Tracing the Latencies of Ares: A DSM Case Study

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

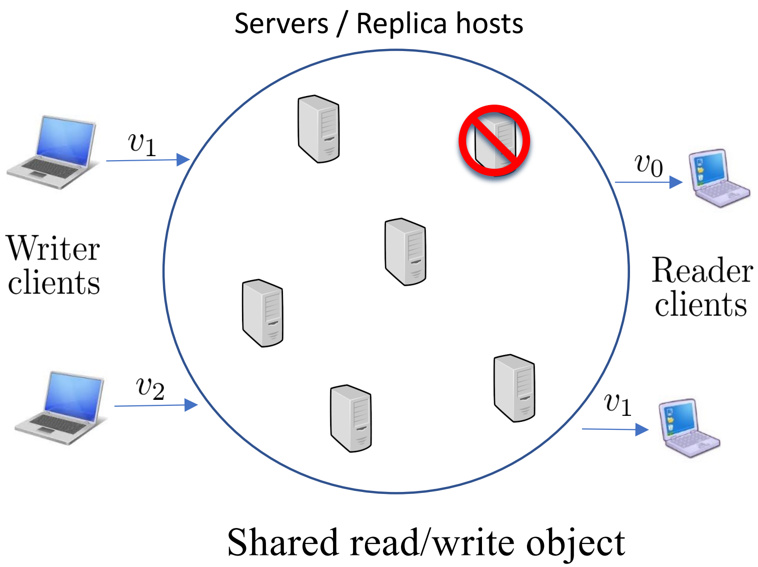
[1]University of Cyprus, Nicosia, Cyprus

[2]Algolysis, Limassol, Cyprus

ApPLIED 2024, Nantes, France

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

Distributed Shared Memory Emulations (DSMs)

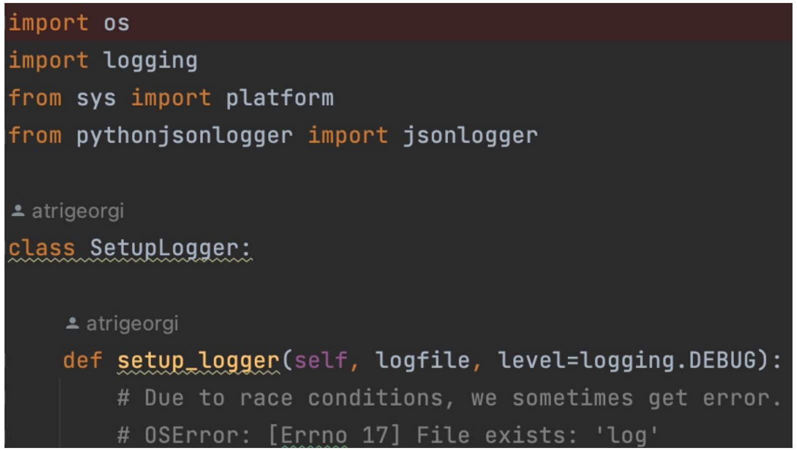


- A set of servers (configuration) maintain replicas of the same data object.
- Clients (readers/writers) access the object by sending messages to these servers.
- Read/Write operations are structured in terms of phases.
- Each phase consists of two communication exchanges (broadcast & convergecast).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

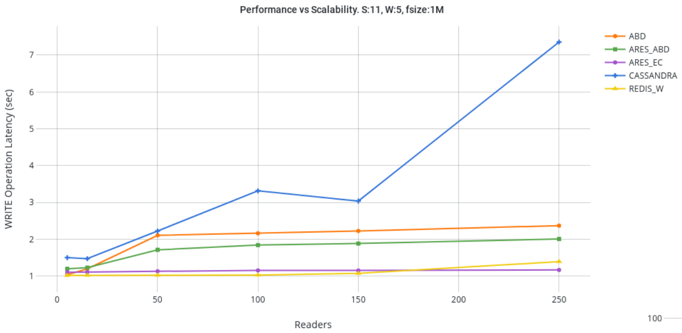
L. Lamport,“On Interprocess Communication,” Distributed Computing, vol. 1, no. 2, pp. 77–101, 1986.

