to 20. The number of servers |S| is set to two different values, 3 and 11. We calculate all possible combinations of readers, writers, and servers where the number of readers or writers is kept to 5. In total, each writer performs 50 writes and each reader 50 reads. The size of the object used is 1 MB. We used a different parity for EC algorithm. The parity value of the EC algorithm is set to m = 1 for |S| = 3 and m = 5 for |S| = 11.

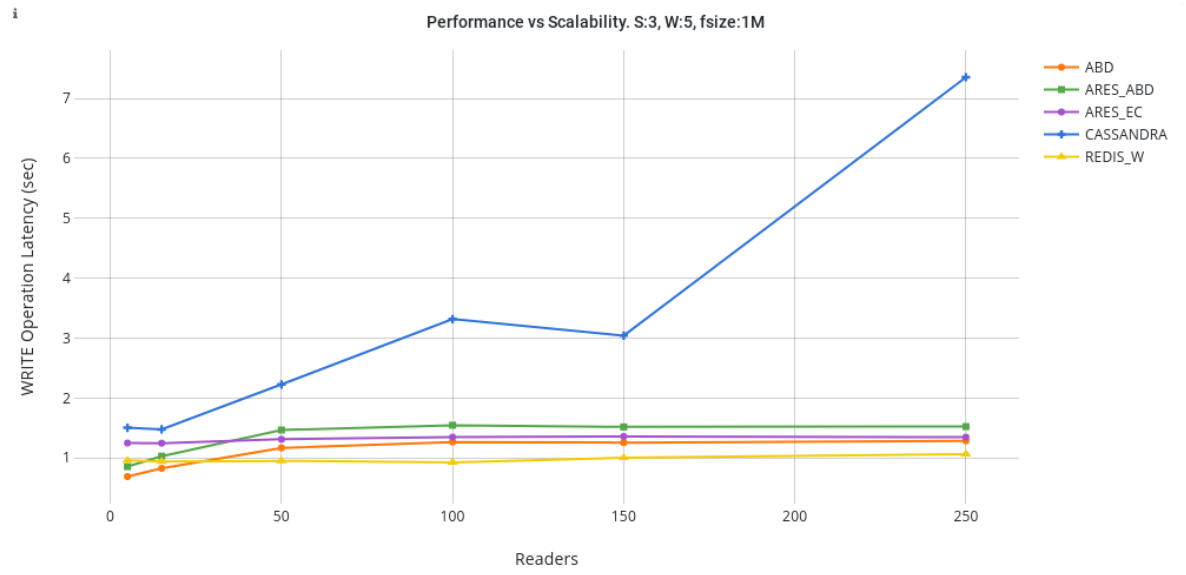


Figure 5: Readers Scalability vs Write Performance, S:3

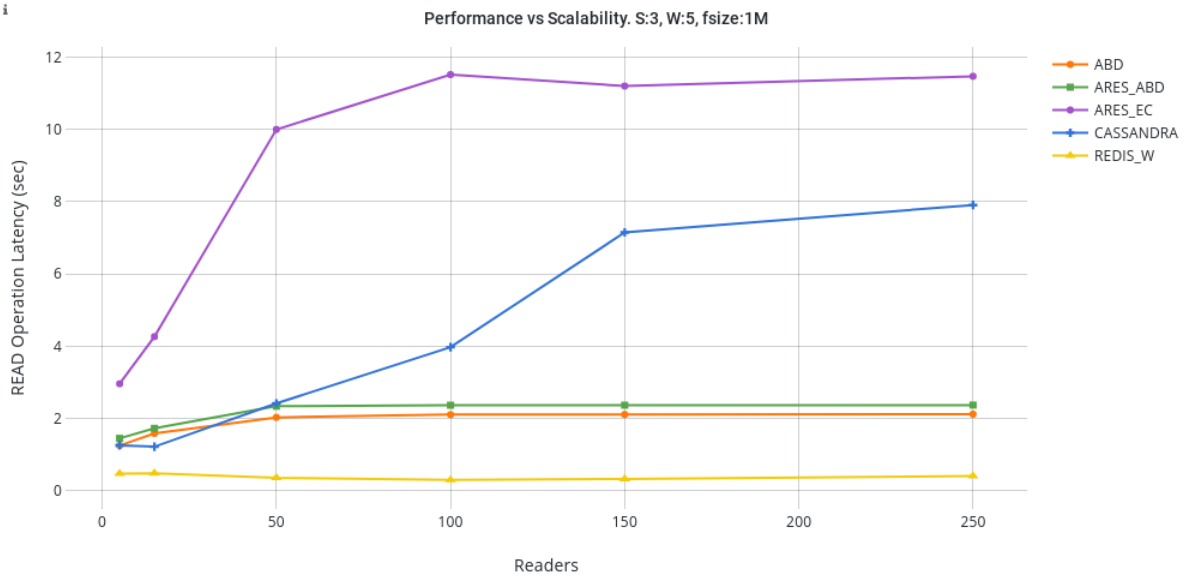


Figure 6: Readers Scalability vs Read Performance, S:3.

Co-funded by the Horizon 2020 Framework Programme of the European Union

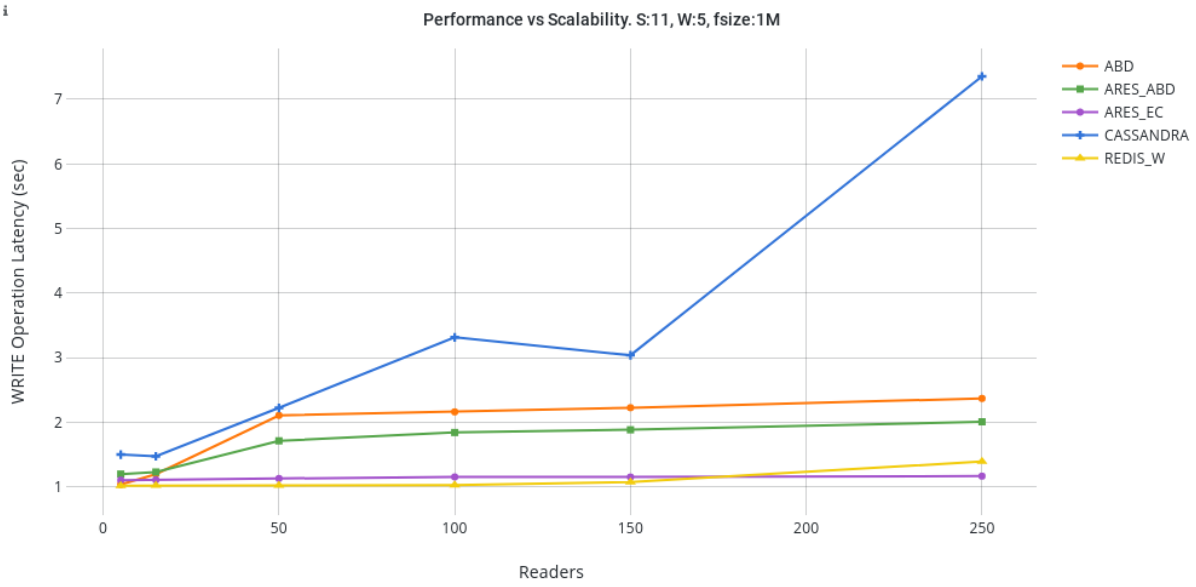


Figure 7: Readers Scalability vs Write Performance, S:11.

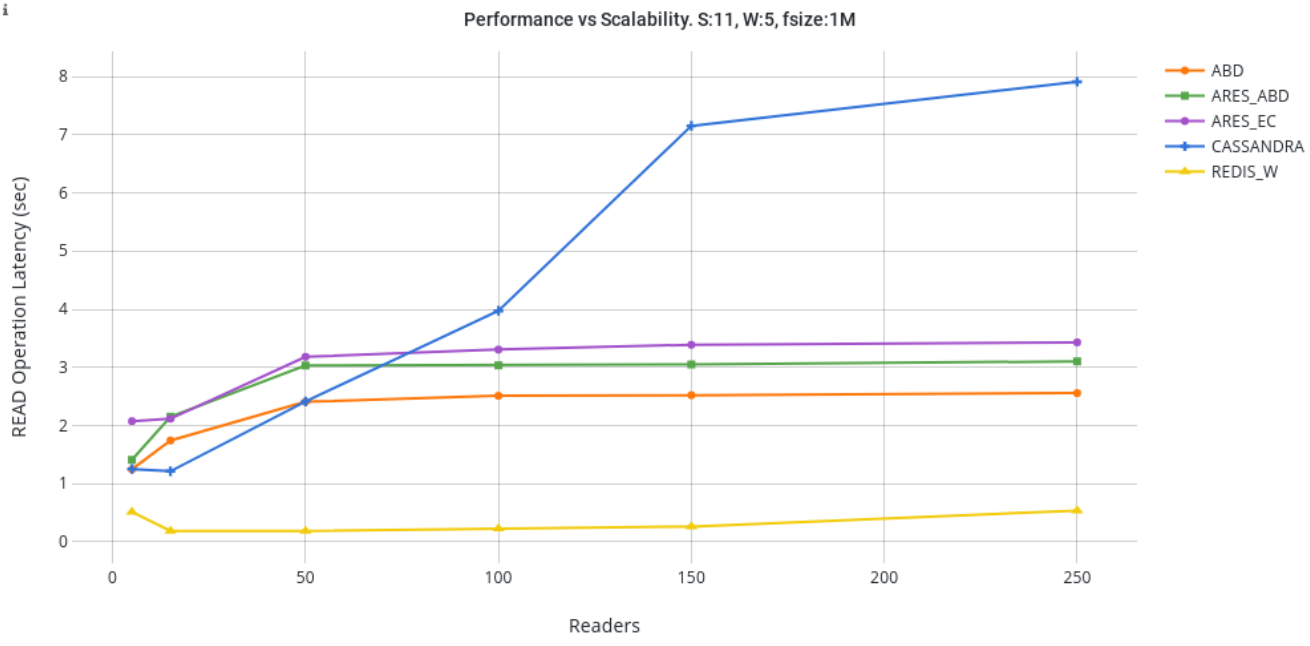


Figure 8: Readers Scalability vs Read Performance, S:11.

Co-funded by the Horizon 2020
Framework Programme of the European Union

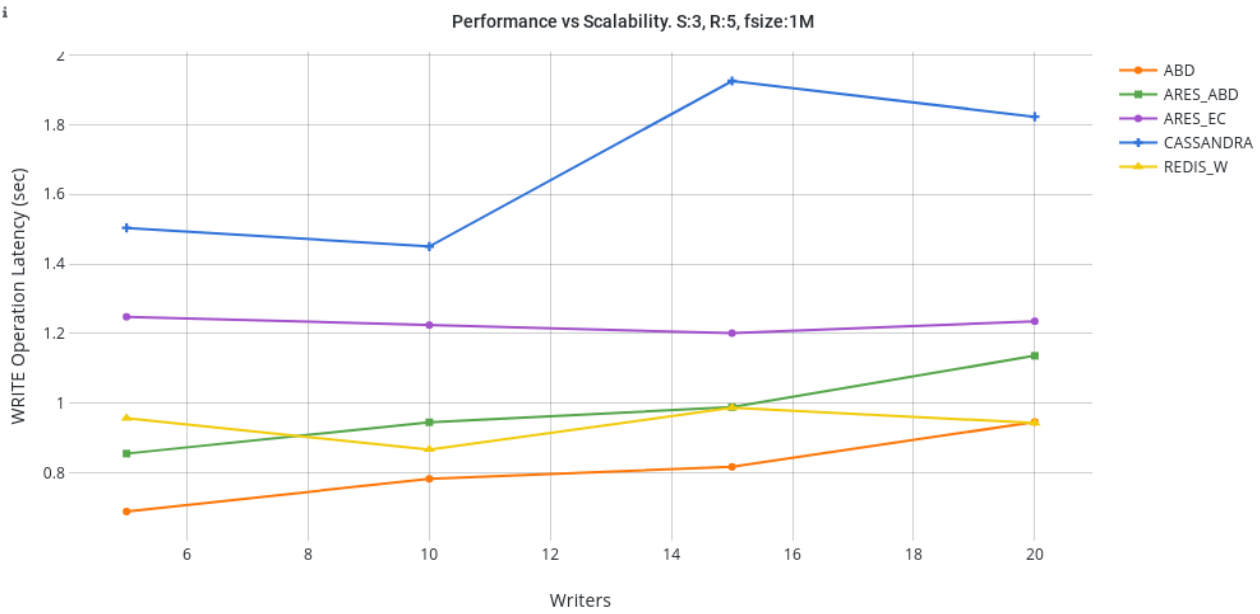


Figure 9: Writers Scalability vs Write Performance, S:3.

