ARES II: Tracing the Flaws of a (Storage) God

Authors: Chryssis Georgiou[1], Nicolas Nicolaou[2], **Andria Trigeorgi**[1,2]

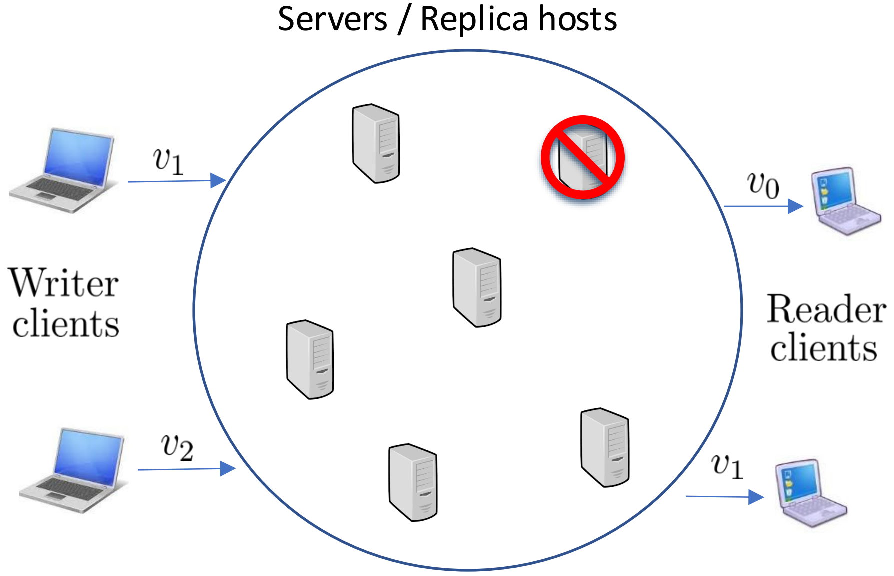
[1]University of Cyprus, Nicosia, Cyprus

[2]Algolysis, Limassol, Cyprus

SRDS 2024, Charlotte, USA

Funded by: PHD IN INDUSTRY/1222/0121 and DUAL USE/0922/0048

Distributed Shared Memory Emulations (DSMs)



Shared read/write object

- A set of servers (configuration) maintain replicas of the same data object.
- Clients (readers/writers) access the object by sending messages to these servers.
- Read/Write operations are structured in terms of phases.
- Each phase consists of two communication exchanges (broadcast & convergecast).
- Fixed Configuration -> Static environment, Reconfiguration -> Dynamic environment
- Consistency guarantees
  - Safety, Regularity, **Atomicity** (Atomic DSMs) [Lamport 1986]

L. Lamport,"On Interprocess Communication," Distributed Computing, vol. 1, no. 2, pp. 77–101, 1986.

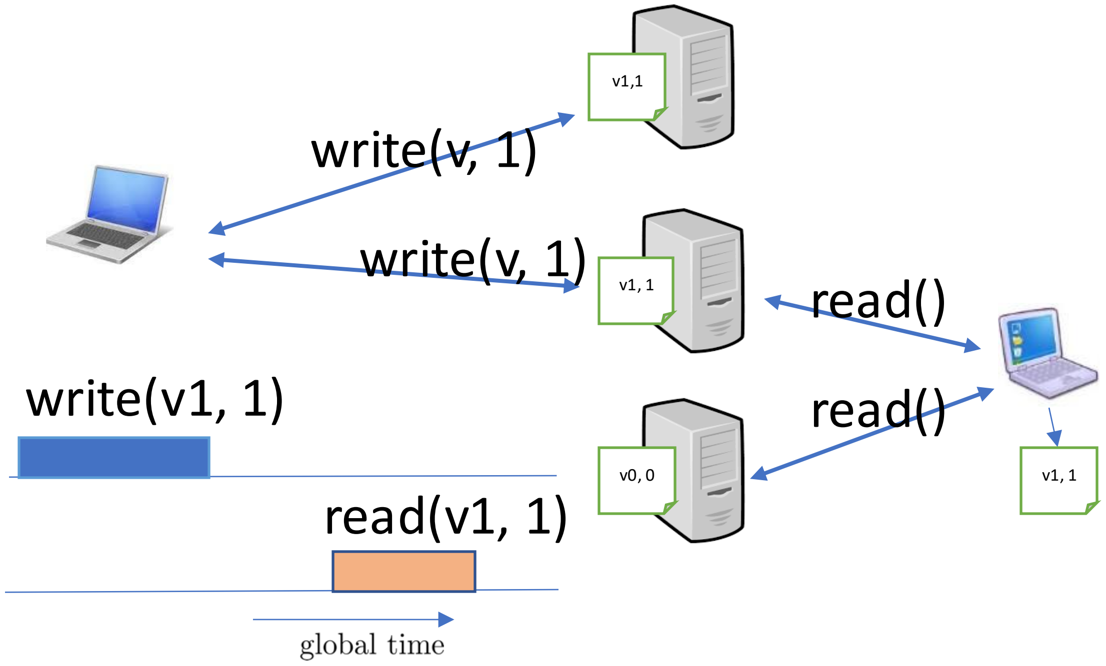
Seminal Algorithm - ABD

An elegant, intuitive solution that

- uses the power of the majority, and
- assigns logical timestamps to written values for ordering the operations.

[Attiya, Bar-Noy, Dolev 1995]
Dijkstra Prize 2011

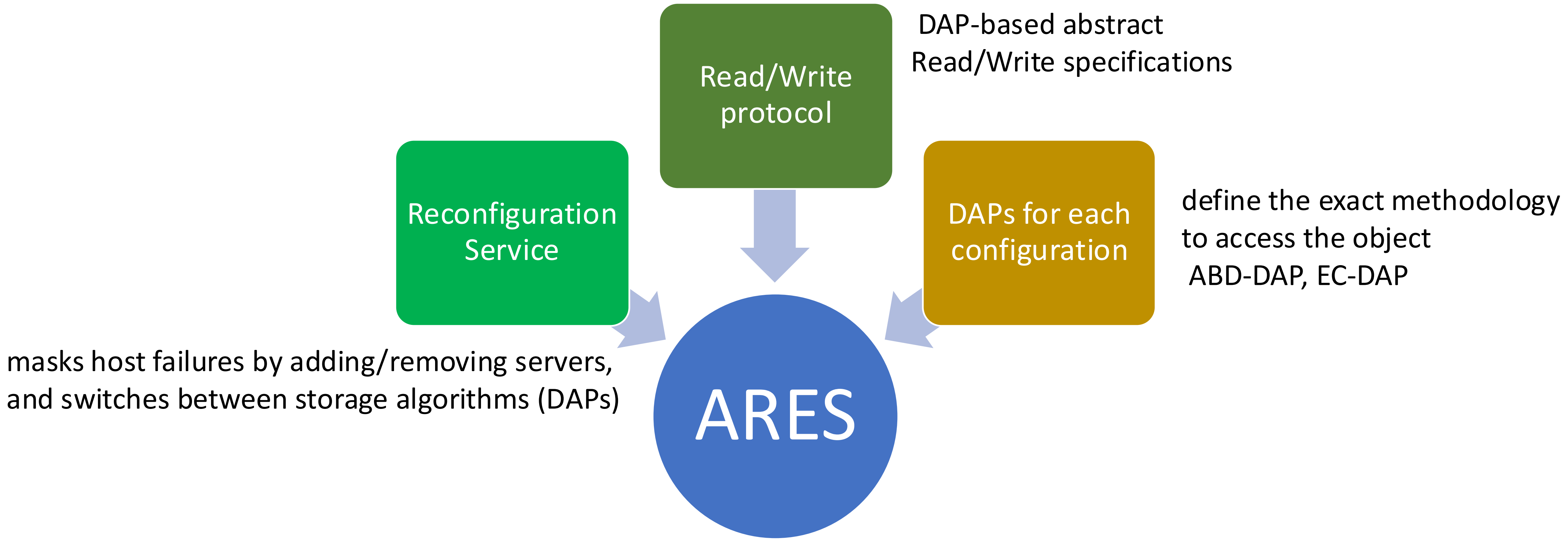
- SWMR atomic registers
- S servers, f < n / 2
- 1 writer
- R readers