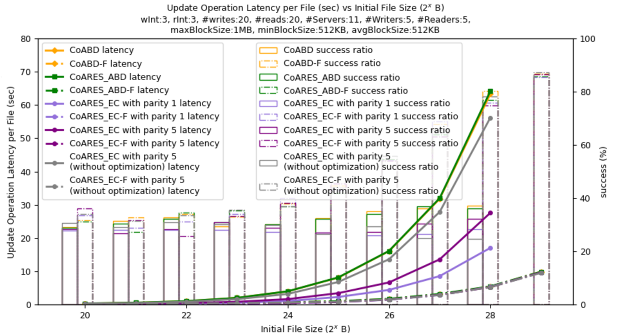
Performance Analysis Challenges in DSMs

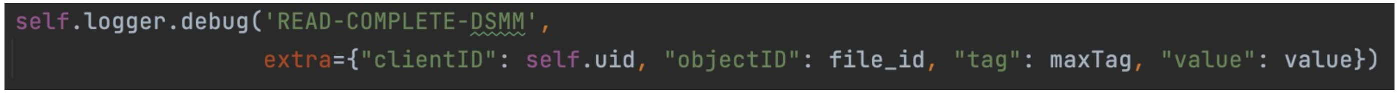
- Identifying performance bottlenecks in complex DSMs can be challenging
- Traditional logging techniques may not provide sufficient insight



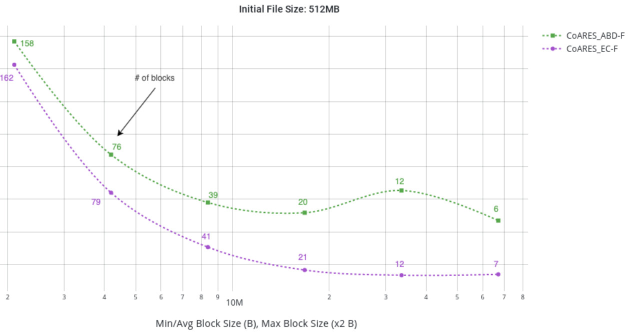
```
import os
import logging
from sys import platform
from pythonjsonlogger import jsonlogger

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'
```







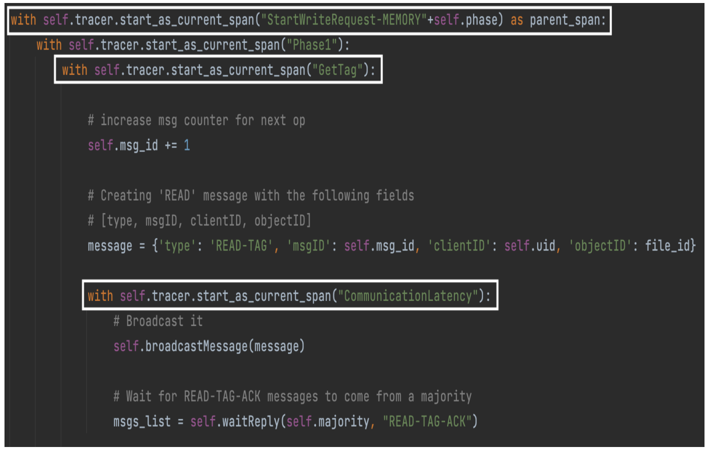
```
self.logger.debug('READ-COMPLETE-DSMM',
                  extra={"clientID": self.uid, "objectID": file_id, "tag": maxTag, "value": value})
```



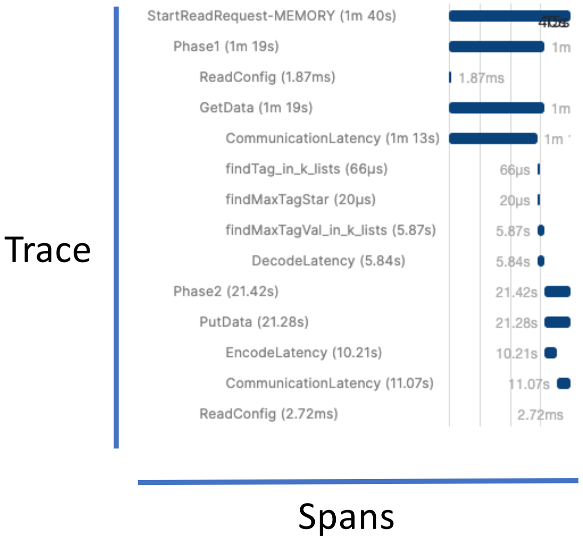
“**Distributing Tracing** is a monitoring technique used to track individual requests as they move across multiple components within a distributed system. It helps to pinpoint where failures occur and what causes poor performance.”