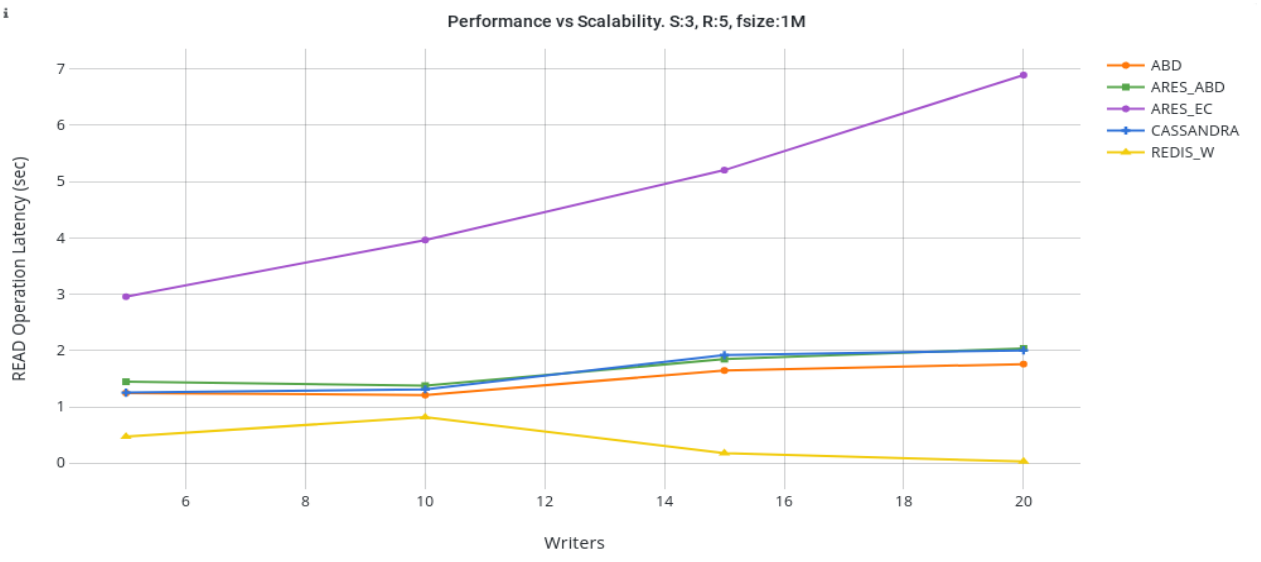


Figure 10: Writers Scalability vs Read Performance, S:3.

Co-funded by the Horizon 2020 Framework Programme of the European Union

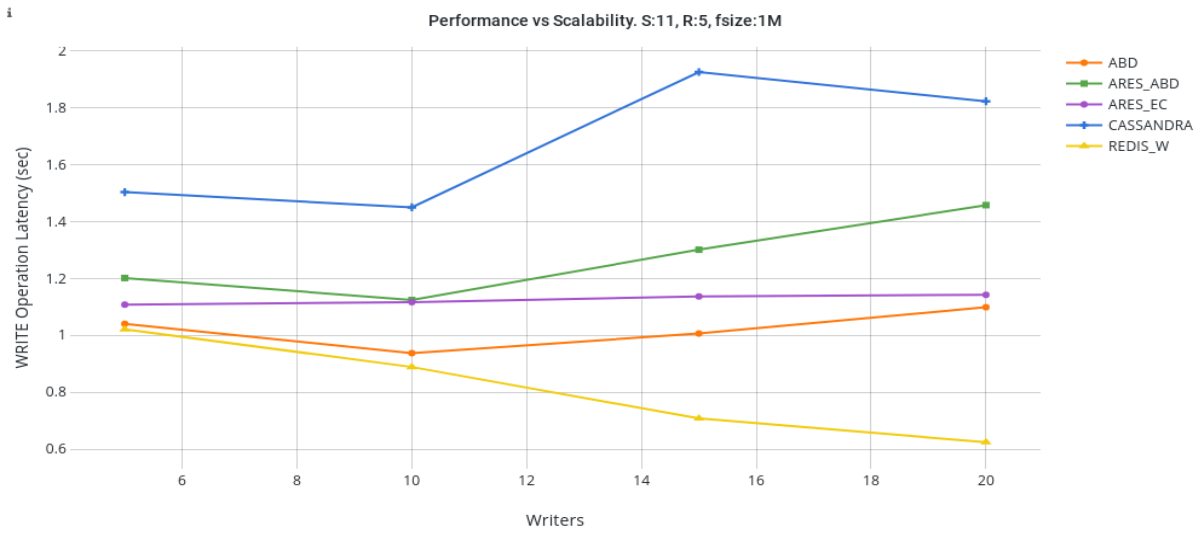


Figure 11: Writers Scalability vs Write Performance, S:11.

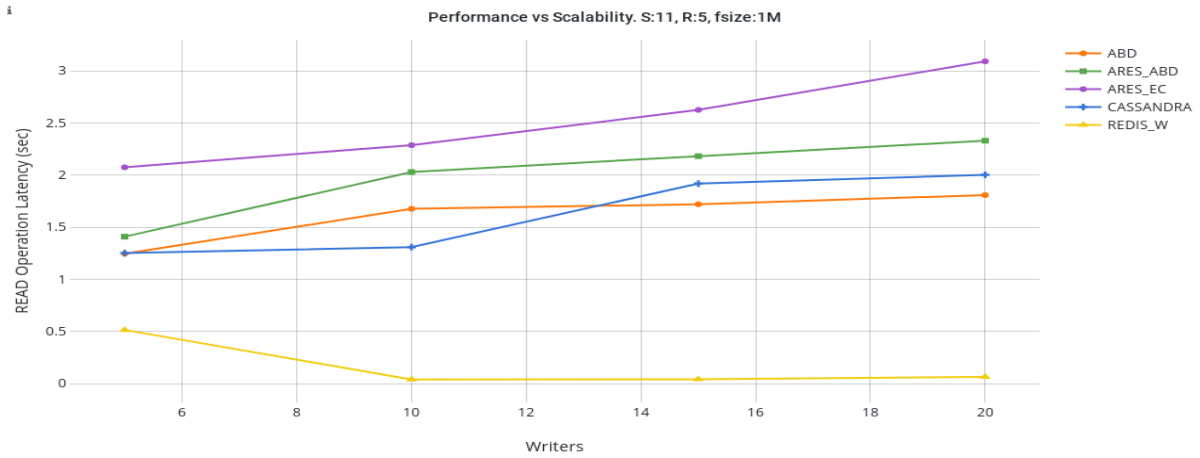


Figure 12: Writers Scalability vs Read Performance, S:11.

Scalability Results: Results obtained while increasing the number of participants in the system appear in Figs. 5 –12. At a first glance, CASSANDRA seems to struggle to keep up as the readers grow in all cases, while REDIS_W does not seem to be affected. Similar observation can be made for the two ABD based algorithms (ABD and ARES_ABD) as they remain at low levels as |R| increases. ARES-EC exposes an interesting behavior as it is the worst performing algorithm when few servers are used, and becomes faster when more servers are deployed. This can be seen in Figs. 7 and 8. The more the servers the more the encoded elements to be distributed and the bigger can be the fragmentation parameter k. Thus, each object fragment becomes smaller, resulting in tremendous benefits on the communication delays. Worth observing is that the latency of the write operation of ARES_EC matches the one of REDIS_W when |S| = 11. Similar findings can be seen as the number of writers |W| grows (Figures 9-12). CASSANDRA has the larger write latency even though it shows a more stable behavior, and the read latency of ARES_EC is the worst when |S| = 3.

5.1.2 Stress Tests - Topology

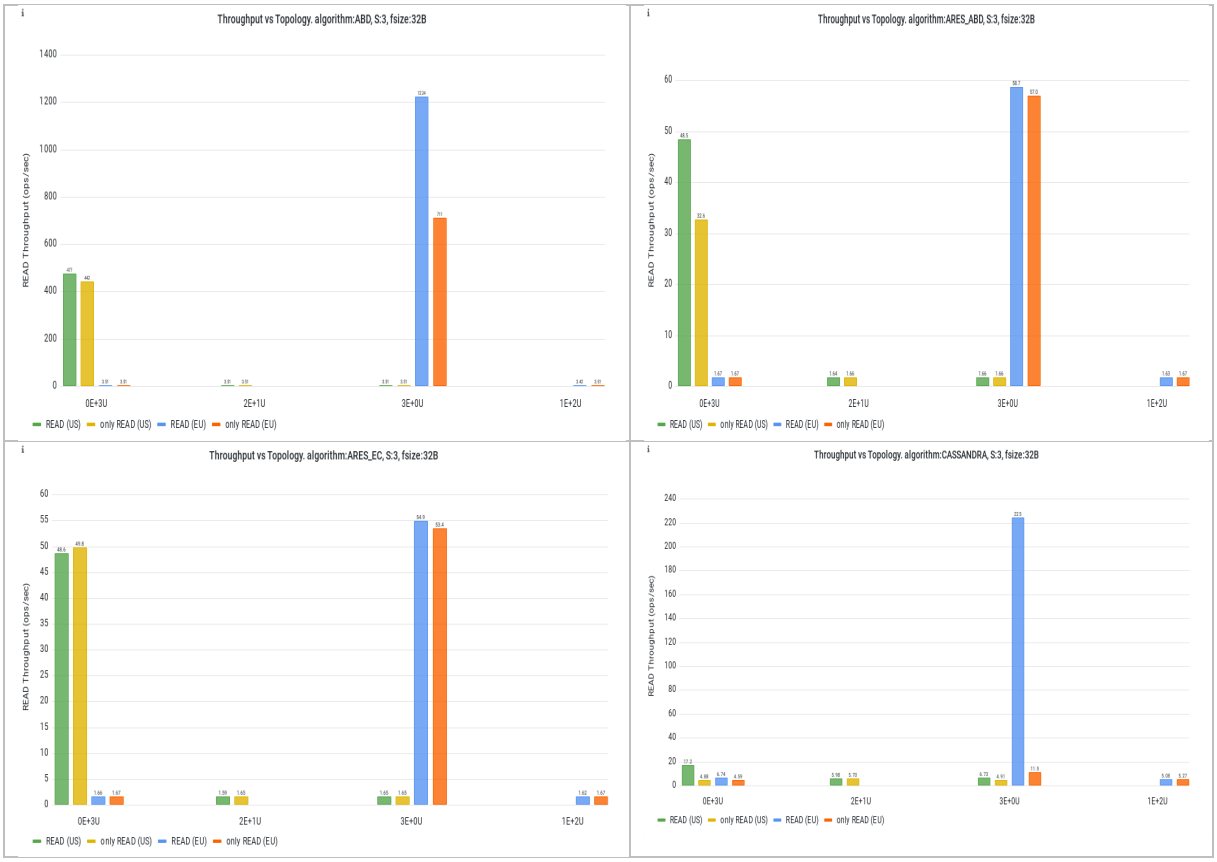
This scenario aims to measure how the performance of the algorithms is affected under different topologies and server participation. In this case we measure the throughput (average number of operations per second) of each algorithm. To avoid any delays due to operation

Co-funded by the Horizon 2020 Framework Programme of the European Union

contention, we chose to use 2 clients (1 reader and 1 writer), the minimum number of servers to form a majority, i.e. 3, and a simple object of 32 B. As we deployed machines on both EU and USA, our servers are split in such a way to either force all of them or their majority to be in a single continent. We considered two scenarios: (1) we have 3 servers under different topologies; (2) we varied the number of servers from 3 to 15.

Scenario 1: The experiments were performed for 3 servers. The three servers were distributed based on the topologies listed in the table below. In each topology, xY means that x servers are deployed in Y continent for E = EU and U = USA. Similarly, we deployed the clients either close (i.e., to the same continent) or away from the server majority. For each topology we run 2 clients (1 writer and 1 reader), only 1 writer, and only 1 reader. The first and last topologies multiply each combination much more when they are conducted for both US and EU clients. The second and third topologies, however, are only made available to EU and US clients, respectively.