- Extended by Lynch and Schwarzmann in 1997 for MWMR, assigning tags <ts,wid> – **MW-ABD**
- Many more complex ABD-like protocols were developed over the years to address various challenges such as fault-tolerance, efficiency, and scalability.

H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing Memory Robustly in Message-Passing Systems," Journal of the ACM (JACM), vol. 42, no. 1, pp. 124–142, 1995.
N. Lynch, A. Shvartsman.. "Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts," In Proc. of FTCS pp. 272–281 (1997).

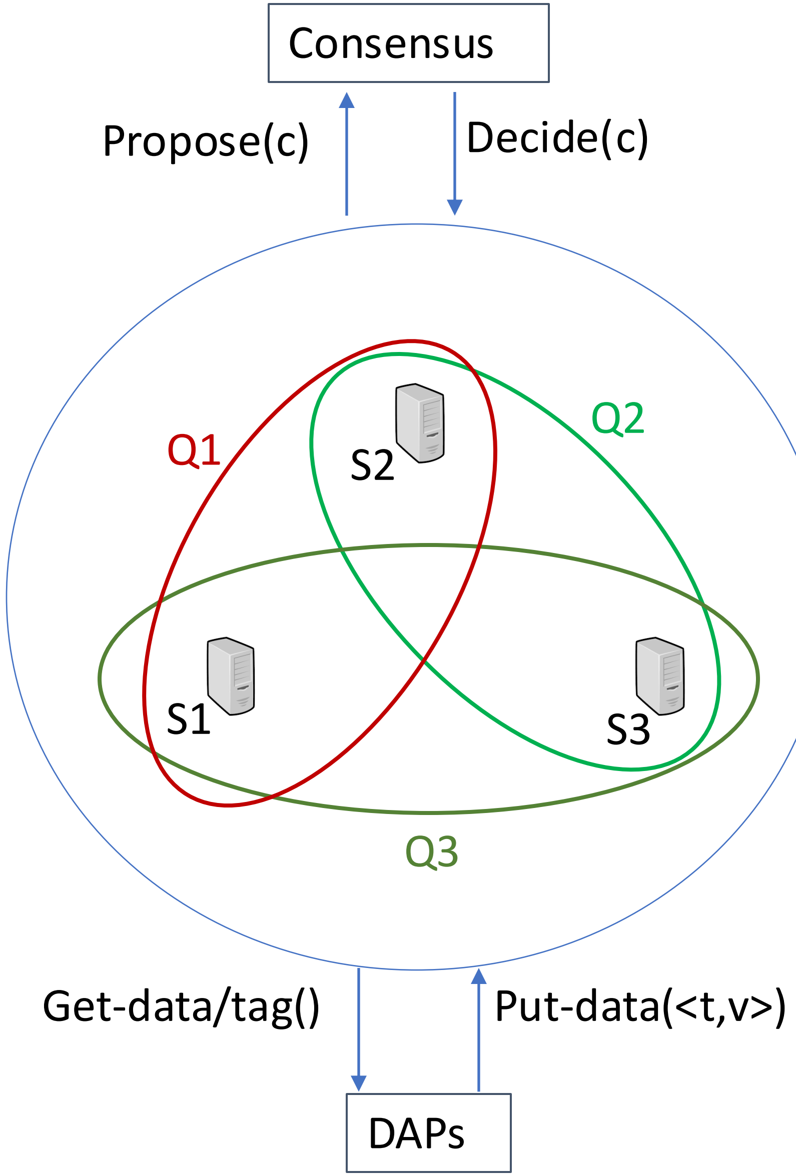
ARES - Adaptive, Reconfigurable, Erasure Code, Atomic Storage



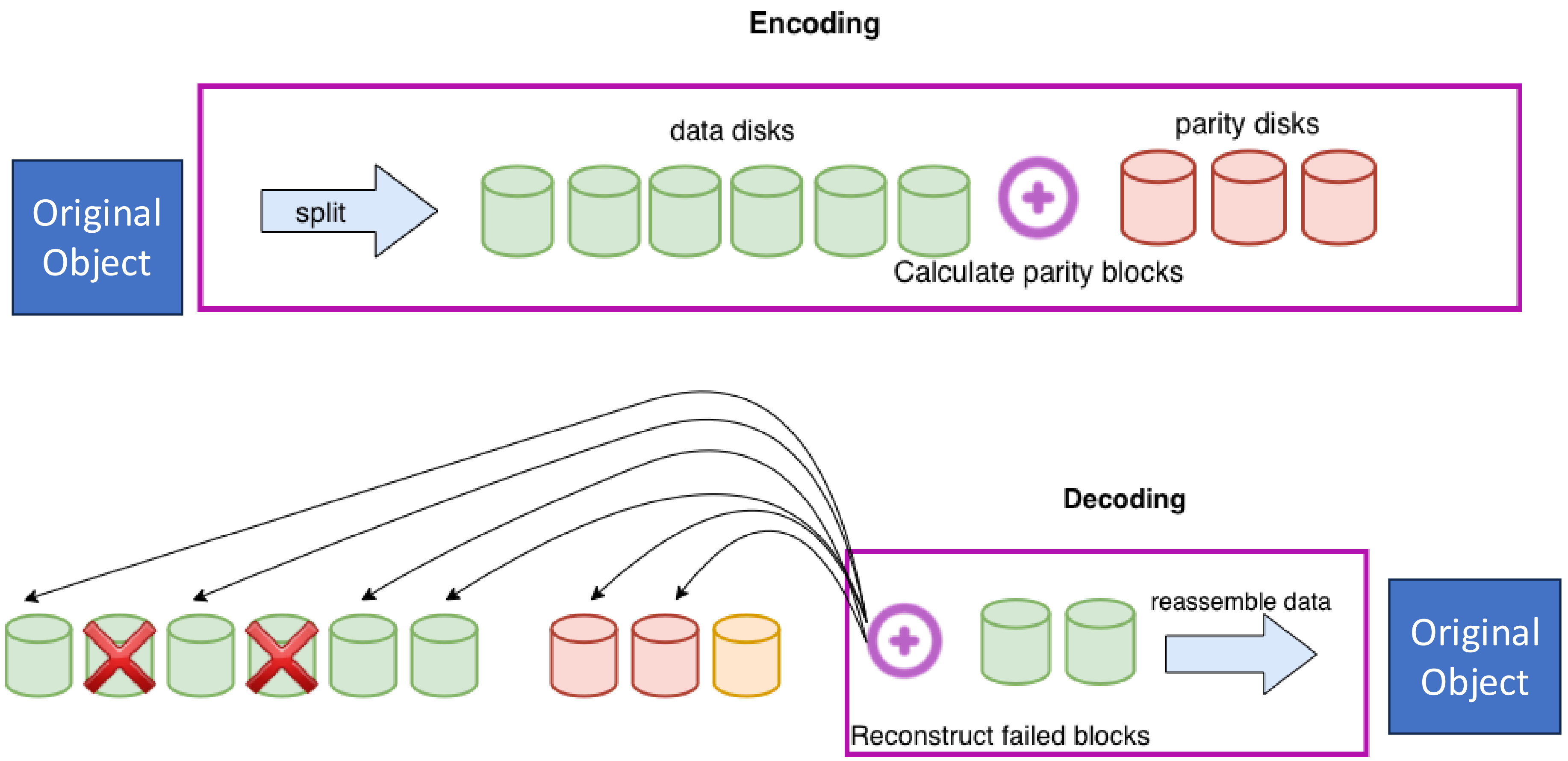
N. Nicolaou, V. Cadambe, N. Prakash, A. Trigeorgi, K. M. Konwar, M. Medard, and N. Lynch, “Ares: Adaptive, reconfigurable, erasure coded, atomic storage,” ACM Trans. Storage, jan 2022. Just Accepted.