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")
```

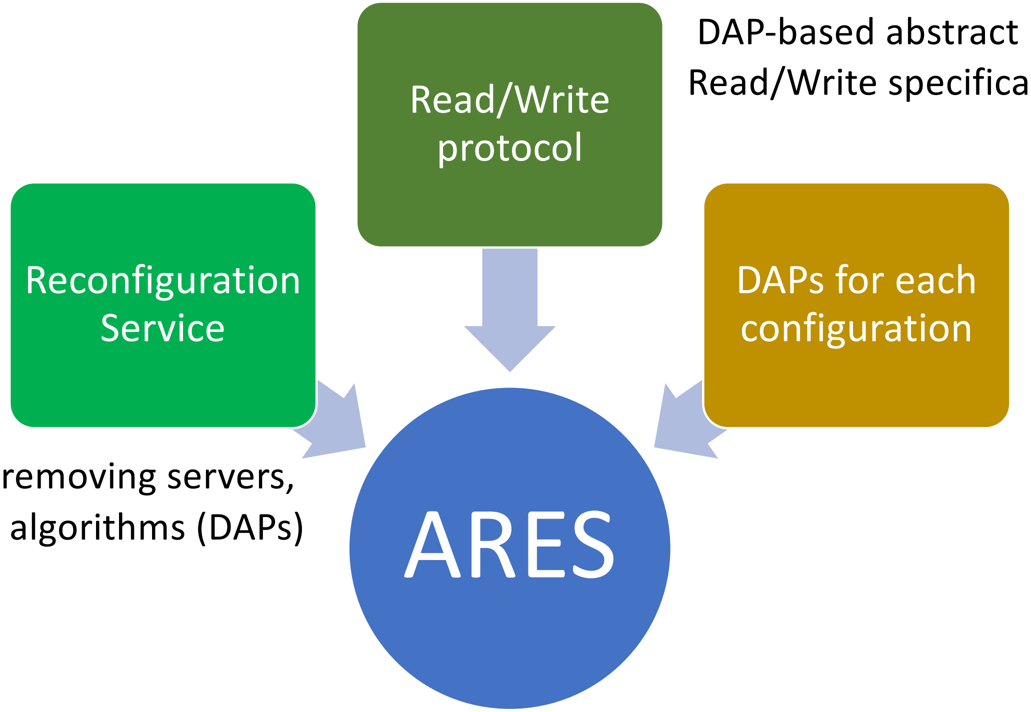


Main Objective

Our main objective is to bring Distributed Tracing into DSMs.

We will achive this through the ***ARES*** DSM.

ARES - Adaptive, Reconfigurable, Erasure Code, Atomic Storage



DAP-based abstract Read/Write specifications

define the exact methodology to access the object

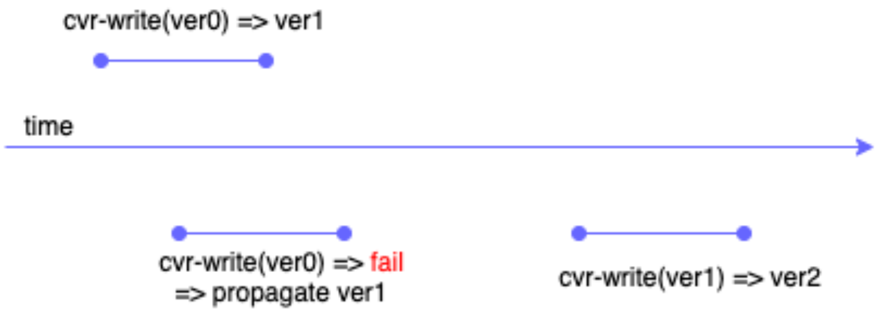
ABD-DAP, EC-DAP

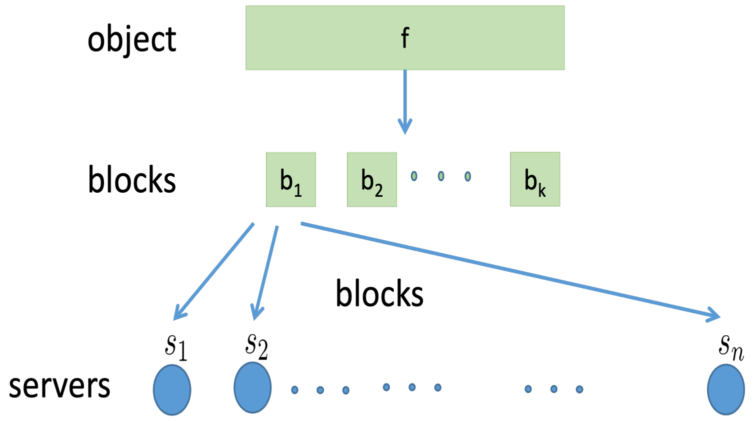
masks host failures by adding/removing servers, and switches between storage algorithms (DAPs)

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.