Topology	Servers in EU	Servers in US
0E+3U	0	3
1E+2U	1	2
2E+1U	2	1
3E+0U	3	0



Co-funded by the Horizon 2020 Framework Programme of the European Union

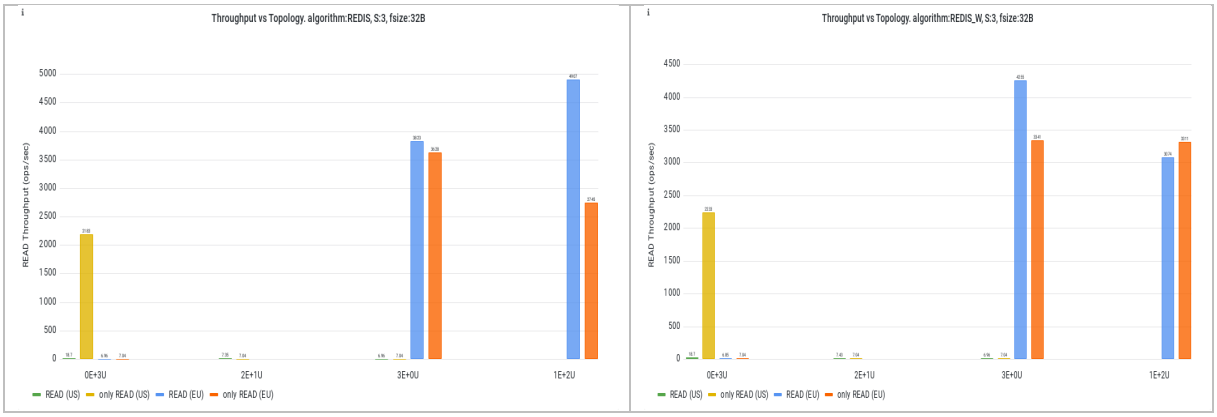
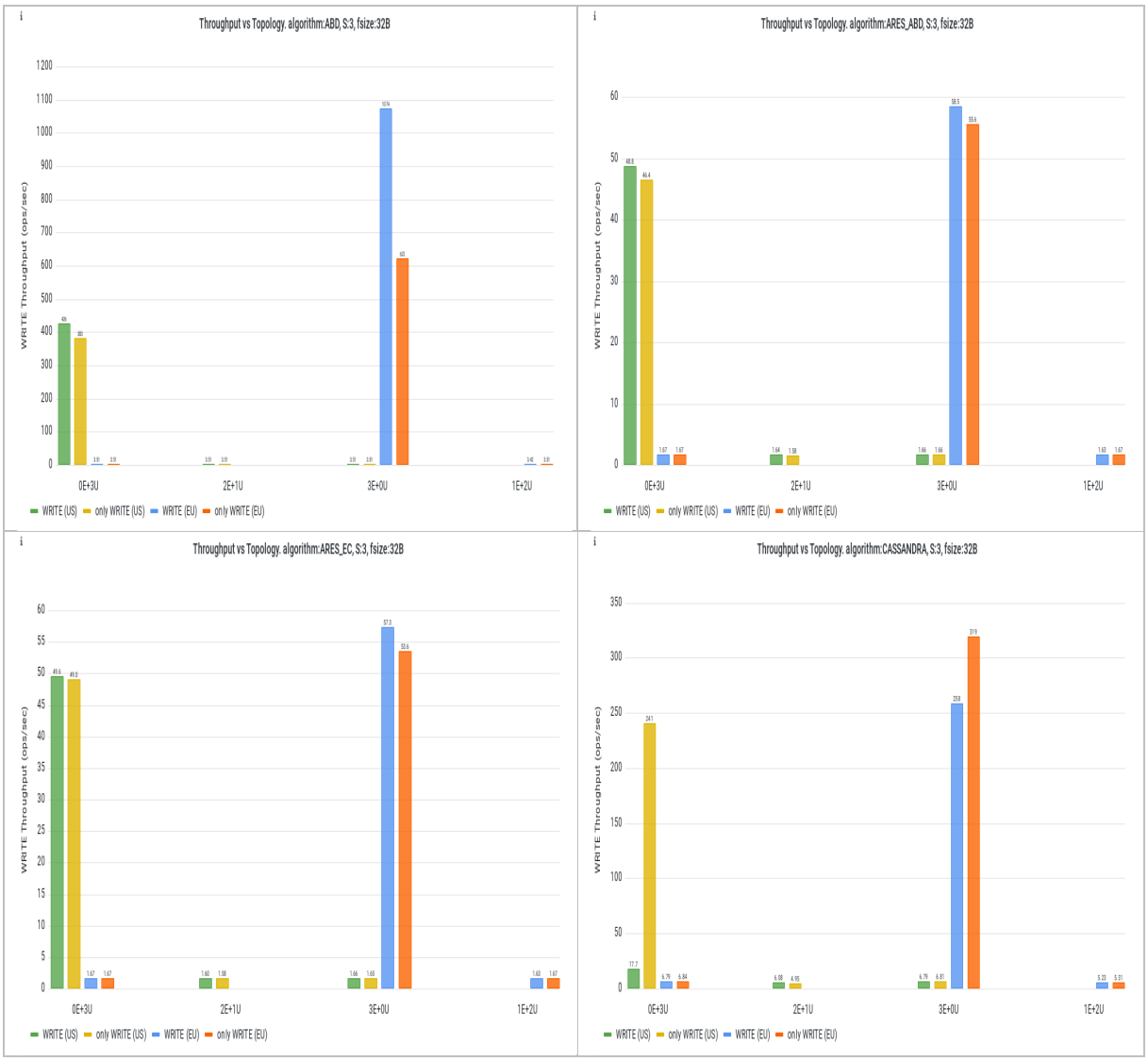


Figure 13: Read Throughput vs Topology.



Co-funded by the Horizon 2020
Framework Programme of the European Union

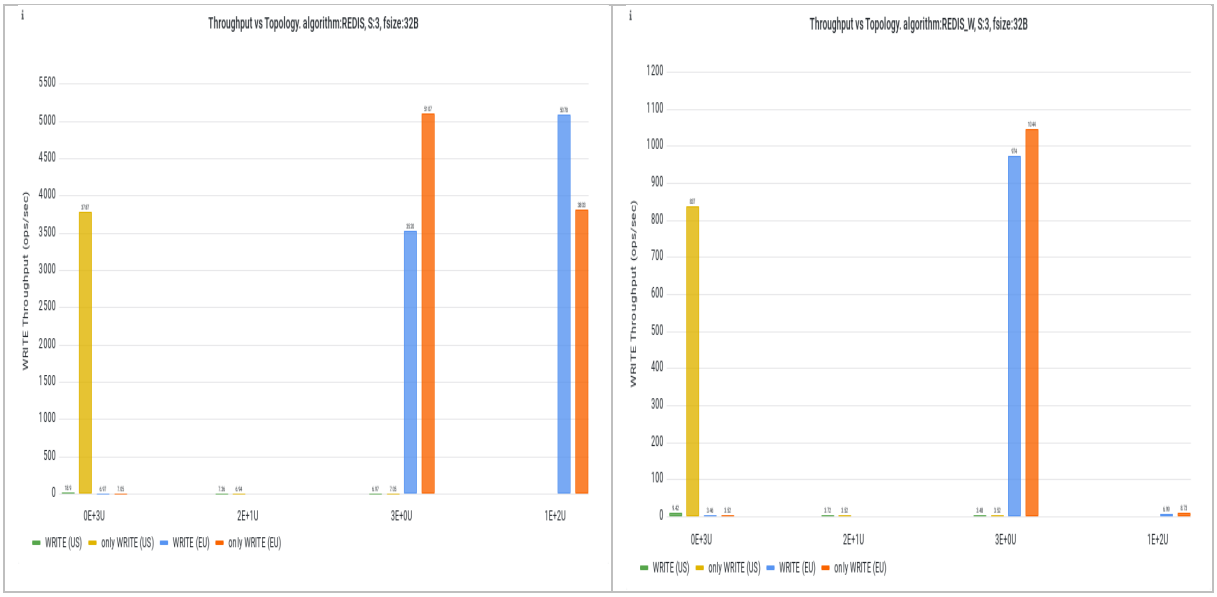


Figure 14: Write Throughput vs Topology.

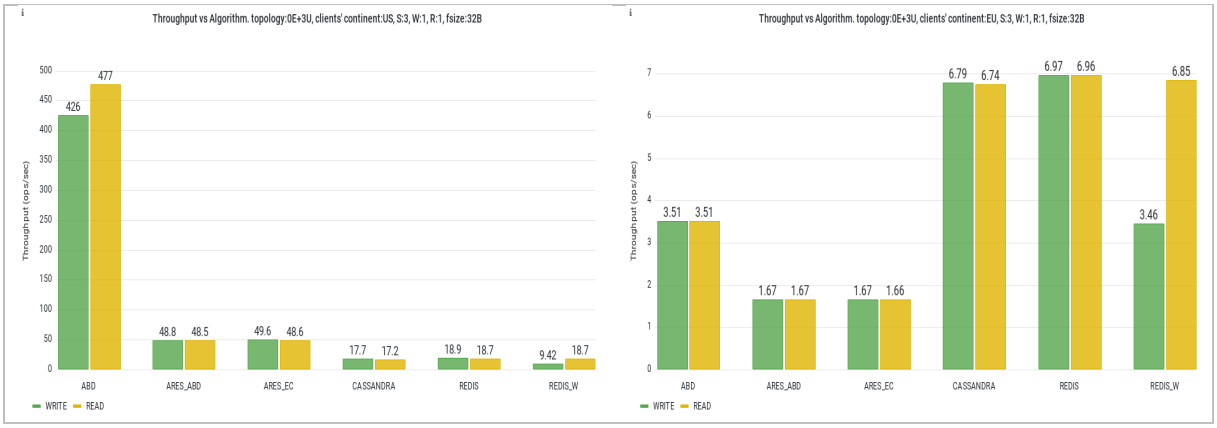


Figure 15: Throughput vs Algorithm. topology:0E+3U, W:1, R:1.

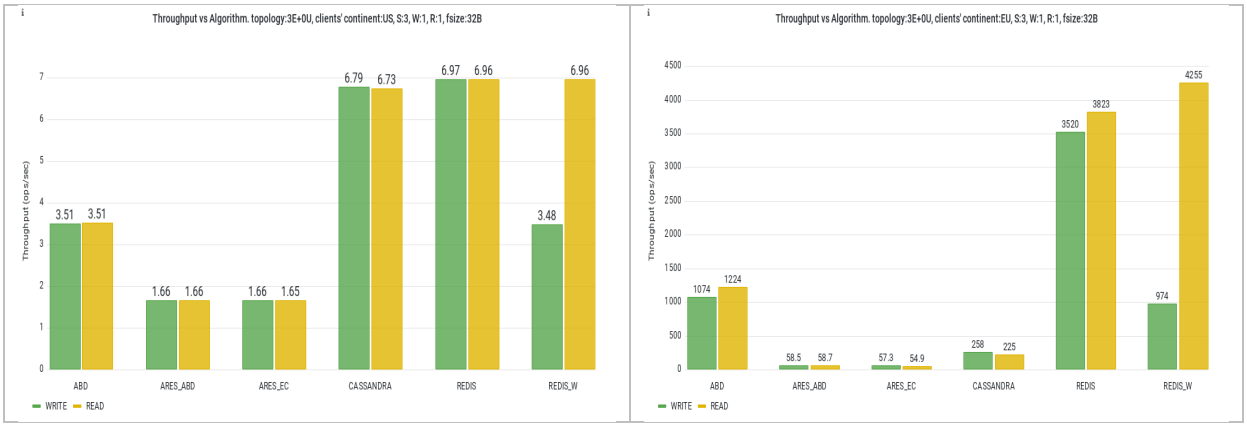


Figure 16: Throughput vs Algorithm. topology:3E+0U, W:1, R:1.

Co-funded by the Horizon 2020
Framework Programme of the European Union

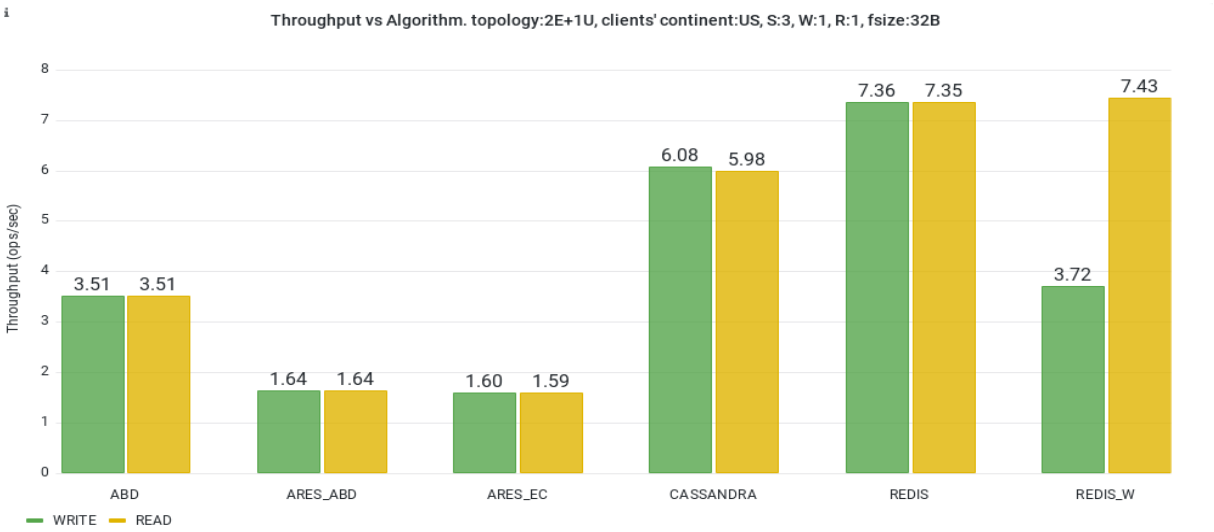


Figure 17: Throughput vs Algorithm. topology:2E+1U, W:1, R:1.

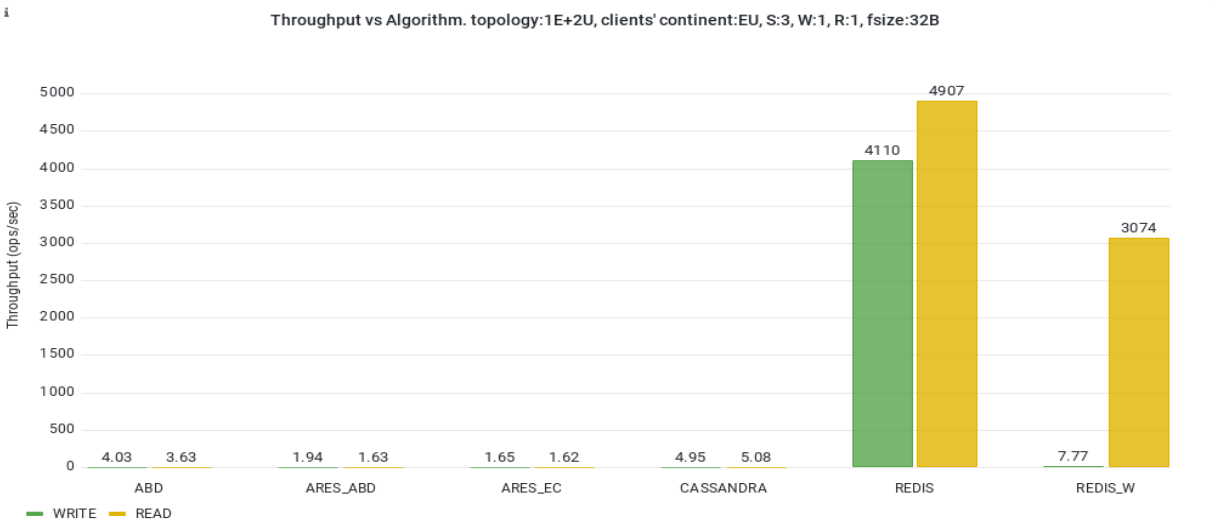


Figure 18: Throughput vs Algorithm. topology:1E+2U, W:1, R:1.