Configurations

- A configuration c is characterized by:
  - A unique identifier
  - A set of servers
  - A quorum set system on servers
  - A consensus instanse
  - A DAP implementation
    - D1. $c.get-tag(\,): returns\ a\ tag\ \tau\ \in\ \mathrm{T}$
    - D2. $c.get-data(\,): returns\ a\ tag-value$ $pair\ (\tau, v) \in \mathrm{T}\ \times\ V$
    - D3. $c.put-data(<\tau, v>)$: $\text{the tag} - \text{value}$ $\text{pair}(\tau, v)\ \in \mathrm{T}\ \times\ V$ as argument
    - ABD-DAP & EC-DAP



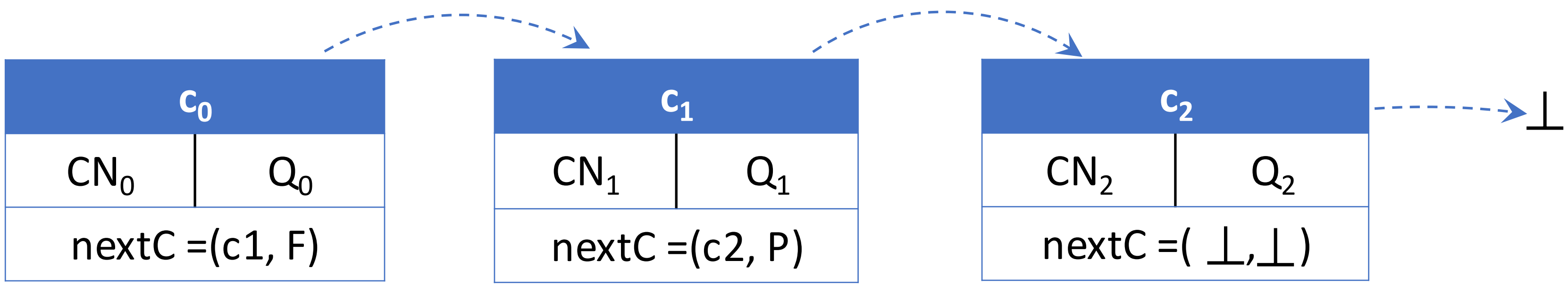
Erasure Code (EC)



(n, k)-Reed-Solomon code: n=servers, k=data servers, m=parity servers
BUT reads and writes are still applied on the entire object

Configuration Sequence

- Global configuration sequence G_L
- nextC: each server points to the next configuration
  - Same nextC to all servers of a single config c (due to consensus)
- Flags {P, F}: pending, finalized
  - Pending: not yet a quorum of servers received msgs
  - Finalized: new configuration propagated to a quorum of servers



Reconfiguration Service

- A reconfig operation performs 2 major steps:
  1) Configuration *Sequence Traversal*
  2) Configuration *Installation*
     - Transfers the object state from the old to the new configuration

```
operation reconfig(c)
    if c ≠ ⊥ then
        cseq ←read-config(cseq)
        cseq ← add-config(cseq, c)
        update-config(cseq)
        cseq ← finalize-config(cseq)
end operation
```

(1)
- attempt get to the latest configuration

(2)
- introduce the new configuration
- move the latest value to the new config
- let servers know it is good to be finalized

This service guarantees that if cseq1 and cseq2 are obtained by two clients resp., then either cseq1 is a prefix of cseq2 or vice versa

Read/Write Operations using DAPs

Reader Protocol

- Traverse Config Sequence cseq
- Find μ = max(<c, F>) in cseq
- Set v = last(<c,*>) in cseq
- Discover for μ ≤ i ≤ v
  (t,v)=max(cseq[i].**get-data**())
- Do
  - cseq[v].**put-data**(t,v)
  - Traverse Sequence cseq
- while(|cseq| > v)

Writer Protocol(val) (at w_i)

- Traverse Config Sequence cseq
- Find μ = max(<c, F>) in cseq
- Set v = last(<c,*>) in cseq
- Discover for μ ≤ i ≤ v
  t_{max}=max(cseq[i].**get-tag**())
- (t,v)= (<t_{max}+1,w_i>, val)
- Do
  - cseq[v].**put-data**(t,v)
  - Traverse Sequence cseq
- while(|cseq| > v)

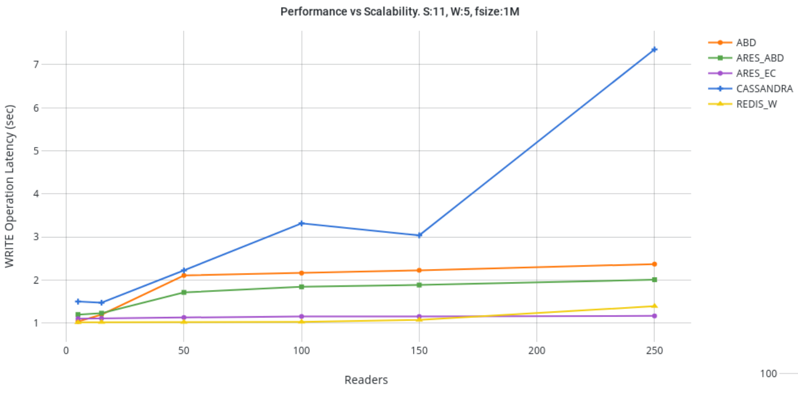
Main Objective

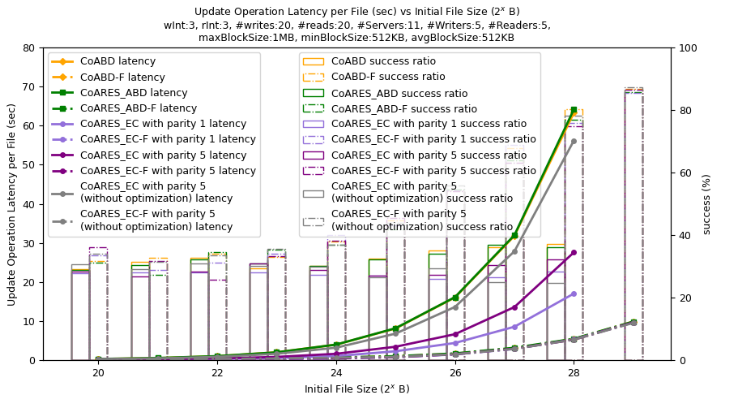
The primary goal is to **identify flaws** in **DSMs** and **guide their optimization.**
We demonstrate this through the ***ARES*** DSM.