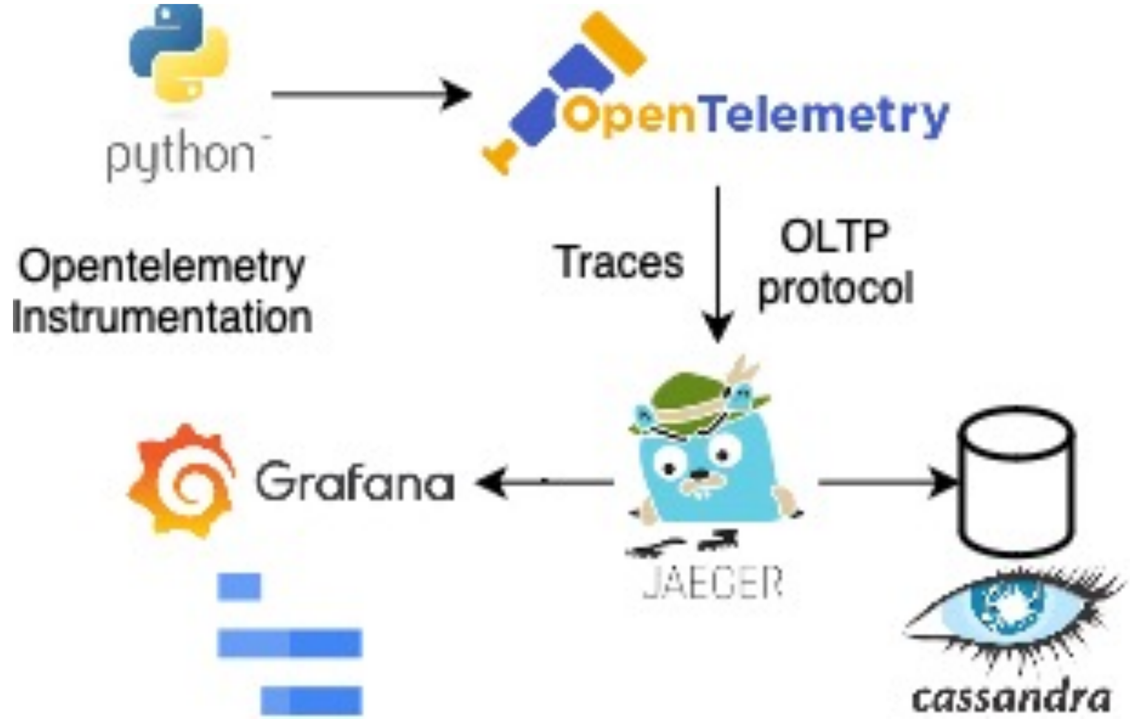
Evaluated Algorithms

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*.

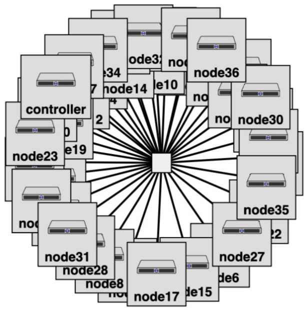




Methodology: ARES Distributed Tracing

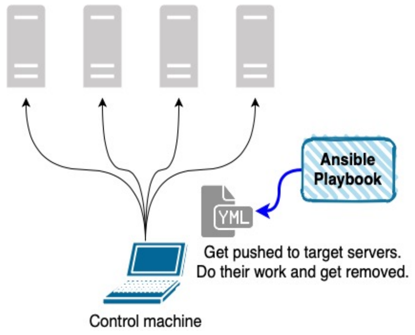


Experimental Setup



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.

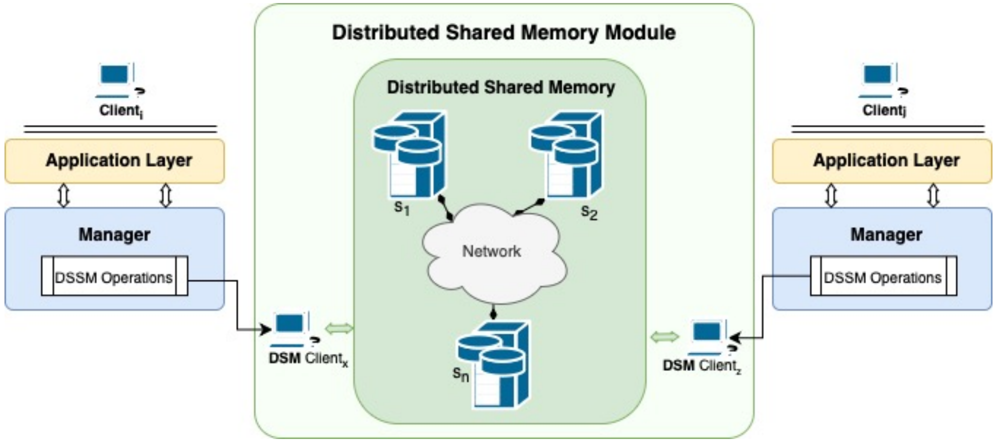

Ansible Playbook
YML
Get pushed to target servers. Do their work and get removed.
Control machine

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

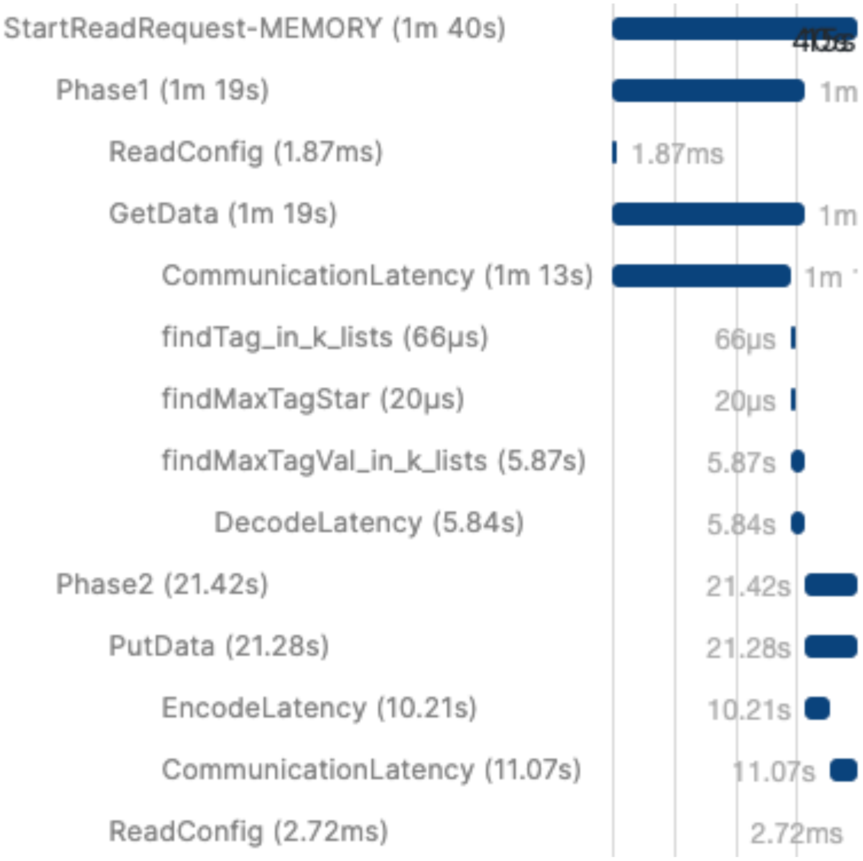
Debug Levels

Monitor read, write, and reconfig operations at two debug levels:

- **User:** This level includes the computation latency and the latencies for exchaning requests with the DSMM.
- **Memory:** This level includes communication and computation latencies within the DSMM.

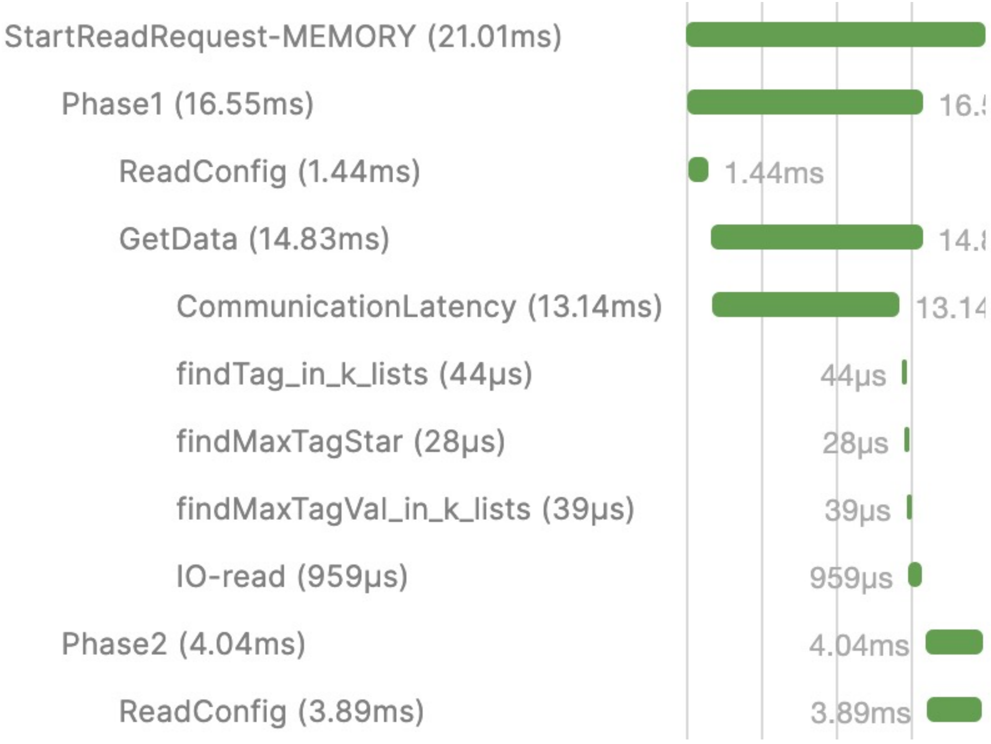


File Size



- StartReadRequest-MEMORY (1m 40s)
  - Phase1 (1m 19s)
    - ReadConfig (1.87ms)
    - GetData (1m 19s)
      - CommunicationLatency (1m 13s)
      - findTag_in_k_lists (66μs)
      - findMaxTagStar (20μs)
      - findMaxTagVal_in_k_lists (5.87s)
        - DecodeLatency (5.84s)
  - Phase2 (21.42s)
    - PutData (21.28s)
      - EncodeLatency (10.21s)
      - CommunicationLatency (11.07s)
    - ReadConfig (2.72ms)