Co-funded by the Horizon 2020
Framework Programme of the European Union

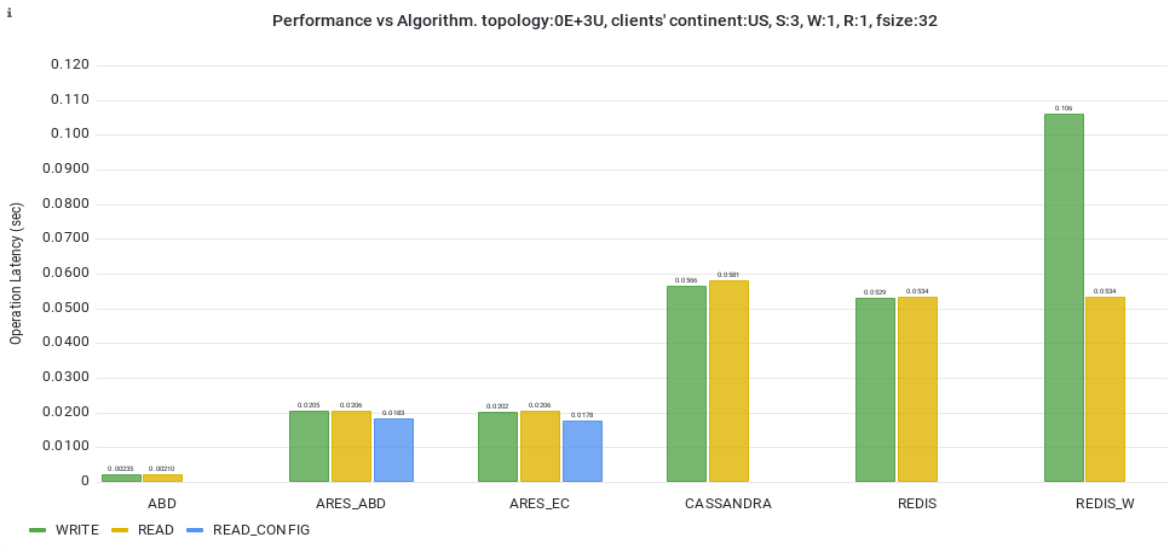


Figure 19: Performance vs Algorithm. topology:0E+3U, W:1, R:1.

Scenario 1 Results: Figure 13Figure 14 illustrate the average number of operations per second (throughput) for each algorithm under different topologies and under different placement of the reader and writer clients (EU or US).

The three algorithms we developed are shown to achieve their maximum read and write throughput when the servers and the users belong to the same continent. Also, there appears to be no difference when the experiment contains concurrent or non-concurrent operations. Finally, the small *fsize* (32 B), amplified the impact of the stable overhead of read-config operations, and they constitute a significant percentage of the total operation latency (see blue bar in Figure 19). From the same figure we interestingly observe that the setup where all servers and clients are deployed in the USA, favoured ARES and ABD algorithms over both Cassandra and Redis.

On the other hand, CASSANDRA shows different behaviour in the topologies between the read and write operations. It achieves the maximum read throughput when both servers and clients belong to the EU. Apart from this, in the write operation, CASSANDRA also succeeds in large throughput when there is only one writer in both topologies where the servers and clients belong to the same continent (EU or US). This shows that the read operation of Cassandra is delaying the write operation when they coexist. REDIS and REDIS_W show similar behavior according to read operation. They achieve their maximum read throughput in all experiments where there are both EU servers and EU clients (of any number). They have a large throughput in the experiments where there is only one writer, and all the participants belong to the US. However, in the write operation, REDIS with WAIT command (REDIS_W) stops having high throughput in the last topology (1E+2U).

Figure 15**Error! Reference source not found.** compare the throughput of the algorithms under the different topologies when there are both read and write operations. Figure 15 shows that when both servers and clients belong to the US, our algorithms perform many more operations per second compared to the three others (CASSANDRA, REDIS, REDIS_W). In the same topology but with clients in the EU, all algorithms show smaller throughputs, but our algorithms have the greatest reduction. In the topology 3E+0U, the same behaviour continues to apply for our algorithms and CASSANDRA. However, the two variants of Redis show better throughput for both clients' continents. In the remaining two topologies (2E+1U and 1E+2U) we can see that in the second case where there are EU clients, all the algorithms have better

Co-funded by the Horizon 2020 Framework Programme of the European Union

throughputs, especially the redis variants. As it was expected, in all experiments, REDIS_W performs fewer operations per sec compared to REDIS (without the wait command).

Scenario 2: This scenario examines the throughput and performance of the algorithms when the number of servers is growing from 3 to 15. In this case, for every server deployed in EU, we deployed 2 servers in the USA. Clients were deployed both in the US and the EU.

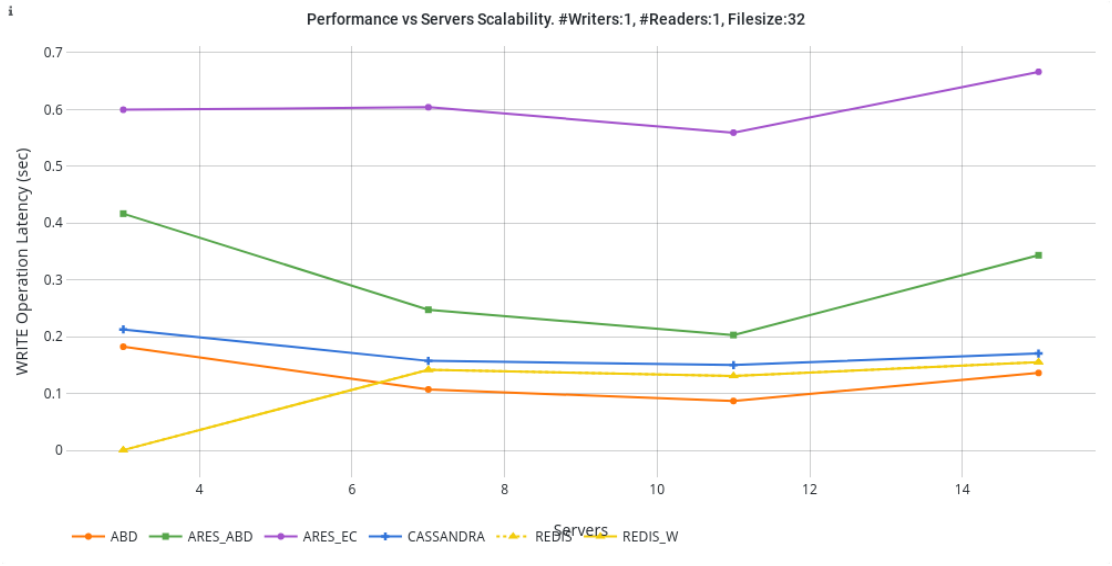


Figure 20 Write Performance vs Servers Scalability. W:1, R:1.

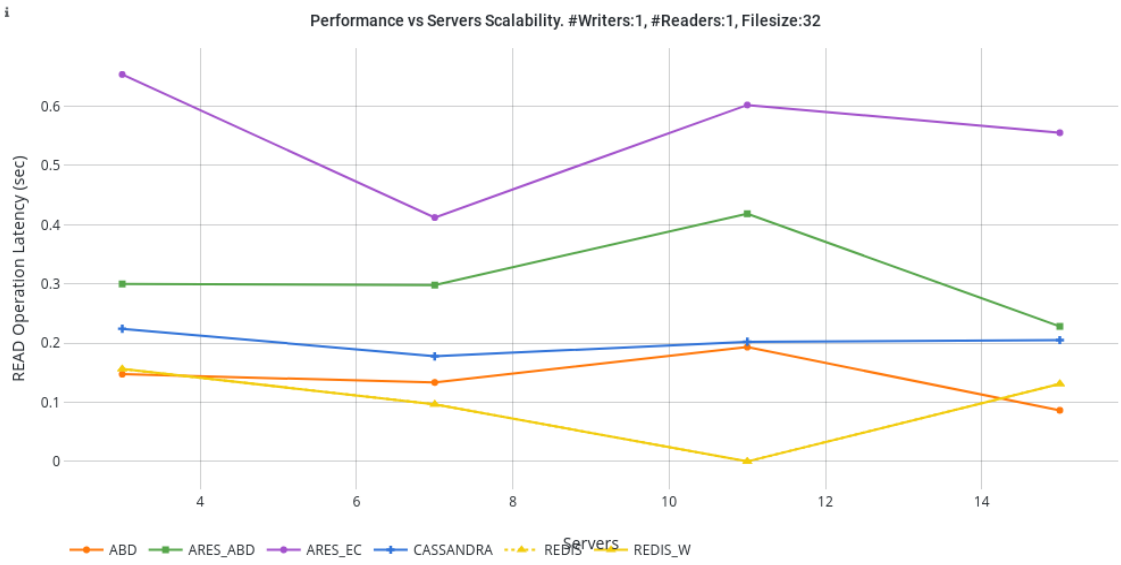


Figure 21 Read Performance vs Servers Scalability. W:1, R:1.

Scenario 2 Results: Figure 20 and Figure 21 show the results of this scenario for write and read latencies respectively. As it was expected for such a small size, the ARES algorithm has larger latencies. Both ARES variants have the standard overhead of the read-config operations which we have already discussed above. At such a small file size, the ARES_EC may not benefit so much from the encoding and decoding. However, the ABD algorithm outperforms CASSANDRA and, in some points, is also better than the two variants of REDIS algorithms. The two Redis variants are very close to each other.

Co-funded by the Horizon 2020 Framework Programme of the European Union

5.1.3 Stress Tests - Object Size

This scenario is made to evaluate how the read and write latencies are affected by the size of the shared object. The file size doubled from 64 KB to 8 MB. The number of servers is fixed to 11. The number of writers and the value of delta are set to 5; delta being the maximum number of concurrent put-data operations. The number of readers is fixed to 5. For ARES algorithm there are two separated runs, one for each examined storage algorithm, ARES_ABD and ARES_EC. The parity parameter *m* of ARES_EC is set to 5. The quorum size of the ARES_EC algorithm is $\left\lceil \frac{11+6}{2} \right\rceil = 9$, while the quorum size of ARES_ABD algorithm is $\left\lfloor \frac{11}{2} \right\rfloor + 1 = 6$. For the CASSANDRA algorithm, we set the replication factor (RF) to the majority, i.e., 6. The writers of REDIS_W also wait for a majority (6) servers to reply.

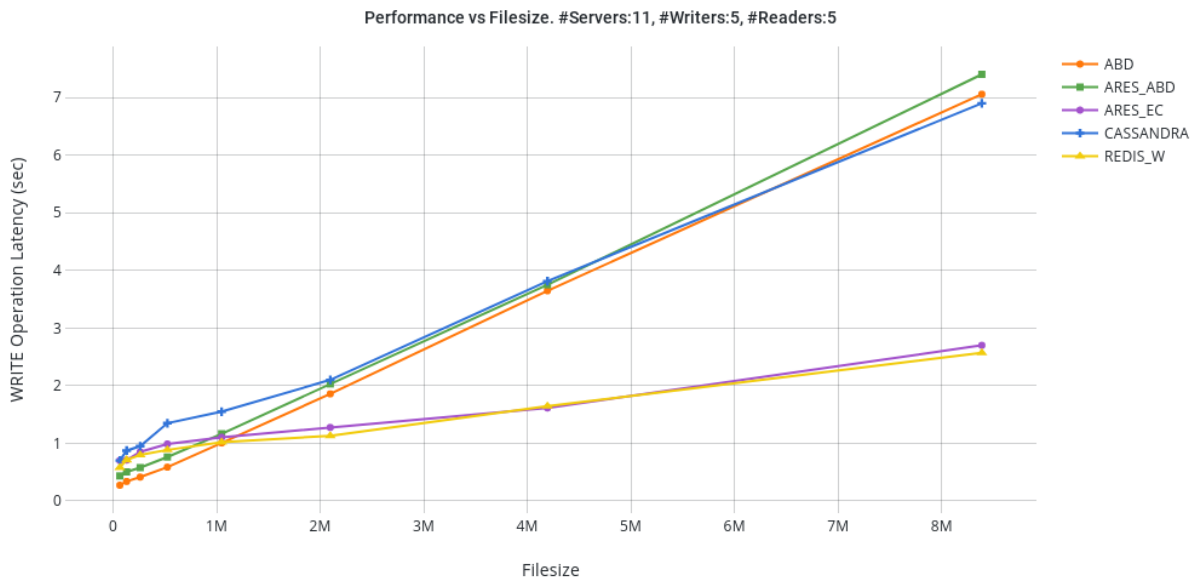


Figure 22: Object Size vs Write Performance.