Performance Analysis Challenges in DSMs

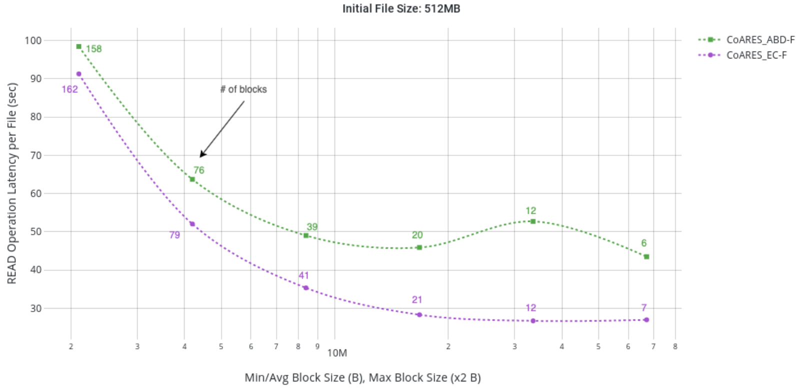
- Identifying performance bottlenecks in complex DSMs can be challenging
- Traditional logging techniques may not provide sufficient insight

```python
import os
import logging
from sys import platform
from pythonjsonlogger import jsonlogger

 atrigeorgi
class SetupLogger:

     atrigeorgi
    def setup_logger(self, logfile, level=logging.DEBUG):
        # Due to race conditions, we sometimes get error.
        # OSError: [Errno 17] File exists: 'log'
```






```python
self.logger.debug('READ-COMPLETE-DSMM',
                  extra={"clientID": self.uid, "objectID": file_id, "tag": maxTag, "value": value})
```

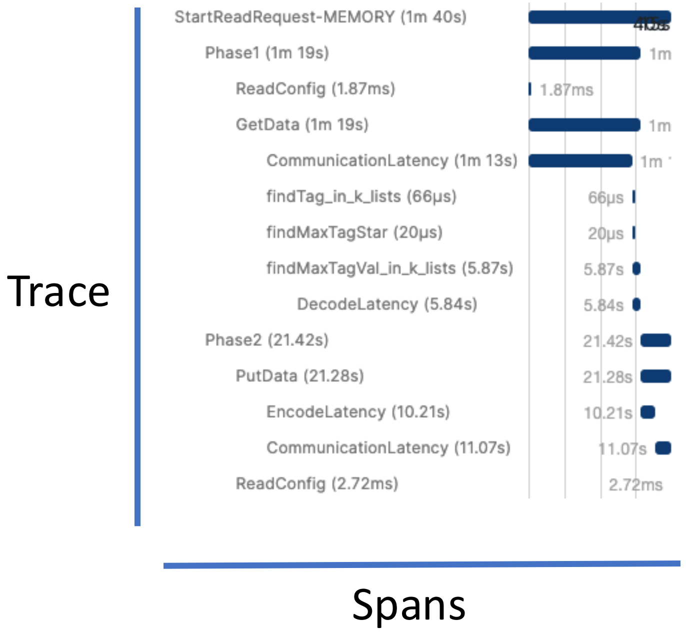


“**Distributing Tracing** is a monitoring technique used to track individual requests as they move across multiple components within a distributed system. It helps to pinpoint where failures occur and what causes poor performance.”

Distributed Tracing – Terminology

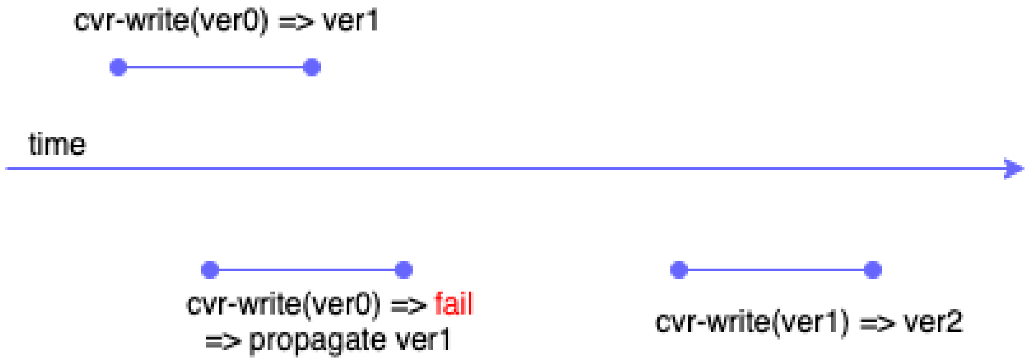
- A **trace** represents the entire journey of a request.
- A **span** represents a unit of work within a trace (e.g., procedures, sections of code).
- Tracings tools: Opentemetry, Zipkin, Jaeger.

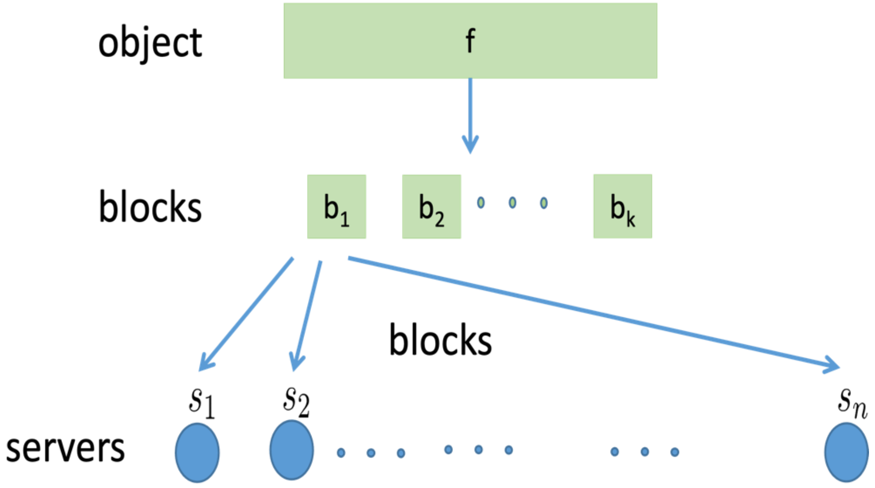
```
with self.tracer.start_as_current_span("StartWriteRequest-MEMORY"+self.phase) as parent_span:
    with self.tracer.start_as_current_span("Phase1"):
        with self.tracer.start_as_current_span("GetTag"):

            # increase msg counter for next op
            self.msg_id += 1

            # Creating 'READ' message with the following fields
            # [type, msgID, clientID, objectID]
            message = {'type': 'READ-TAG', 'msgID': self.msg_id, 'clientID': self.uid, 'objectID': file_id}

            with self.tracer.start_as_current_span("CommunicationLatency"):
                # Broadcast it
                self.broadcastMessage(message)

                # Wait for READ-TAG-ACK messages to come from a majority
                msgs_list = self.waitReply(self.majority, "READ-TAG-ACK")
```