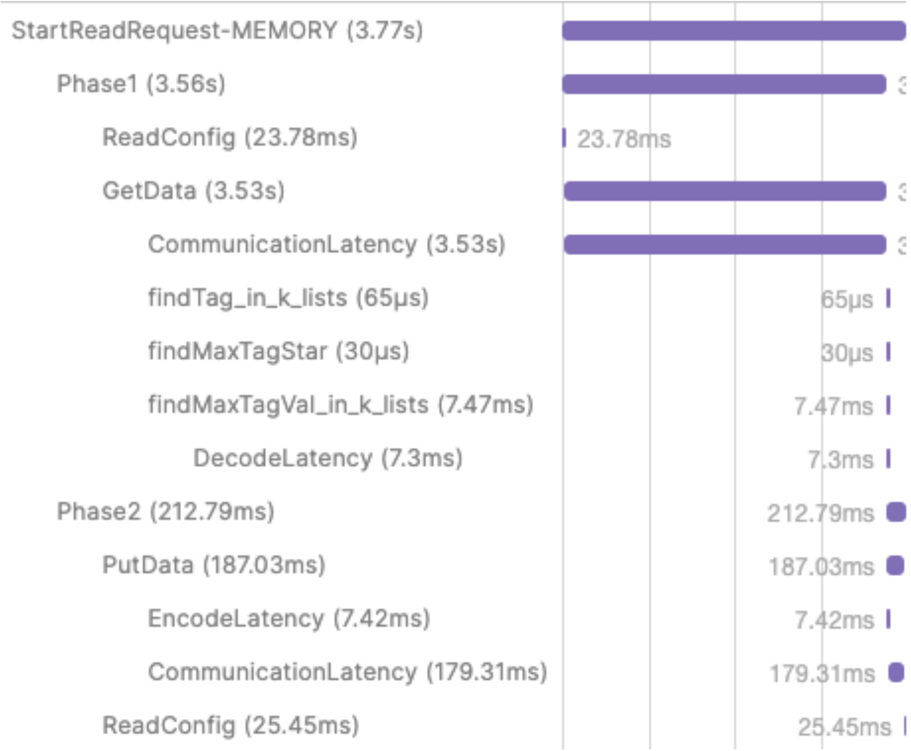
ARESEC, S:11, W:5, R:5, fsize:512MB, Debug Level:DSMM



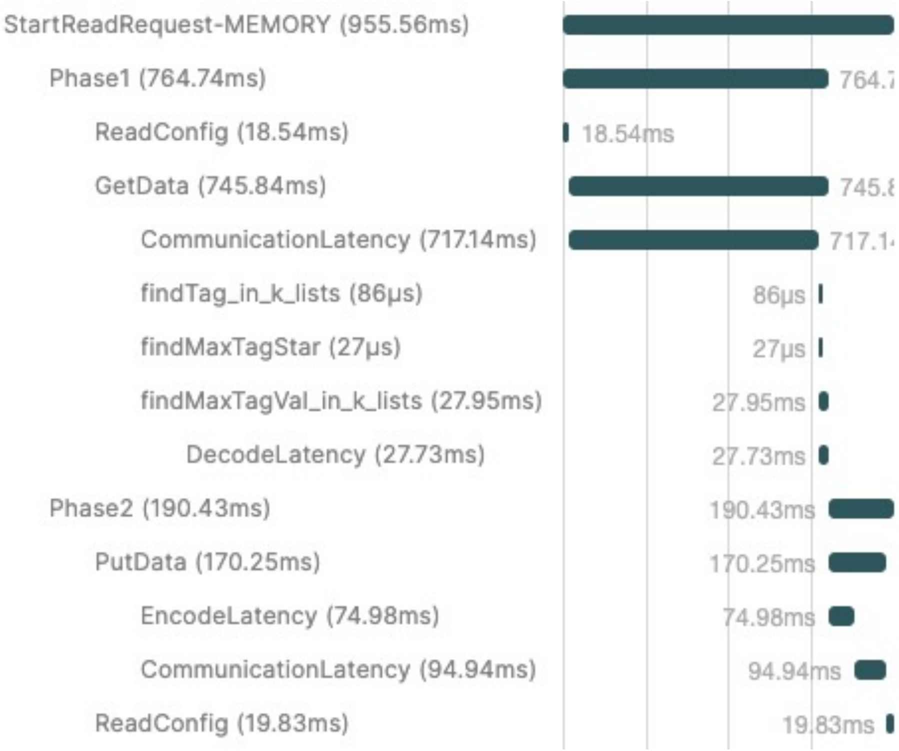
- StartReadRequest-MEMORY (21.01ms)
  - Phase1 (16.55ms)
    - ReadConfig (1.44ms)
    - GetData (14.83ms)
      - CommunicationLatency (13.14ms)
      - findTag_in_k_lists (44μs)
      - findMaxTagStar (28μs)
      - findMaxTagVal_in_k_lists (39μs)
      - IO-read (959μs)
  - Phase2 (4.04ms)
    - ReadConfig (3.89ms)

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

Participation Scalability

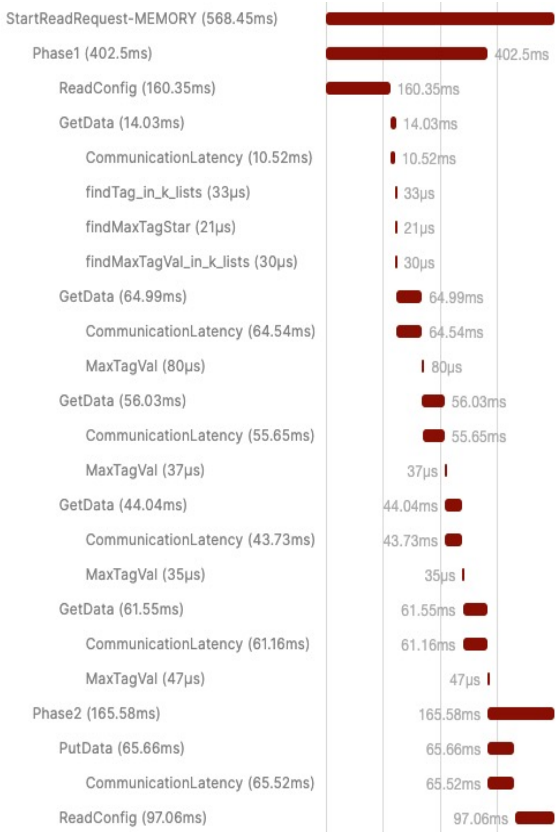


ARESEC, S:3, W:5, R:50, fsize:4MB, Debug Level:DSMM

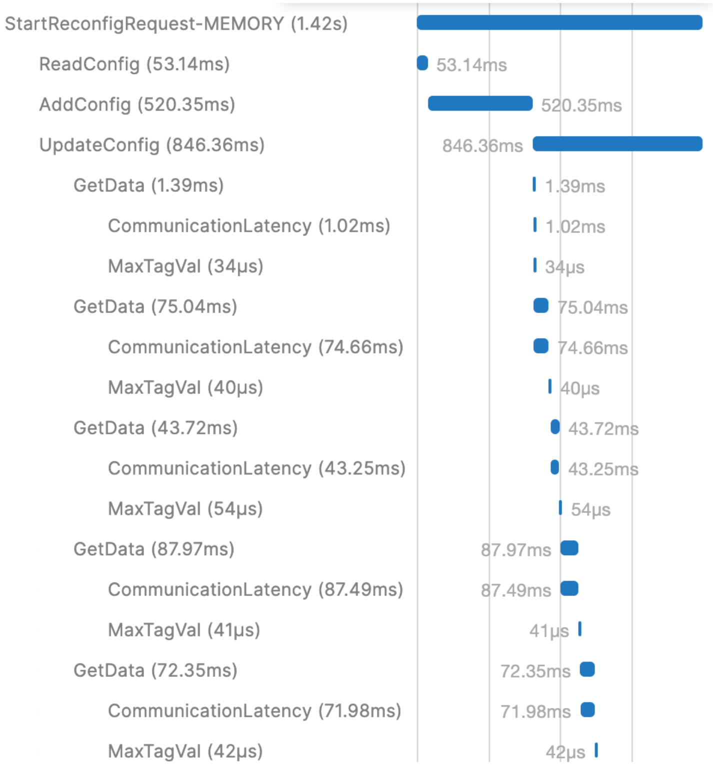