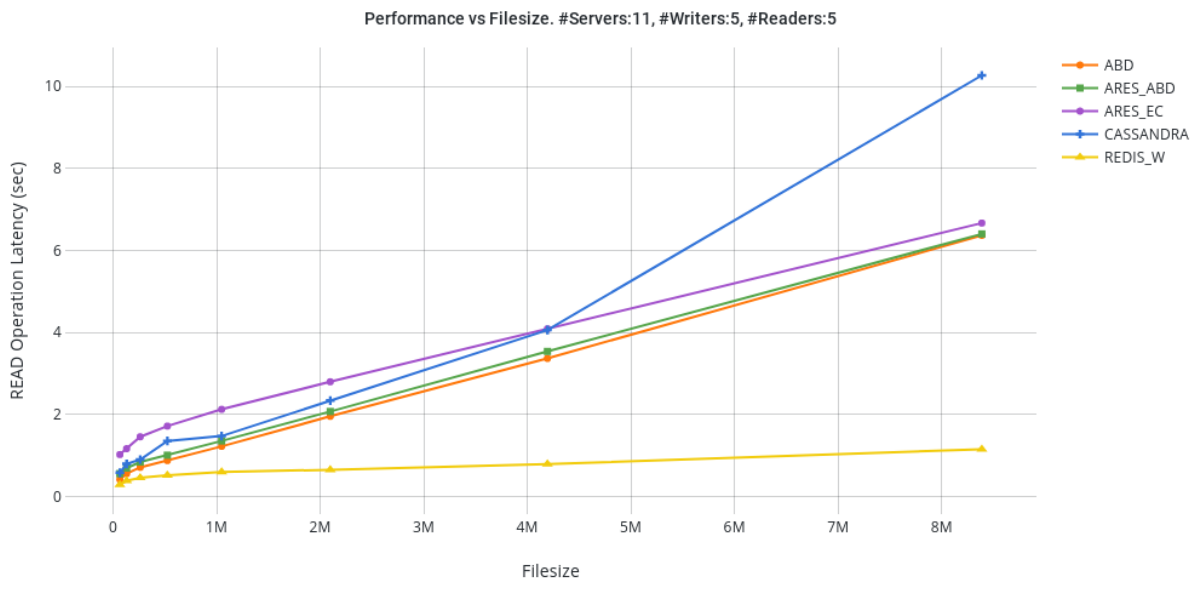


Figure 23: Object Size vs Read Performance.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Object-Size Results: The results of these experiments are captured in Figure 22 and Figure 23. In Figure 22, we observe that the write latencies of all operations, except ARES_EC and REDIS_W, grow significantly as the *fsize* increases. The fragmentation applied by the ARES_EC benefits its write operations, which follow a slower increasing curve like the REDIS_W. The write latencies of all other algorithms are close to each other. Results in Figure 23 show that read operations of ARES_EC suffer the most delays until 4 MB. It is also worth mentioning that the first phase of the read operation does decoding, which is slower than the first phase of the write, which simply finds the maximum tag. However, at larger file sizes (8 MB) CASSANDRA has the slowest read operations. As expected, the REDIS_W read operations provide the best results, and its write operations with the WAIT command have higher latency compared to the read operations. However, both of them remain at low levels as the *fsize* increases.

5.1.4 Stress Tests - Fragmentation Parameter k

This scenario applies only to ARES_EC since we examine how the read and write latencies are affected as we modify the erasure-code fragmentation parameter k (a parameter of Reed-Solomon). We assume 11 servers and we increase k from 2 to 10. The number of writers (and hence the value of δ -- the maximum number of concurrent write operations) are set to 5. The number of readers is fixed to 15 and the size of the object used is 4 MB.

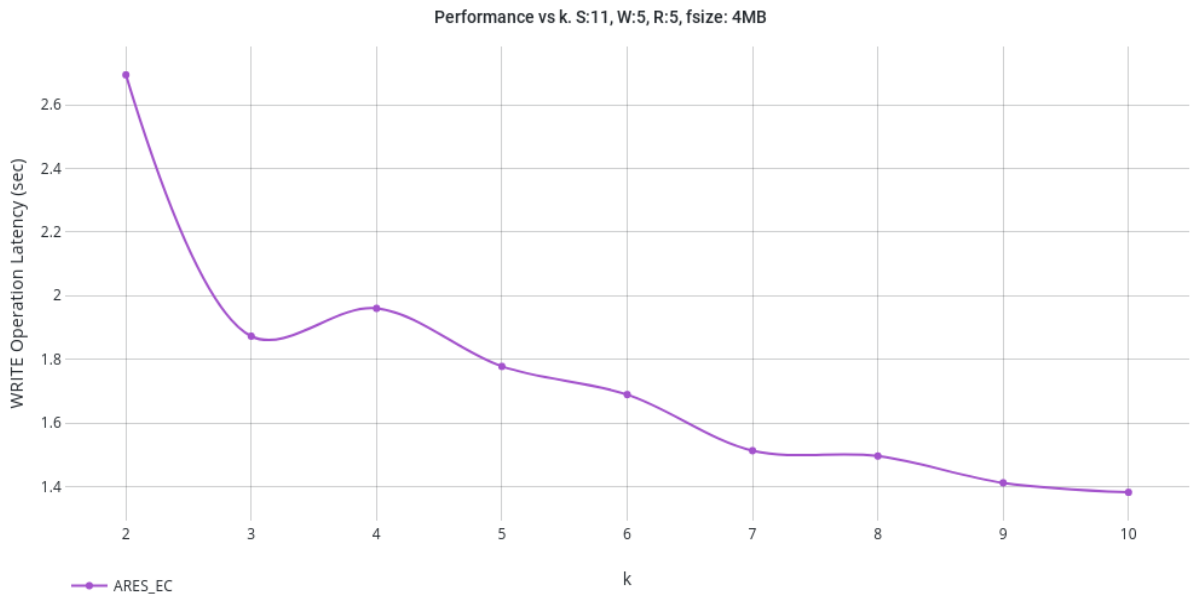


Figure 24: k Scalability vs Write Performance.

Co-funded by the Horizon 2020 Framework Programme of the European Union

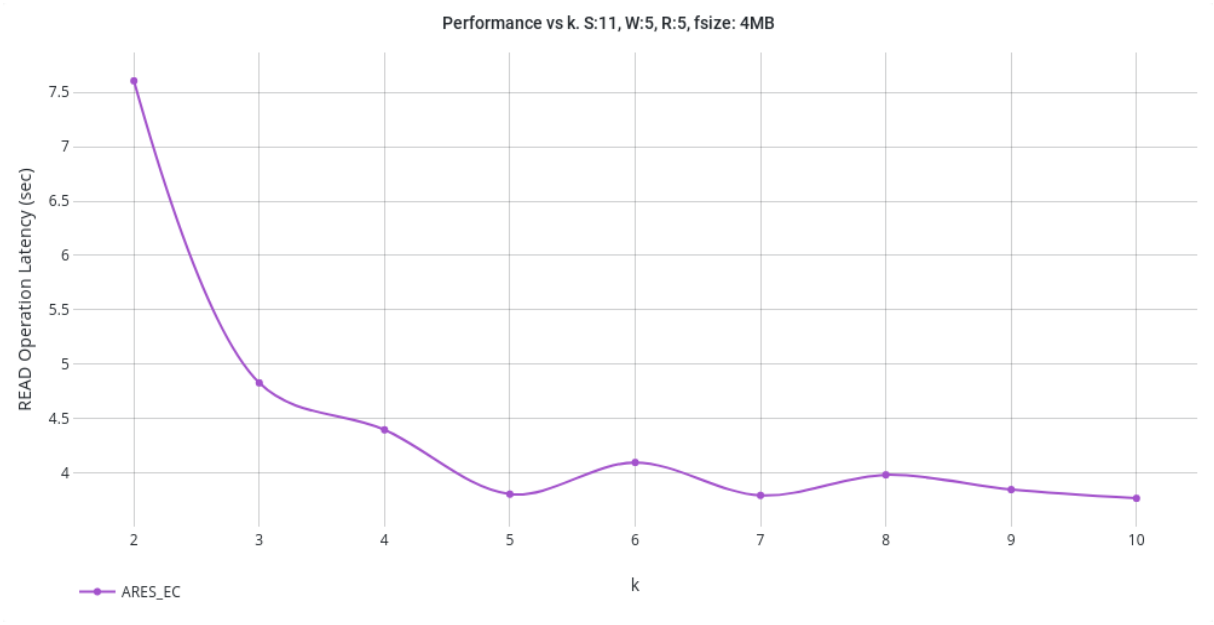


Figure 25: k Scalability vs Read Performance.

Fragmentation parameter Results: From Figure 24 and Figure 25 we can infer that when smaller k is used, the write and read latencies reach their highest values. In both cases, small k results in the generation of fewer but bigger data fragments and higher redundancy. For example, consider the cases RS(11,7) and RS(11,6), setting k=7 and k=6 respectively. This implies that we fragment the initial object into 7 and 6 fragments in each case, with RS(11,6) to have bigger fragments than the ones obtained in RS(11,7). Notice that in both cases each client waits for $\left\lceil \frac{11+7}{2} \right\rceil = \left\lceil \frac{11+6}{2} \right\rceil = 9$ servers to reply, thus tolerating in both cases 2 server failures. However, as the fragments are bigger when k=6, this generates an impact on the communication delay of each operation and thus on their average latency. The write latency seems to be less affected by the number of k since the write operation only encodes and does not decode the object value, while the read operation does both. In conclusion, there appears to be a trade-off between operation latency and fault-tolerance in the system: the further increase of the k (and thus lower fault-tolerance) the smaller the latency of read/write operations.

5.1.5 Fault-Tolerance – Server Crashes

In this scenario, we introduced server fail-crashes in the ARES algorithm to verify the fault-tolerance guarantees and the responsiveness of the system. The number of servers |S| is set to 11 with m = 5. We varied the number of reconfigurers from 1 to 3. Each reconfigurer switches between the two DAPs. The numbers of writers and readers are fixed to 5 and 15, respectively. The size of the object used is 1 MB. We execute 2 crashes during each experimental run: server0 crashes 100 seconds after the start of the experiment and server3 crashes 200 seconds after. Both failed servers are from the imec Virtual Wall 2 testbed (EU), since we observed that they are included in the most quorum replies.
We assign a unique id to each quorum. However, the quorum size of each DAP was different: for ARES_ABD is 6, while the quorum size of ARES_EC is 9. In total, ARES_ABD has 462 quorums and ARES_EC has 55. So, for ease of visualization, we categorize the quorums of the two DAPs into three groups: Group 0, which includes all quorums; Group 1, which excludes quorums involving server0; and Group 2, which excludes quorums involving either server0 or server3.

Co-funded by the Horizon 2020 Framework Programme of the European Union

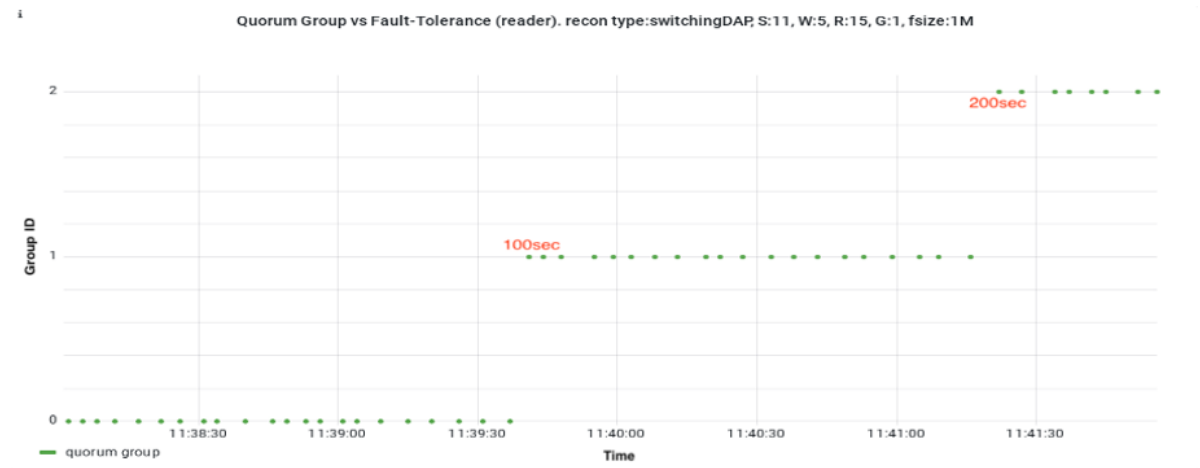


Figure 26: Quorum replies to reader6. Server0 crashes after 100s and Server1 crashes after 200s.

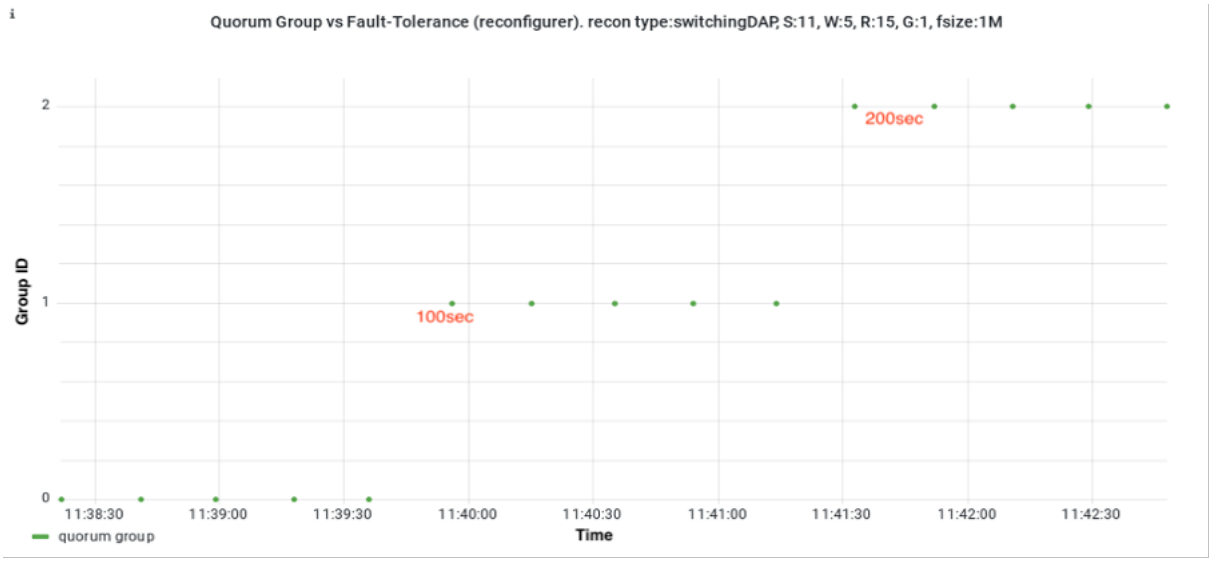


Figure 27: Quorum replies to recon1. Server0 crashes after 100s and Server1 crashes after 200s.