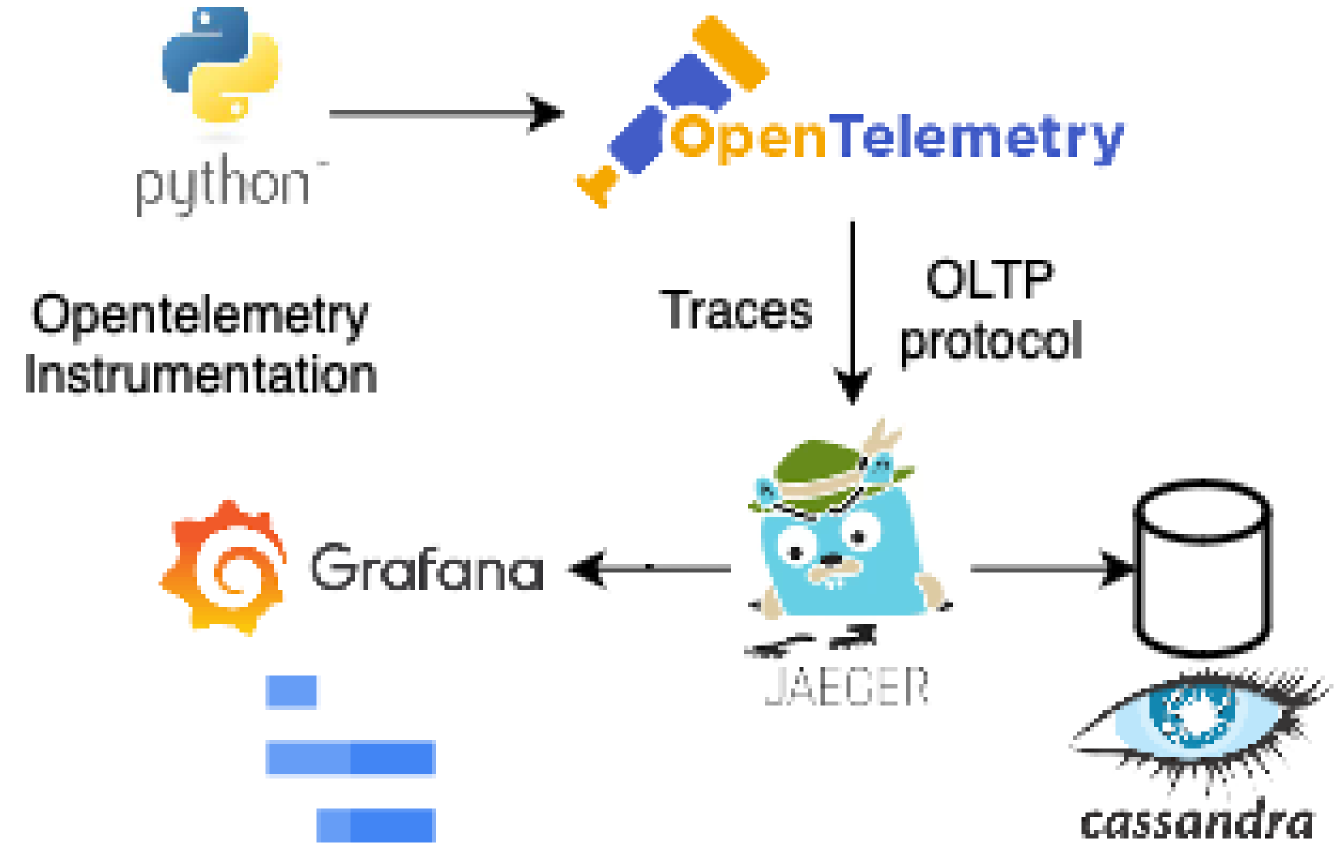
Evaluated Algorithms

ARESABD	This is Ares that uses the ABD-DAP implementation.
CoARESABD	The coverable version of *ARESABD.*
CoARESABDF	The fragmented version of *CoARESABD*.
ARESEC	This is *ARES* that uses the EC-DAP implementation.
CoARESEC	The coverable version of *ARESEC.*
CoARESECF	This is the two-level data striping algorithm obtained when *CoARESF* is used with the EC-DAP implementation; i.e., it is the fragmented version of *CoARESEC*.