ARESEC, S:11, W:5, R:50, fsize:4MB, Debug Level:DSMM

Longevity

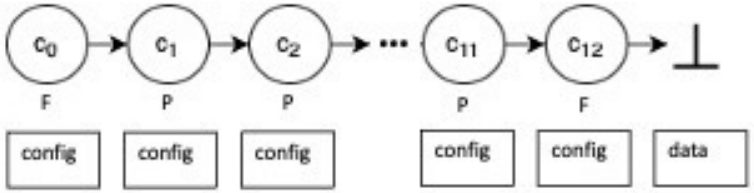


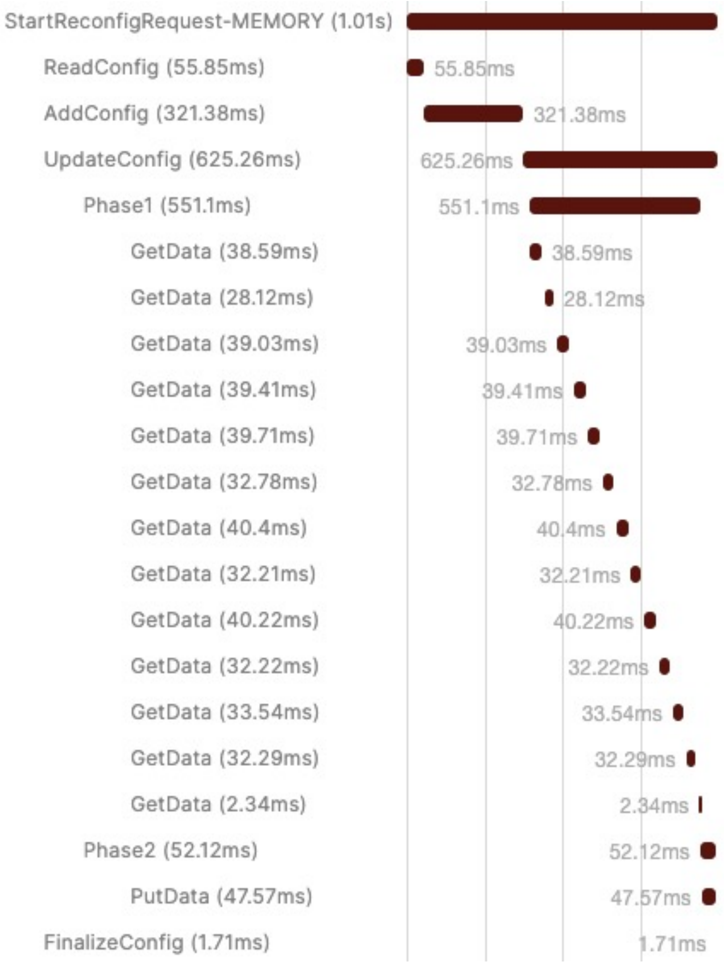
CoAresF, S:11, W:5, R:15, G=5, fsize:4MB, Debug Level:DSMM



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

The Latencies of **read-config** and **get-data**.




StartReconfigRequest-MEMORY (1.01s)
ReadConfig (55.85ms)
AddConfig (321.38ms)
UpdateConfig (625.26ms)
Phase1 (551.1ms)
GetData (38.59ms)
GetData (28.12ms)
GetData (39.03ms)
GetData (39.41ms)
GetData (39.71ms)
GetData (32.78ms)
GetData (40.4ms)
GetData (32.21ms)
GetData (40.22ms)
GetData (32.22ms)
GetData (33.54ms)
GetData (32.29ms)
GetData (2.34ms)
Phase2 (52.12ms)
PutData (47.57ms)
FinalizeConfig (1.71ms)

Conclusions

Distributed tracing is crucial for diagnosing and resolving performance issues in DSM algorithms.

Optimization Strategies

- **Piggy-backing**: Integrating configurations with read/write messages to expedite configuration discovery.
- **Garbage Collection**: Eliminating obsolete configurations for quicker access to the latest data.
- **Data Batching**: A single reconfiguration across multiple objects to enhance efficiency.

Thank you!

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

A. Trigeorgi et al. An Experimental Evaluation - SSS 2022 17