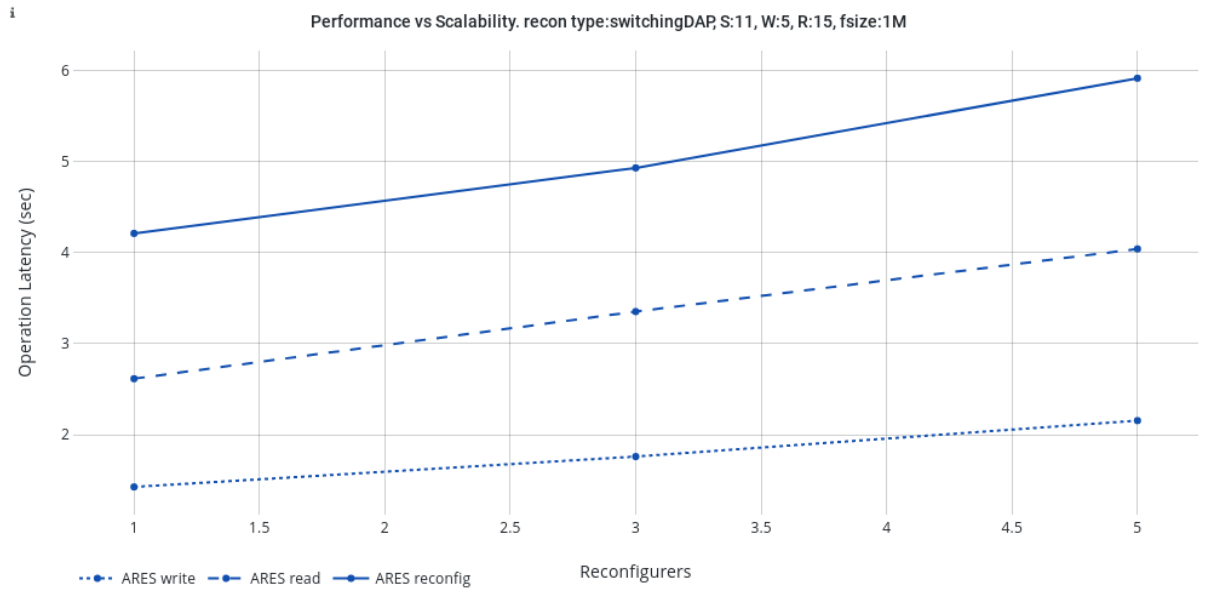


Figure 28: Reconfiguring DAP Alternately and 2 Server Fails.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Fault-tolerance results: Figure 26 and Figure 27 show to which quorum group (0, 1, or 2) the responding servers belong when only 1 reconfigurer exists. That is, Figure 26 shows the quorum group that sends to reader6 every 1 second interval, and Figure 27 the quorum that sends to reconfigurer1. Until the first 100sec of each operation, the quorum group is 0, e.g., all quorums were active in the system. From that moment on, the clients receive responses only from the group 1, e.g., quorums excluded server0. With the second kill, after 200sec, only the quorums included in group 2 remain active. Figure 28 shows the read, write, and reconfig operation latency as the number of reconfigurers increases. During each experiment, the two server failures took place, but our system kept running without interruptions.

5.1.6 Fault Tolerance – Reconfiguration Performance

The scenarios below examine the performance of ARES when reconfigurations are executed concurrently with read/write operations. The experiments differ in the way the reconfigurers work. In the first scenario the reconfigurers introduce a random configuration from a pool of servers and alternate between ARES_EC and ARES-ABD DAPs in each proposed configuration. In the second scenario, we reconfigure always to the same DAP. For the second, we have two separated runs, one for ARES_ABD and one for ARES_EC. In the first scenario the reconfigurers choose randomly the number of servers from the set {3,5,7,9,11}. The parity value of the EC algorithm is set to m = 1 for |S| = 3, m = 2 for |S| = 5, m = 3 for |S| = 7, m = 4 for |S| = 9 and m = 5 for |S| = 11. In the second scenario, the number of servers |S| is kept constant to 11. In both scenarios, we varied the number of reconfigurers from 1 to 3. The numbers of writers and readers are fixed to 5 and 15, respectively. The size of the object is 1 MB.
We ran these experiments with two different deployments of the Raft consensus service to see how the machines' capabilities affect the speed of reconfigurations. At first, we used an external implementation of RAFT deployed on top of Raspberry Pi (RPIs) devices, and subsequently, we deployed RAFT on nodes of the Emulab[10] testbed.

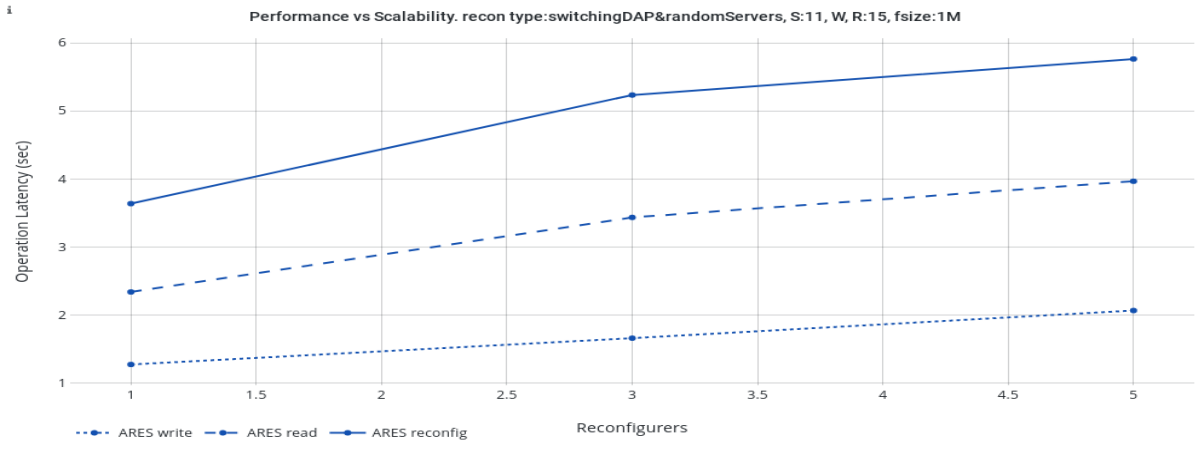


Figure 29: Reconfiguring DAP Alternately and Servers Randomly (RAFT on RPI).

Co-funded by the Horizon 2020 Framework Programme of the European Union

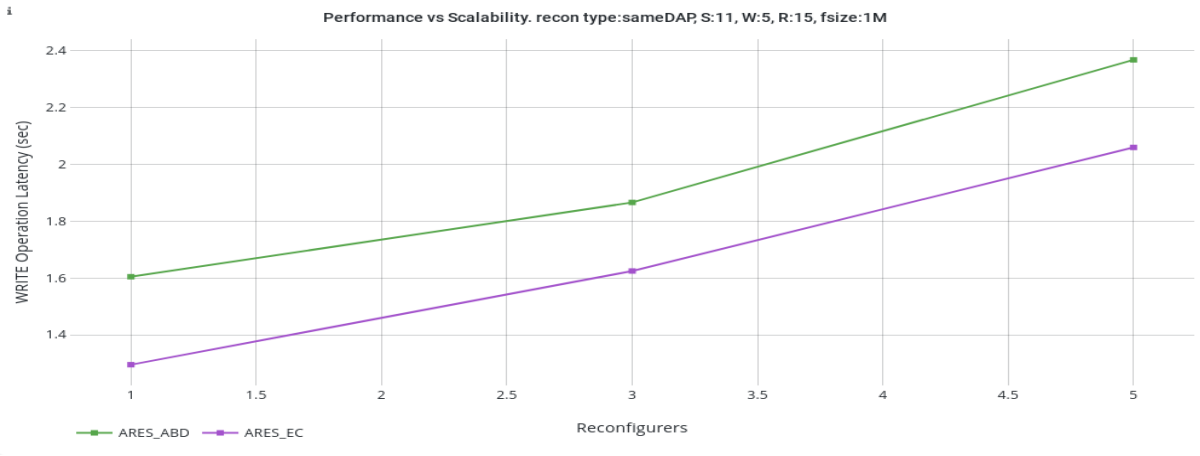


Figure 30: Write when Reconfiguring to the same DAP (RAFT on RPI).

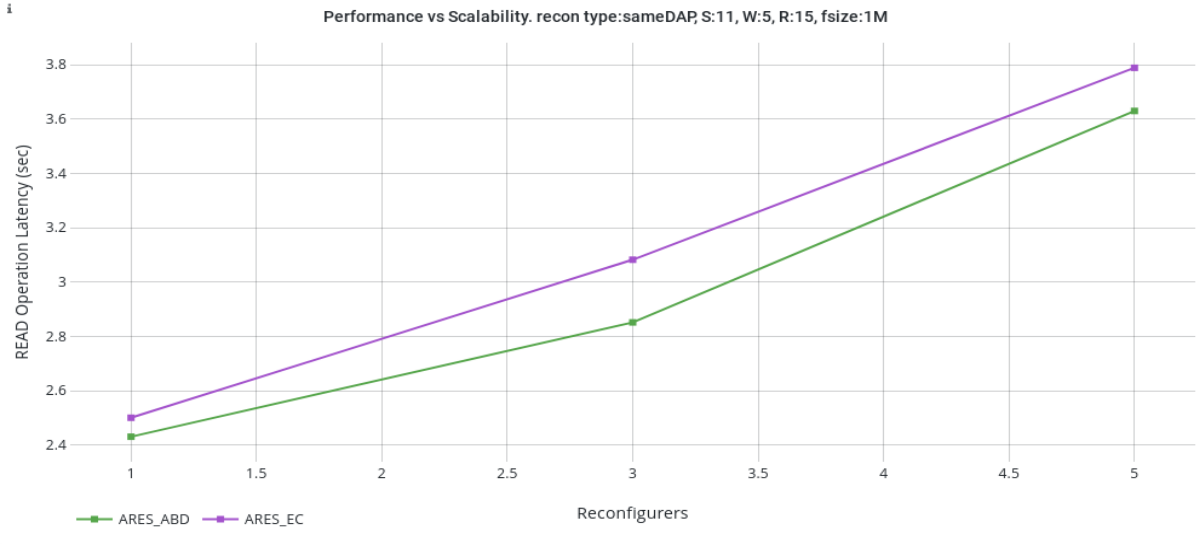


Figure 31: Read when Reconfiguring to the same DAP (RAFT on RPI).

Co-funded by the Horizon 2020
Framework Programme of the European Union

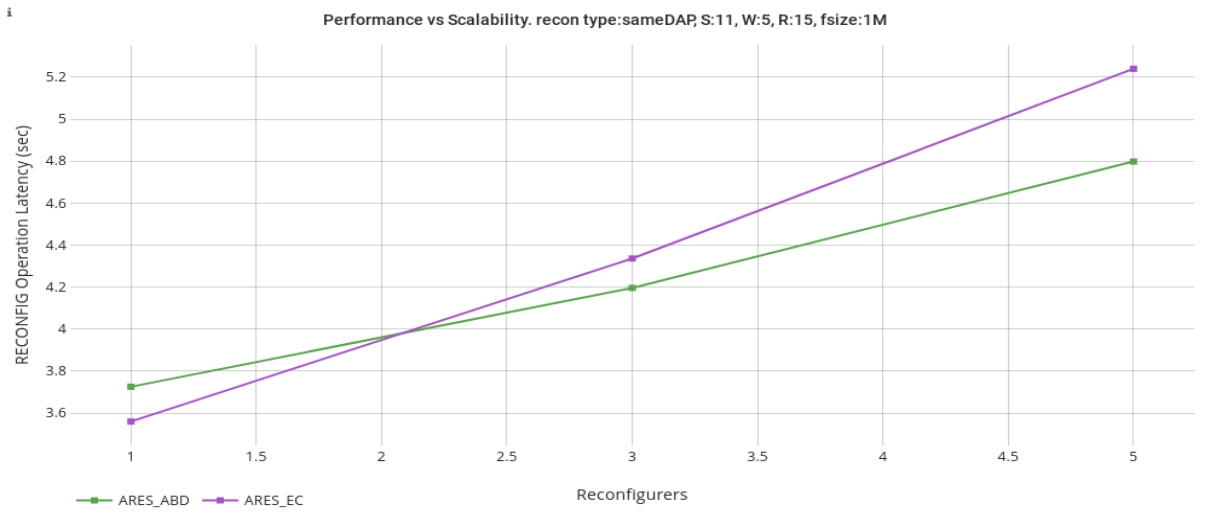


Figure 32: Reconfig when Reconfiguring to the same DAP (RAFT on RPI).

Figure 33: Reconfiguring DAP Alternately and Servers Randomly (RAFT on Emulab).

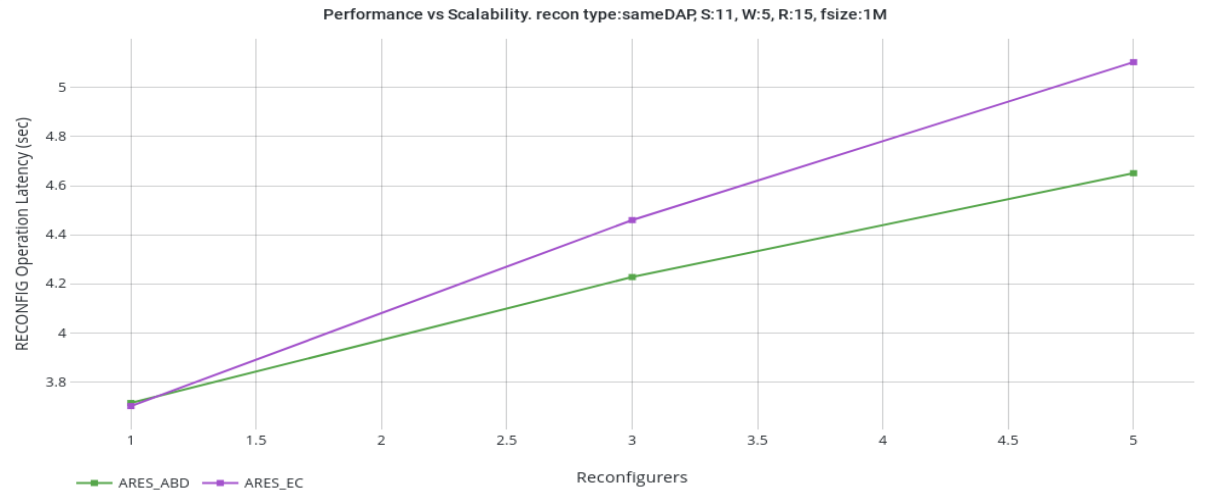


Figure 34: Reconfig when Reconfiguring to the same DAP (RAFT on Emulab).