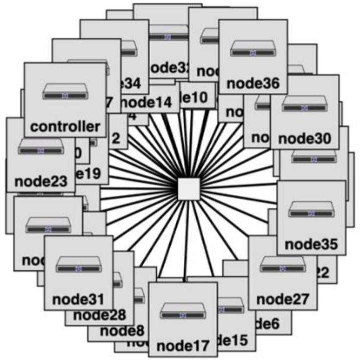




Methodology: ARES Distributed Tracing

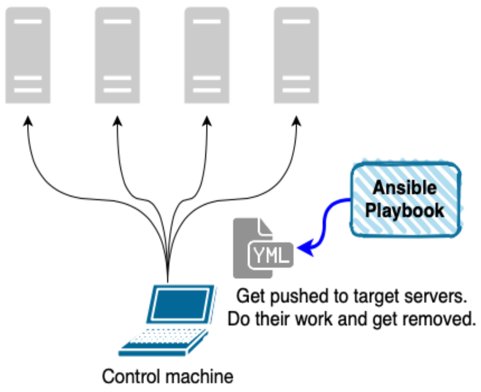


Experimental Setup



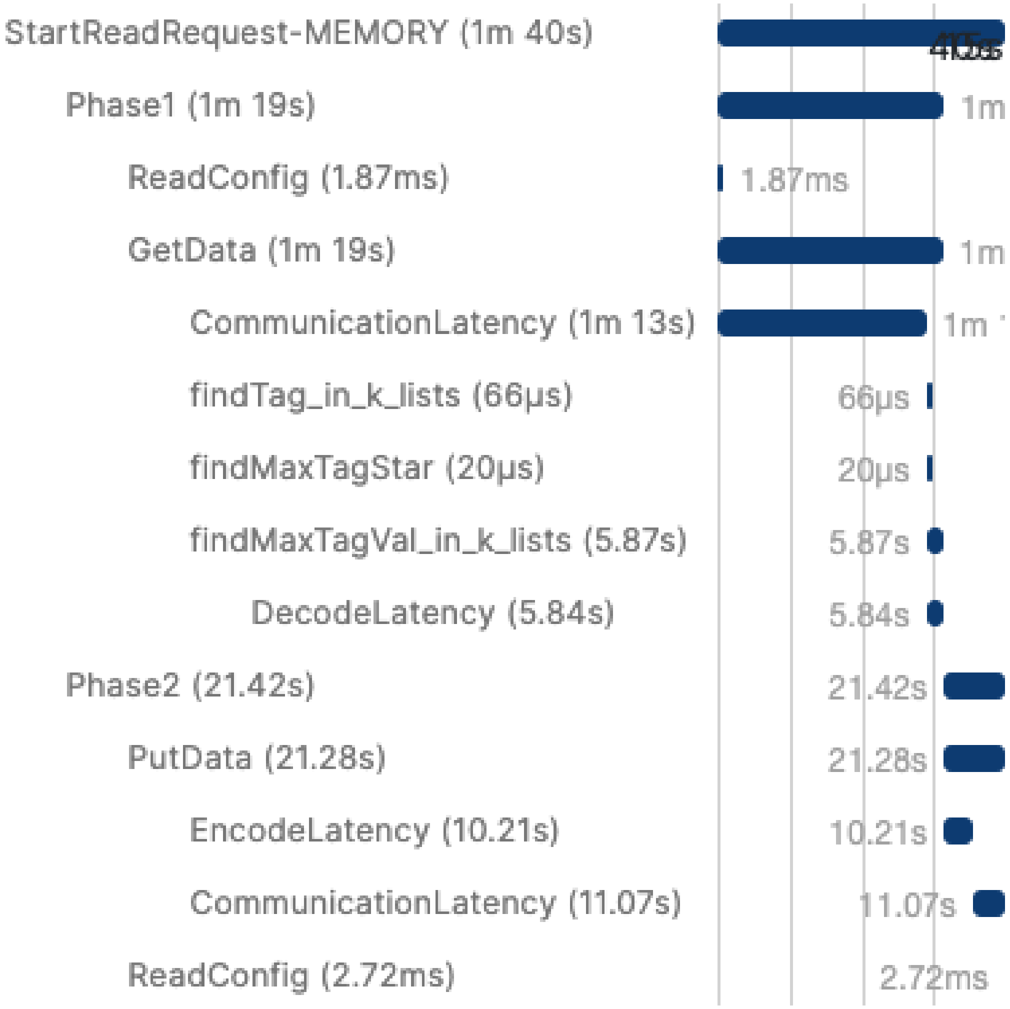
We used two main tools to run the experiments:

- **Emulab:** an emulated WAN environment testbed.
  - 39 machines with 100 Mb/s bandwidth
  - Each server is deployed on a different machine.
  - Clients are all deployed in the remaining machines in a round robin fashion.
- **Ansible:** a tool to automate different IT tasks.

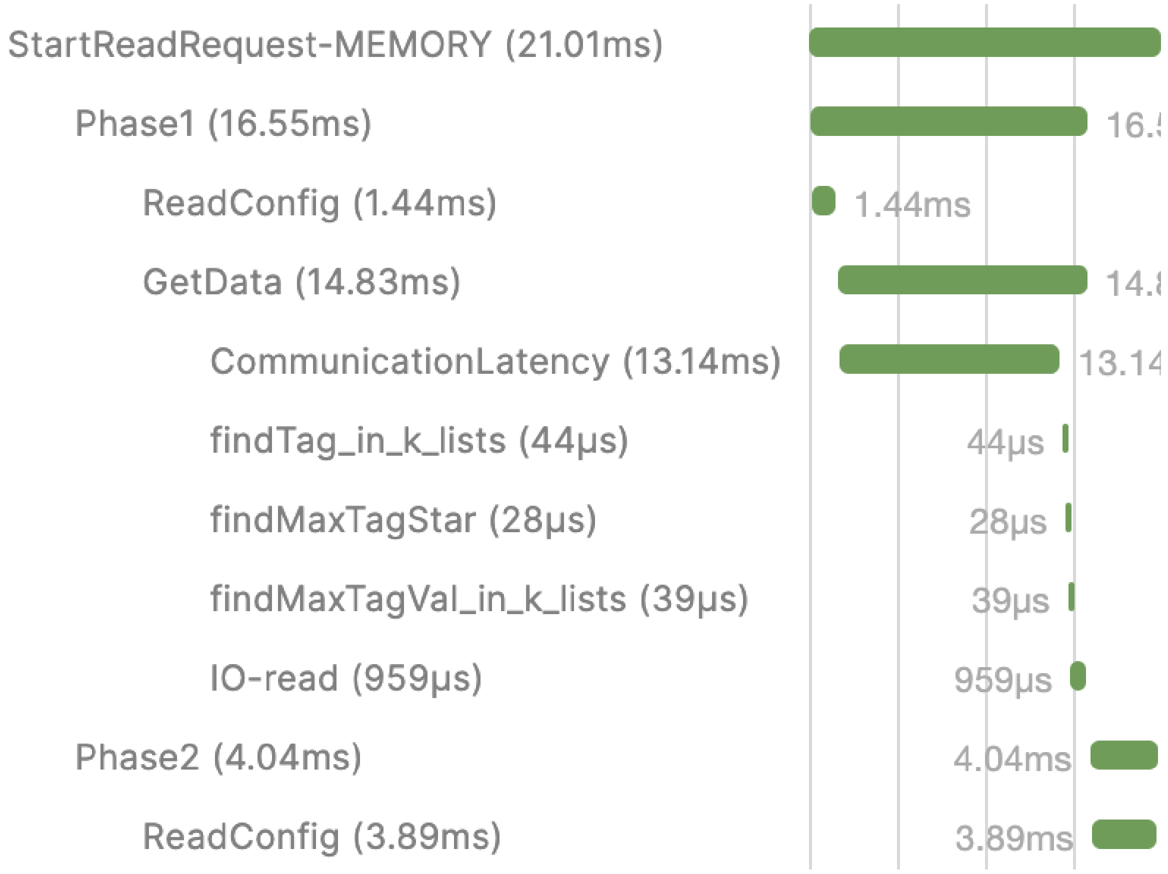


- **Performance Metric**
  - Operation latency of clients (Communication + Computation Overhead).
  - Sample traces near the average duration for each scenario.
  - Three executions.

Object Size

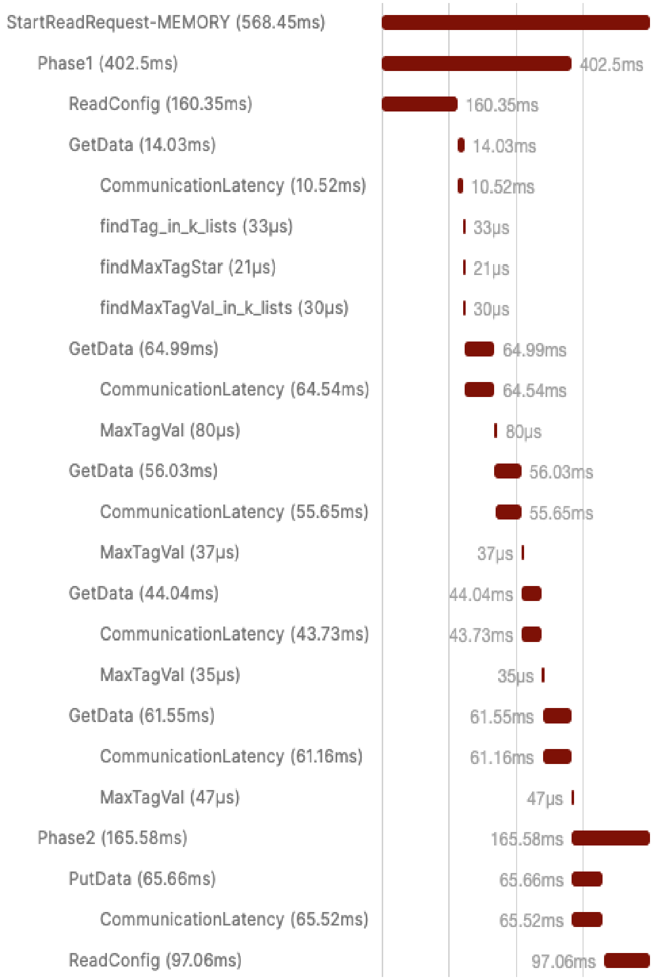


ARESEC, S:11, W:5, R:5, fsize:512MB, Debug Level:DSMM

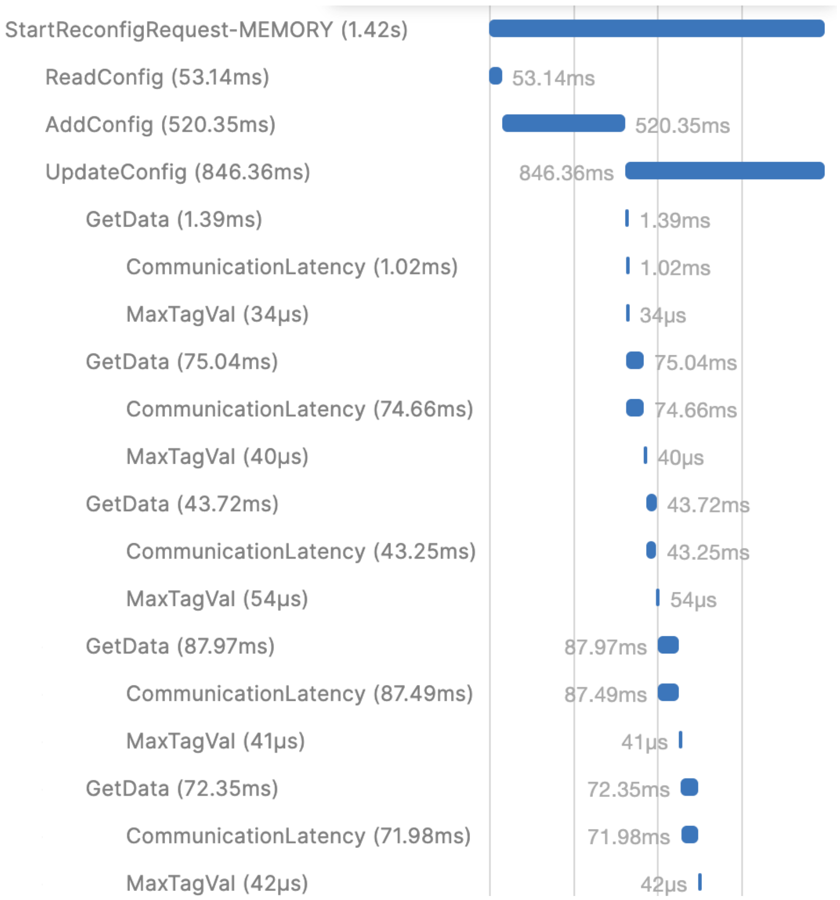


CoARESECF, S:11, W:5, R:5, init fsize:512MB, Debug Level:DSMM

Longevity



CoAresF, S:11, W:5, R:15, G=5, fsize:4MB, Debug Level:DSMM



CoAresF, S:11, W:5, R:15, G=5, fsize:4MB, Debug Level:DSMM

From ARES to ARES II

Optimisation 1: Piggybacking

Main difference: skip read-config for latest config

Query phase: embed latest configs with data

Extra function: discover the next config

operation write(val), $val \in V$
  ~~$cseq \leftarrow$ read-config($cseq$)~~
  $\mu \leftarrow \max(\{i : cseq[i]].status = F\})$
  $cs \leftarrow cseq[\mu]$
  while $cs \neq \bot$ **do**
    $\tau_c, Cs \leftarrow cs.cfg$.get-tag()
    $\tau_{max} \leftarrow \max(\tau_c, \tau_{max})$
    $cs, cseq \leftarrow$ find-next-config($cseq, Cs$)
  end while
  $\langle \tau, v \rangle \leftarrow \langle \langle \tau_{max}.ts + 1, \omega_i \rangle, val \rangle$

  $\lambda \leftarrow \max(\{i : cseq[i] \neq \bot\})$
  $cs \leftarrow cseq[\lambda]$
  while $cs \neq \bot$ **do**
    $Cs \leftarrow cs.cfg$.put-data($\langle \tau, v \rangle$)
    ~~$cseq \leftarrow$ read-config($cseq$)~~
    $cs, cseq \leftarrow$ find-next-config($cseq, Cs$)
  end while
end operation

Optimisation 1: Piggybacking – EC-DAP

Algorithm 3 EC-DAP II implementation

at each process $p_i \in \mathcal{I}$

procedure c.get-tag()
  send (QUERY-TAG) to each $s \in c.Servers$
  until p_i receives $\langle t_s \rangle, nextC_s$ from $\left\lceil \frac{n+k}{2} \right\rceil$ servers in $c.Servers$
  $Cs \leftarrow \{nextC_s :$ received $nextC_s$ from $s\}$
  $t_{max} \leftarrow \max(\{t_s :$ received t_s from $s\})$
  return t_{max}, Cs
end procedure

procedure c.get-data()
  send (QUERY-LIST) to each $s \in c.Servers$
  until p_i receives $List_s, nextC_s$ from $\left\lceil \frac{n+k}{2} \right\rceil$ servers in $c.Servers$
  $Cs \leftarrow \{nextC_s :$ received $nextC_s$ from $s\}$
  $Tags_{dec}^{\geq k}$ = set of tags that appears in k Lists
  $t_{max}^{dec} \leftarrow \max(Tags_{dec}^{\geq k})$
  if $Tags_{dec}^{\geq k} \neq \emptyset$ **then**
    $fragments \leftarrow \{e : \langle \tau, e \rangle \in Lists$ & $\tau = t_{max}^{dec}\}$
    if $\nexists \bot \in fragments$ **then**
      $v \leftarrow$ decode value for t_{max}^{dec}
    else
      $v \leftarrow \bot$
    return $\langle t_{max}^{dec}, v \rangle, Cs$
end procedure