Longevity Results: Firstly, we discuss the results when the RAFT service was run on RPI devices. Figure 29 illustrates the results of ARES when the reconfigurers switch between DAPs and change the number of servers randomly. It is possible for a read or write operation to

Co-funded by the Horizon 2020 Framework Programme of the European Union

access configurations that implement both the ARES_ABD and ARES_EC algorithms, when the operation is concurrent with recon operations. As expected, the write operation has the lowest latencies compared to the latencies of the other two operations, read and reconfig. Since a reconfiguration involves more communication rounds, it was expected to experience higher latency than the other operations. However, all the latencies increase as the number of reconfigurers increases, since operations need to access more configurations before completing. In Figure 30 to Figure 32, we can see the results when the reconfigurers change to the same DAP. Our choice of k minimizes the coded fragment size but introduces bigger quorums and thus larger communication overheads. As a result, in smaller file sizes, the ARES algorithm may not benefit so much from the erasure coding, resulting in longer latencies for its read and reconfig operations when using the ARES_EC. Figure 33 and Figure **34** illustrate results for the same experiments while the RAFT service was deployed on Emulab machines. We observe that the latencies of the operations have the same behaviour as when the RAFT was deployed on RPI devices. Thus, manipulating the location of the RAFT service does not dramatically impact the latencies of the operations.

6 Present and Foreseen TRL

The successful implementation of the proof of concept along with the experimental validation allowed this project to reach a **Technology Readiness Level 4** (TRL4). Algolysis will now be able to take an informed and data-backed decision on whether the technology is technically feasible and engage in a longer-term targeted investment. In case the technical and market feasibility is viable, Algolysis will integrate the developed technology into the business strategy of the company and allocate resources for further development attempting to achieve a **Technology Readiness Level 5-9** (TRL5-9). Ultimately, Algolysis envisions to bring the technology to the market within a larger Memory-as-a-Service framework which will provide the basis for advanced, robust, and dependable distributed applications. With the advent of powerful mobile devices, the need for efficient distributed applications is imminent and the time is right for a tool like ARES to penetrate the global market.

The Academic partner will have the chance to identify weaknesses of the current technology and explore more efficient solutions, advancing the field into new levels. This may lead to important scientific publications that will be published in prestige conference and journal venues, generating a direct impact on the research community.

7 Exploitation, Dissemination and Communication Status

Dissemination and Communication Plan: In this project we committed to a number of dissemination activities, proportional to the duration of the project, in order to bring the project and its experimentation results to the attention of peers, potential industry partners and investors, as well as the wider public in EU, the US, and elsewhere. Bellow we provide additional details on the status of the various activities:

- **Project Website:** A project website was created in order to outline the objectives of the project. We use the website to announce and record progress and interesting outcomes from our experiments. The website hosts descriptions and visualizations of our experimental outcomes. Ultimately, the website aims to expose the project to the general public and stakeholders that may have an interest in the developed technology. Project Website: https://projects.algolysis.com/ares-ngi/

Co-funded by the Horizon 2020 Framework Programme of the European Union

Project Reference Website: https://www.algolysis.com/projects/ngiatlantic-project/

- **Social Media:** To bring the project to the attention of a larger international community we posted in the partner profiles' in popular social media platforms like LinkedIn, Facebook, Twitter, etc.
- **Technical Publications:** The current deliverable contains all the details of the implementation of the project and will be kept internally as a technical report for future reference. Two scientific publications are expected to emerge from our work in the project:
  (1) A work which carried out in the company and was investigating the use of Shared Memory technologies (like ARES) in distributed VR applications was published in the **ApPLIED 2022** workshop that was held in conjunction with the Symposium on Principles of Distributed Computing (PODC). Parts of the implementation developed in this project were used in that work to establish the underlying communication between the participating nodes.
  (2) We plan to consolidate the final results of the project and submit a manuscript for publication in the International Symposium on Stabilization, Safety, and Security of Distributed Systems (**SSS 2022)**. The manuscript will present the project outcomes and the comparison of the performance of the ARES protocol with Cassandra and Redis (the two commercial solutions we considered in our work).
  Both venues bring together designers and practitioners of distributed systems from both academia and industry to share their point of view and experiences, so this is a targeted audience and ideal to expose the results of the project.
- **Presentations:** Members of the consortium participated and presented the work of the project in the **IoTWeek 2022** conference held in Dublin, Ireland. Apart from the presentation, the researchers had the chance to network with other participants and expose the activities and the outcomes of the project to external stakeholders.

Exploitation Plan: The successful implementation of the proof of concept and the experimental validation (TRL4) allows the partners to plan beyond this project in both the technical and business fronts. The results of the project proved that the technology is promising (with respect to existing solutions) and may be exploited further for reaching higher TRL levels and eventually made available in the global market.

Together Algolysis and the partners in PSU, could co-exploit such an ambitious service and offer it through connections of the two organisations both in the EU and the US. On one hand this will provide a great competitive advantage to Algolysis, and on the other hand bringing research results into practical applications will improve the standing of the PSU and the two researchers in the impactful index.

We aim to promote the technology mainly to (i) researchers of the distributed computing community; (ii) developers of distributed applications that may leverage the technology for simplifying the implementation of data sharing and synchronization modules; and (iii) tech businesses that are interested in utilizing our memory services for the needs of their real-time, data and consistency driven multi-user cloud applications (such as massively multiuser games, worldwide virtual classes, on-demand services, etc.) and emerging technologies (i.e., IoT, VR, AR etc.), both in a cloud and a peer-to-peer setup. Once the technology matures, it will constitute the one-stop-shop for distributed applications that would like to benefit from cloud-RAM like services. We believe that distributed applications will become the norm and future

Co-funded by the Horizon 2020 Framework Programme of the European Union

generations will become accustomed with using cloud storage services not only for specialized applications but even for in-school programming exercises.

The outcomes from our experiments help both partners to showcase the potential of the technology and thus utilize them to secure further funding (either through investment or EU/US projects) to further improve the technology. The EU partner has already applied for a national grant and we plan to apply for Eurostars project, a competitive opportunity for European SMEs. Moreover, the outcomes will help the partners to attract external collaboration proposals from partners that are interested in embedding the technology in their research proposals or future products. The EU partner has already been invited in a Horizon Europe proposal for the application of the technology in remote VR/AR applications.

8 Impacts

Impact 1: Enhanced EU – US cooperation in Next Generation Internet, including policy cooperation.

The teams from the EU and US worked on the investigation of efficient tools that enable performant distributed applications in the next generation of internet. Our outcomes show that we do possess a promising technology that may complement existing solutions and may raise the need for the introduction of new standards in the era of distributed applications. The researchers at Penn State University (USA) collaborated closely with the Algolysis LTD (Cyprus, EU) team, to develop solutions that are being tested in a cross-Atlantic setup. This teamwork laid the foundation for the next generation international network protocols and further strengthened the relationships of EU and US partners towards future collaborations.

Impact 2: Reinforced collaboration and increased synergies between the Next Generation Internet and the US Internet programmes.

The US and EU teams collaboratively devised the scenarios that were executed in this project. Furthermore, ideas were exchanged during the analysis of the results and the identification of shortcomings and performance bottlenecks that provide space for further optimization. On this basis, the partners on this consortium will maintain a close collaboration beyond the end of this project, to enrich their knowledge and improve over the performance we achieved in these experiments.
Furthermore, the two partners will seek new application domains for the developed technologies. The US partner has extensive experience in the design and evaluation of distributed algorithms and erasure coded storages. On the other hand, the EU partner has experience with devising state-of-the-art algorithmic solutions related to distributed systems, mobile & sensor networks, and VR/AR interactive experiences. We aspire to continue integrating and exchanging our knowledge after the end of this project and attempt to utilize the devised technology to the different domains of our expertise, reaching a demonstrator that will allow us to influence technology adoption and policy making.

Impact 3: Developing interoperable solutions and joint demonstrators, contributions to standards.

This collaboration between the EU and US partners yields an ADSS that operate across borders, demonstrating a new protocol for robust, and consistent data sharing. At its heart, ADSS

Co-funded by the Horizon 2020 Framework Programme of the European Union

algorithms (like ARES) promote interoperability, as they expose a simple application interface making them attractive for a large set of Cloud Applications and emerging technologies (i.e., IoT, VR, AR etc.). Our implementation and deployment of ARES is one of the few ADSS implementations that exist at the current stage. Our current deployment is the first practical implementation of a strongly consistent shared storage service to date. Results from our experiments may trigger the composition of new data sharing standards, across countries, applications, and digital assets.

Impact 4: An EU - US ecosystem of top researchers, hi-tech start-ups / SMEs and Internet-related communities collaborating on the evolution of the Internet

The collaboration with PSU is integral for Algolysis (an SME in the South-eastern Med area of the EU), allowing the company's researchers to interact, collaborate, and exchange knowledge with top researchers and engineers in the field. This already provides a competitive advantage to the company and establishes strong collaborative bonds with researchers from one of the leading academic institutions in the world.
On the other hand, researchers in PSU had the chance to collaborate with members from a high-tech startup in the EU which is mostly focused in the production of technologies to be used in the next generation of applications. Therefore they had the chance to experience the process that is followed to bring an idea coming from a basic research to a higher TRL.

We strongly believe that both partners gained from this project and will help them to build a long-lasting collaboration from this point onward.

9 Conclusion and Future Work

As a general finding, achieving strong consistency is more costly than providing weaker semantics as we experienced with Redis and Redis_W. However, the performance gap is not prohibitively large and future optimizations of ARES may close it enough so as to substantiate trading performance for consistency. Compared to the atomic version of Cassandra, ADSM algorithms seem to scale better, but lack behind in the throughput when dealing with small objects. Both approaches seem to be affected by the object size, but ARES_EC suggests that fragmentation may be the solution to this problem. Finally, we demonstrated that ARES may handle efficiently failures in the system, and reconfiguring from one DAP to another without service interruptions. Also, by examining the fragmentation parameter, we exposed trade-offs between operation latency and fault-tolerance in the system: the further increase of the parity (and thus higher fault-tolerance) the larger the latency.

ARES, an algorithm that always offers provable guarantees, competes closely and in many cases outperforms existing DSS solutions (even when offering weaker consistency guarantees). As a future work, it would be of utmost importance to study how optimizations may improve the performance of ARES. For example, fragmentation techniques as presented in [17] may have a positive impact on the performance of the algorithm.

Co-funded by the Horizon 2020 Framework Programme of the European Union

10 References

[1] Nicolas C. Nicolaou, Viveck R. Cadambe, N. Prakash, Kishori M. Konwar, Muriel Médard, Nancy A. Lynch: ***ARES: Adaptive, Reconfigurable, Erasure Coded, Atomic Storage***. ICDCS 2019: 2195-2205.
[2] H. Attiya, A. Bar-Noy, and D. Dolev. ***Sharing memory robustly in message passing systems.*** Journal of the ACM, 42(1):124–142, 1996.
[3] Nancy A. Lynch and Alexander A. Shvartsman. ***Robust emulation of shared memory using dynamic quorum acknowledged broadcasts.*** In Proceedings of Symposium on Fault-Tolerant Computing, pages 272–281, 1997.
[4] **Redis.** https://redis.io/
[5] **ZeroMQ**. https://zeromq.org
[6] **Ansible**. https://www.ansible.com/overview/how-ansible-works/
[7] Avinash Lakshman and Prashant Malik. ***Cassandra: a decentralized structured storage system.*** SIGOPS Oper. Syst. Rev. 44, 2 (April 2010), 35–40. https://doi.org/10.1145/1773912.1773922
[8] Diego Ongaro and John Ousterhout. 2014. ***In search of an understandable consensus algorithm.*** In Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference(USENIX ATC'14). USENIX Association, USA, 305–320.
[9] **RAFT Implementation**. https://raft.github.io
[10] **Emulab**. https://emulab.net
[11] **FED4FIRE**. https://www.fed4fire.eu
[12] **GRID'5000**. https://www.grid5000.fr/w/Grid5000:Home
[13] **Virtual Wall**. https://doc.ilabt.imec.be/ilabt/virtualwall/
[14] **InstaGENI**. https://groups.geni.net/geni/wiki/GeniAggregate
[15] **Cloudlab Utah**. https://www.cloudlab.us
[16] **jFed**. https://jfed.ilabt.imec.be/
[17] Georgiou, C., Nicolaou, N., Trigeorgi, A.: ***Fragmented ARES: Dynamic storage for large objects.*** In Proc. of DISC (2022), to appear. Also at arXiv:2201.13292.