procedure c.put-data($\langle \tau, v \rangle$)
  $code\text{-}elems = [(\tau, e_1), \ldots, (\tau, e_n)]$, $e_i = \Phi_i(v)$
  send (PUT-DATA, $\langle \tau, e_i \rangle$) to each $s_i \in c.Servers$
  until p_i receives $nextC_s$ from each server $s \in \mathcal{S}_g$ s.t. $|\mathcal{S}_g| = \left\lceil \frac{n+k}{2} \right\rceil$ and $\mathcal{S}_g \subset c.Servers$
  $Cs \leftarrow \{nextC_s$: received $nextC_s$ from each $s \in \mathcal{S}_g\}$
  return Cs
end procedure

requests max τ and nextC in one go

returns max τ and servers' nextC

requests data list and nextC

If nextC is finalized, servers sends only the tag and their nextC

requests data update and nextC

returns all servers' nextC

Optimisation 2: Garbage Collection

procedure gc-config($cseq$)
$\mu \leftarrow \max(\{i : cseq[i].status = F\})$ — Last finalized config
$CID \leftarrow \{i : cseq[i] \neq \perp \wedge i < \mu\}$ — older configs from last finalized config
for id in CID **do**
  send (GC-CONFIG, $next$) to each $s \in cseq[id].Servers$ — send GC request to servers
  until $\exists Q, Q \in cseq[id].Quorums$ s.t. rec_i receives ACK from $\forall s \in Q$
  /* remove the {id, cseq[id]} */
  $cseq \leftarrow cseq \setminus \{id, cseq[id]\}$ — Remove the GC configs
return $cseq$
end procedure

EC-DAP

Upon receive (GC-CONFIG, $cfgT_{in}$) $_{s_i, c_k}$ **from** q
  if $cfgT_{in}.cfg.ID > nextC.cfg.ID$ **then**
    $nextC \leftarrow cfgT_{in}$ — server updates nextC to point to the last finalized config
  for τ, e in $List$ **do**
    $List \leftarrow List \setminus \{\langle \tau, e \rangle\}$
    $List \leftarrow List \cup \{\langle \tau, \perp \rangle\}$ — the server sets the data of config to ⊥
  send ACK to q
end receive

Server' Response

Optimisation 3: Batching

```
operation reconfig(c, D)
   if c ≠ ⊥ then
      cseq ← read-config(cseq)
      cseq ← add-config(cseq, c)
      update-config(cseq, D)
      cseq ← finalize-config(cseq)
      cseq ← gc-config(cseq)
end operation
```

executes on a domain of objects

transfer latest tag-value pairs for each object from configs

```
procedure update-config(cseq , D)
   μ ← max({j : cseq[j].status = F})
   λ ← max({j : cseq[j] ≠ ⊥})
   M = []
   for o in D do
      M[o] ← ∅
   for i = μ : λ do
      for o in D do
         ⟨t, v⟩, _ ← cseq[i].cfg.get-data() for object o
         M[o] ← M[o] ∪ {⟨τ, v⟩}
   for o in D do
      ⟨τ, v⟩ ← max_t{⟨t, v⟩ : ⟨t, v⟩ ∈ M[o]}
      cseq[λ].cfg.put-data(⟨τ, v⟩) for object o
end procedure
```

gathers the tag-value pairs for each object o in D

transfers the maximum pair to the new config

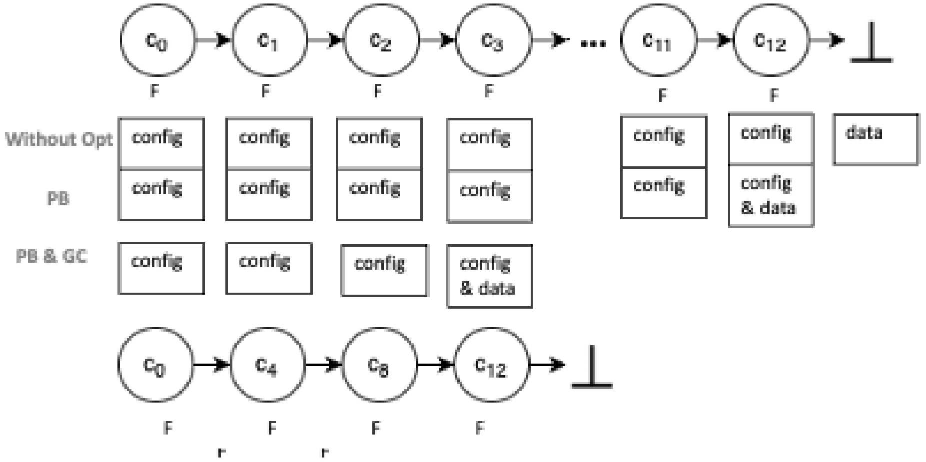
Optimization Results – Piggyback

alg./f_{size}	CoAresABDF	CoAresABDF PB	CoAresECF	CoAresECF PB
1MB	284ms	278ms	149ms	142ms
256MB	9s	5s (44%)	9.65s	3.82s (60%)
512MB	21.8s	15.2s (30%)	23.2s	10.9s (53%)

TABLE I: READ Operation - File Size - S:11, W:5, R:5

- **Non-Fragmented Algorithms:** No notable improvements for medium-sized objects.
  - Removal of **read-config** occurs only twice, so impact on latency is minimal.
- **CoARESF (256 MB & 512 MB):** Significant performance drops without optimization.
  - Non-optimized: **4 rounds per block** with double read-config.
  - With PB Optimization: Reduced to **2 rounds**, lower read latency.

Optimization Results – Garbage Collection



alg./f_{size}	ARES	ARES PB	ARES $PB\&GC$	CoARES	CoARES PB	CoARES $PB\&GC$	CoARESF	CoARESF PB	CoARESF $PB\&GC$
11 Pending Reconfiguration & 1 Finalized									
1MB	159ms	494ms	107ms	162ms	506ms	110ms	181ms	191ms	127ms
64MB	5.57s	27.4s	5.58s	5.81s	26.8s	5.73s	6.78s	6.62s	6.61s
12 Finalized Reconfiguration									
1MB	159ms	166ms	119ms	163ms	167ms	122ms	186ms	193ms	135ms
64MB	5.80s	5.76s	5.71s	5.88s	5.98s	5.82s	6.92s	6.73s	6.74s

TABLE II: READ Operation - Reconfigurations - S:11, W:1, R:10:, G:4

<u>Scenario 1:</u>

- PB version has the worst latency, since transfers data and config in 11 round trips.
- PB with GC is fastest, since it updates pointers reducing actions.
- CoARESF & Larger Objects (64MB): No differences between versions since the first block finds the latest config and the next block starts from that config.

<u>Scenario 2:</u>

- Original vs. PB has similar performance with one extra round trip.
- PB with GC is faster, skipping every 4 configurations, fewer rounds needed.

Conclusions and Future Work

- Used tracing to pintpoint inefficiencies by monitoring individual procedures.
- Develop optimizations, leading to ARES II.
- Show the **correctness** of ARES II and conduct performance **evaluations** to showcase its improvements over ARES.

Future Work - Devise strategies on **when** and **how** to introduce new configurations.

- Ensure that the system remains operational despite server failures.
- Improve performance by replacing older servers with more powerful ones.

* Monitoring tools to collect health metrics, threshold-based approaches for determine when to reconfigure, machine learning algorithms for anomaly detection, server rebalancing policy

Thank you!

For more information you can see the websites of our related projects:

HARISMA

MARS