11 Glossary

5G	Fifth Generation (mobile/cellular networks)
NGI	Next Generation Internet
R&D	Research and Development
SDN	Software Defined Networks
TRUST-IT	TRUST-IT (Project Partner)
VNF	Virtual Network Function
WIT	Waterford Institute of Technology (Coordinating Partner)
DSS	Distributed Shared Storage
ADSS	Atomic Distributed Shared Storage
PoC	Proof-of-Concept
HO	Host Organisation
MaaS	Memory As A Service
PC	Project Coordinator

Co-funded by the Horizon 2020 Framework Programme of the European Union

Deliverable 3: Part II

Financial and cost information

This part is to be treated as a consortium confidential deliverable, and access is restricted to consortium partners and EU commission operatives.

Co-funded by the Horizon 2020
Framework Programme of the European Union

12 Workplan Progress and Travel Details

WP 1	Start month	1	Duration	6	Total PM	2
Title	*Project Management*					
Partners involved				Person months		
Partner 1	*Algolysis LTD (ALGO) - LEAD*			1.50 (1.50)		
Partner 2	*Penn State University (PSU)*			0.50 (0.50)		

Goal: The goals of this WP can be summarized in the following points: (i) coordination of the project partners and organization of regular meetings, (ii) reporting to the funding agency (iii) monitoring of project activities and their timely execution, (iv) address any issues that potentially arise during the project, (v) dissemination activities for the project outcomes, and (vi) produce all the necessary progress reports and deliverables.

Activities Description:

Task 1.1 – Coordination and Monitoring Activities [ALGO]: Coordinate the researchers and developers in all partners, to achieve the goals of the project by maintaining a smooth execution of all activities. Monitor the timely completion of all WPs, within budget, and in compliance to the rules and regulations of the Work Programme.

Task 1.2 – Financial and Risk Management [ALGO]: Safeguard the appropriate financial management of the project and time tracking of activities by all participants. Proactively make checks to identify and tackle potential problems that may arise during the execution of the project. Communicate and collaborate with key personnel of all partners to resolve issues. No issues found up to this point.

Task 1.3 – Communication [ALGO]: The PI communicated with the Work Programme personnel to deliver deliverable and seek advice for the smooth execution of the project.

Task 1.4 – Progress Reports [ALGO]: Coordinate with other participants for the preparation of progress reports. A total of two progress reports (including this one) were prepared until now.

Task 1.5 – Meetings [ALGO, PSU]: The project team has decided to meet monthly via online collaboration tools (e.g., Slack, Skype) to discuss updates and plan the tasks ahead as the project progresses. The EU and the US teams had three meetings until now. The European team hold weekly face-to-face meetings, and more than 10 meetings took place until now. Ad hoc meetings will be held when needed to coordinate over technical matters.

Task 1.6 – Project Website [ALGO]: A project website is created in order to outline the objectives of the project and it will be used to record the progress and all the interesting outcomes from our experiments.

Project Website: https://projects.algolysis.com/ares-ngi/

Project Reference Website: https://www.algolysis.com/projects/ngiatlantic-project/

Co-funded by the Horizon 2020 Framework Programme of the European Union

WP 2	Start month	1	Duration	1	Total PM	2
Title	*Design the Experiment*					
Partners involved				Person months		
Partner 1	*Algolysis LTD (ALGO)*			1.00 (1.00)		
Partner 2	*Penn State University (PSU) - LEAD*			1.00 (1.00)		

Goal: The ultimate objective of this WP is to address the design of the experimental scenarios we will run on the various testbeds. In particular the PSU worked closely with ALGO to define: (i) the environmental and traffic parameters in the deployment, (ii) the topology of the deployment, and (iii) the metrics to be collected.

Activities Description:

Task 2.1 – Environmental & Traffic Parameters [PSU, ALGO]: In the very first week we defined the environmental parameters that must take place in the deployment. In particular, we had to define (i) the appropriate number of participants (replica servers and end-users) in order to test the scalability of the service, (ii) the timing of the actions of the end-users on distributed shared memory objects in order to test the performance of the service, and (iii) the number of processors fail-crashes in the system in order to verify the fault-tolerance guarantees and the responsiveness of the service. Notice that, measurements of the performance involve multiple execution scenarios. Each scenario is dedicated in investigating the behaviour of the system affected by a particular system parameter. Thus, to better assess the overall performance of the service, it was agreed that each scenario will be evaluated against various combinations of system participants (i.e., increasing/decreasing the number of readers/writers and servers in the system, operation frequencies, number of failures etc.).

Task 2.2 – Topology [PSU, ALGO]: As per the nature of our experiments it was decided exactly which infrastructures will be utilized and how many resources will be allocated from each infrastructure (based on the various geo-locations, machine capabilities and network limitations) for our needs. We wanted the Proof-of-Concept implementations to be deployed and function in near real-world conditions while utilizing fully reconfigurable infrastructures.

Task 2.3 – Metrics [PSU, ALGO]: We had to investigate which metrics are most suitable to be considered to evaluate the performance of the various algorithms based on our KPIs. It was agreed that the efficiency of the algorithms will be assessed using the following metrics: (i) read and write operation latency including both communication delays and local computation time, (ii) reconfiguration latency for all the dynamic algorithms, and (iii) an empirical verification of the responsiveness and consistency of the ADSS.

Co-funded by the Horizon 2020 Framework Programme of the European Union

WP 3	Start month	2	Duration	5	Total PM	3.75
Title	*Develop and Deploy the Protocol*					
Partners involved				Person months		
Partner 1	*Algolysis LTD (ALGO) – LEAD*			3.25 (3.25)		
Partner 2	*Penn State University (PSU)*			0.50 (0.50)		

Goal: The goal of this WP was to develop and deploy all the protocols on the various testbeds based on the design practises specified and agreed during WP2. Additional details for each task completed are given bellow.

Activities Description:

Task 3.1 – Protocols Development [ALGO]: We had to start implementing one by one the various protocols that will be tested in the experiments. Up to the first half of the project we were able to implement ABD, ARES, and Cassandra. By the end of the project protocols ABD, ARES_EC, ARES_ABD, Cassandra and REDIS were fully developed and deployed.

Task 3.2 – Local Testing [ALGO]: It was performed in order to verify the proper functionality of the various algorithms that resulted from task 3.1 before live deployment.

Task 3.3 – Deployment Methodologies [ALGO]: We investigated and designed various methodologies and implemented various scripts to achieve an easy, fast, reliable and painless deployment procedure of the experiments on the network testbeds both in Europe as well as in US. We were able to fully deploy our protocols on four different testbeds (in the EU and the USA), that are supported by JFed: (i) imec Virtual Wall 1/2 (Belgium – EU), (ii) Cloudlab (Utah – USA), (iii) InstaGENI (NYU, UCLA, and Utdallas – USA) and (iv) Grid5000 [15] (France – EU). In total, we used 39 nodes, where the InstaGENI ones are XEN VMs with Ubuntu 18.04.1 LTS and routable IPs, and the rest are physical machines with Ubuntu 20.04 LTS. Additionally, we deployed our testing code on Emulab testbed, and the RAFT consensus algorithm on RPI and Emulab devices.

Task 3.4 – Live Testing [ALGO]: No work had been done in the first half of the project for this task. Experimentation on all the real deployments started once Task 3.3 was successfully completed early in the second half of the project. During this task we performed initial evaluations of our deployment and experiment execution methodologies.

Co-funded by the Horizon 2020 Framework Programme of the European Union

WP 4	Start month	4	Duration	6	Total PM	6
Title	*Experimentation and Data Collection*					
Partners involved				Person months		
Partner 1	*Algolysis LTD (ALGO) – LEAD*			5.00 (5.00)		
Partner 2	*Penn State University (PSU)*			1.00 (1.00)		

Goal: This WP involves running the experiments and collecting the generated data. The collected data were analysed based on the metrics specified in WP2 and the outcomes were presented in proper visualization formats. Additional details for each task completed are given bellow.

Activities Description:

Task 4.1 – Results collection and analysis from Emulab [ALGO]: Results were collected from Emulab testbed and were strictly analysed in order to verify the correctness of their structure and content.

Task 4.2 – Run the experimental scenarios [ALGO]: We performed six types of scenarios. The *scalability* scenario was constructed to compare the read and write latency while the number of readers, writers, or servers increases. We then ran three stress test scenarios: (i) topology, (ii) objectsize, and (iii) fragmentation parameter k. The *topology* scenario examined the throughput of the algorithms under different topologies. The *objectsize* scenario was used to evaluate how the read and write latencies are affected by the size of the object. In the *fragmentation parameter k* scenario, we examined the read and write latencies with a different number of k (a parameter of Reed-Solomon). In the *longevity* scenario, we evaluated the read, write, and reconfig latencies when increasing the number of reconfigurers while also changing the configuration. Finally, we run a *fault-tolerance* scenario to verify the service's fault-tolerance guarantees.

Task 4.3 – Data Collection and Import [ALGO]: We used python-json-logger library to configure our implementations to log DEBUG and higher-level messages from clients to .log files on disk. In each experimental round a playbook was dedicated to fetch the log files. We devised a parser written in Python that gets all the necessary info from the log files and parse them into the InfluxDB (an open-source time series database).

Task 4.4 – Data Visualization and Analysis [ALGO]: To plot graphs we used Grafana, an open-source visualization platform that lets you visualize metrics data in a variety of graphs and charts (time series, bar chart, histogram, etc). We imported the InfluxDB data to Grafana to be able to monitor the metrics in it. We created visualizations such as plotly, time series, and bar chart using queries and transformations.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Travel Expenses

Trip	1		Total Cost	€3502,20
Start date		19/06/2022	End Date	23/06/2022
From			To	
Larnaca, CY			*Dublin, IR*	
Costs	**Flights**		**Accommodation/ Subsistence**	**Other**
Nicolas Nicolaou	€356,85		Accommodation: €438 Per Diem: 5x€100 = €500	Registration: €337,50
Efstathios Stavrakis	€356,85		Accommodation: €438 Per Diem: 5x€100 = €500	Registration: €575
Participation and presentation of the Experiment results in IoTWeek 2022 in Dublin Ireland. Dr. Nicolaou and Dr. Stavrakis participated in IoTWeek 2022 for the dissemination of the outcomes and the results of the Experiment and the NGI Atlantic project. The visit to the conference allowed the two researchers to meet with other research groups and EU officials for exposing the current project and also for identifying future opportunities for exploiting the results of this project and participate in research consortia. Last, but not least, Dr. Nicolaou and Dr. Stavrakis had the chance to meet with policy makers and express their opinion for the establishment of future calls for EU and US collaboration, similar to the NGI Atlantic initiative.				
Trip	**2**		**Total Cost**	€1307,93
Start date		23/07/2022	**End Date**	26/07/2022
From			**To**	
Larnaca, CY			*Salerno, IT*	
Costs	**Flights**		**Accommodation/ Subsistence**	**Other**
Theophanis Hadjistasi	€337,93		4 x €200 = €800	Registration: €170
Participate and Present relevant work at Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED 2022) conference. Dr. Hadjistasi attended ApPPLIED 2022 Worksop, which was held in conjuction with PODC2022, the most prestigious conference in the area of distributed computing, to present a peer reviewed work related to the topic of this project. In particular, the work presented experimental application of the Distributed Shared Memory (DSM) technology in Network Virtual Environments. DSM is the core technology we explore in this NGI Atlantic project (# OC04-347) and for which we developed a number of components within the project's duration. For the proper implementation of the DSM the work presented in the paper used ideas and components that we have developed during this project. So essentially the ApPLIED paper demonstrated the use of the technology we develop in our experiments in the interesting and high impact domain of NVE and VR environments.				

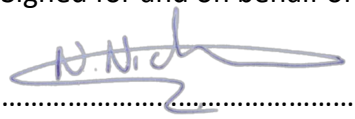
Co-funded by the Horizon 2020 Framework Programme of the European Union

13 Funds Utilisation Report

Cost Title	Amount	Description
Personnel cost(s)	€49350	WP1: Nicolas Nicolaou 1 PM @ 5600 = €5600 Efstathios Stavrakis 0.5 PM @ 5600 = €2800 WP2: Nicolas Nicolaou 0.5 PM @ 5600 = €2800 Efstathios Stavrakis 0.5 PM @ 5600 = €2800 WP3: Nicolas Nicolaou 1.05 PM @ 5600 = €5880 Efstathios Stavrakis 0.35 PM @ 5600 = €1960 Theophanis Hadjistasi 1.1 PM @ 4200 = €4620 Andria Trigeorgi 0.75 PM @ 2800 = €2100 WP4: Nicolas Nicolaou 0.6 PM @ 5600 = €3360 Efstathios Stavrakis 1 PM @ 5600 = €5600 Theophanis Hadjistasi 1.65 PM @ 4200 = €6930 Andria Trigeorgi 1.75 PM @ 2800 = €4900
Other Direct Costs – Travel only	€4810.13	
Total Direct Costs	€54160.13	
Indirect Costs	€13540.03	
Total Costs	**€67700.16**	
Received Amount	€27500	
Remaining Amount	**€40200.16**	

On behalf of Algolysis Ltd, I, Nicolas Nicolaou confirm that this funds utilisation report is in accordance with the contract already in place between Algolysis Ltd and Waterford Institute of Technology under financial support to third parties from Article 15 of Grant Agreement number 871582 — NGIatlantic.eu. I confirm that this report also includes all the expenditures (limited to PM and travel) incurred by all partners in this project and adhere to all instructions contained in H2020 Annotated Model Grant Agreement[1]. These are referenced in section 3 and 5 of the contract. I also confirm that any applicable VAT or tax payments on the amount due to the Grant Recipient shall be fully borne by the Grant Recipient.

Signed for and on behalf of **Algolysis Ltd**



……

Full Name: **Nicolas Nicolaou**
Title: Director
Complete Address: Ellados 12A, Erimi, 4630, Limassol, Cyprus

1 http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/amga/h2020-amga_en.pdf

Co-funded by the Horizon 2020 Framework Programme of the European Union