University of Connecticut

institute imdea networks

COMPUTATIONALLY LIGHT "MULTI-SPEED" ATOMIC MEMORY

ANTONIO FERNÁNDEZ ANTA, IMDEA NETWORKS

THEOPHANIS HADJISTASI, UCONN

NICOLAS NICOLAOU, IMDEA NETWORKS

On Principles Of DIstributed Systems - OPODIS

Madrid, Spain - December 2016

Problem

Goal: Emulate atomic read/write shared objects in an *asynchronous, messaging-passing, crash-prone* system

Approach: Objects are ***replicated*** to cope with crashes

Challenge: Providing ***consistency*** when read and write operations concurrently access different replicas

- ***Atomicity*** [L79] (*or linearizability* [HW90]) is the most intuitive semantic, providing the ***illusion*** of a single-copy object.

Challenge: Making read and write operations ***efficient***

- In particular, in terms of ***communication*** and ***computation demands***

System Model

Components	• Clients: 1 writer & R readers (SWMR) • Servers: S replica hosts
Operations	• write(*v*): updates the object value to *v* • read(): retrieves the object value • Well-Formedness (only a single operation at a time)
Communication	• Asynchronous • Message-Passing • Reliable Channels (messages are not lost or altered)
Failures	• Crashes • Any reader or the writer • *Up to* ***f*** *servers may fail where,* $f < |S|/2$

Efficiency Metrics

- ***Computation time***: computation ***steps*** in each operation

- ***Communication delay***: number of ***communication exchanges***
  - *Communication Exchange*: a set of sends and matching receives for a specific message type within the protocol.

One ABD-style round is thus equivalent to 2 exchanges

Algorithm ABD: Recalling the past

[Attiya, Bar-Noy, Dolev 1996] (Dijkstra Prize 2011)

Order Operations by using <ts, v> pairs.

Reader Protocol (2 phases)

- Phase 1:
  - Send read to all
  - Collect <ts,v> from a majority
  - Discover max(<ts,v>)
- Phase 2:
  - Send max(<ts,v>) to all
  - Collect ack from a majority and return v

Writer Protocol

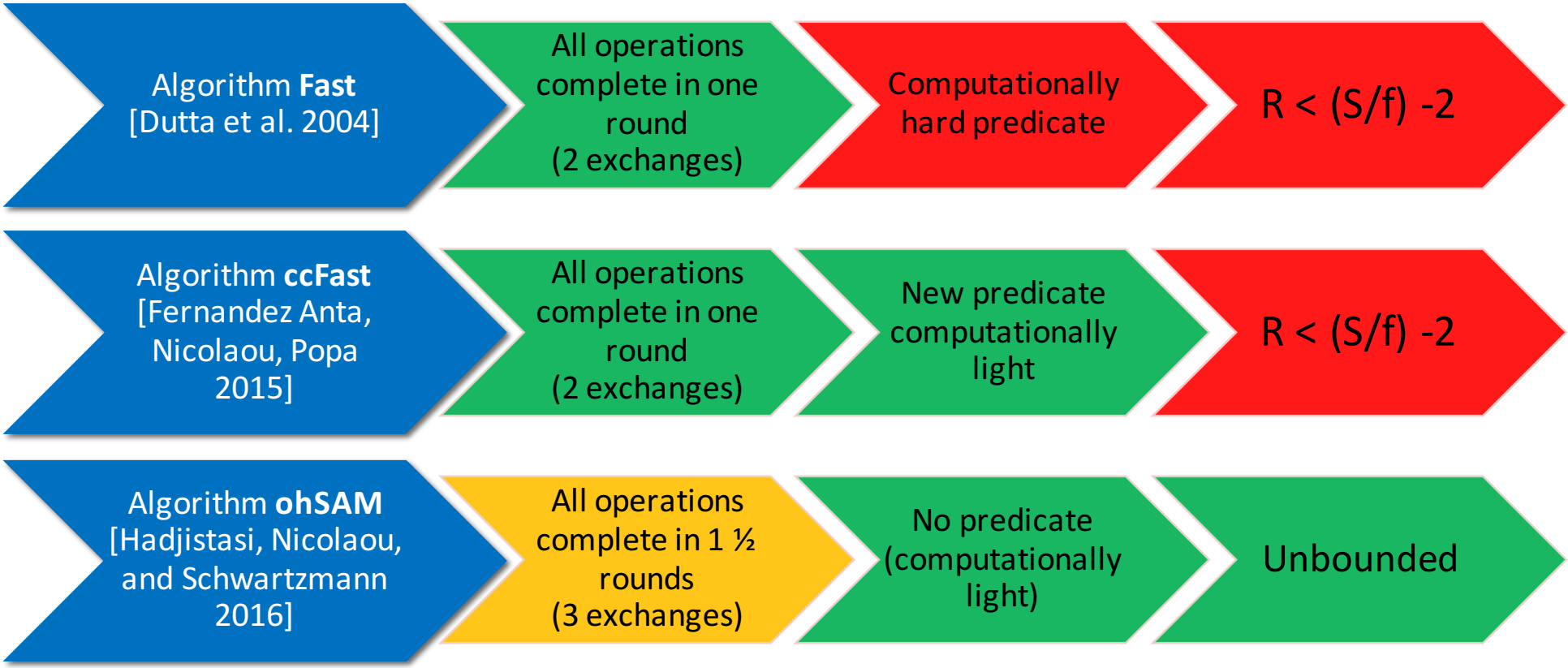
- ts++ //increment ts
- Send <ts,v> to all
- Wait for a majority to reply

Server Protocol (Upon rcv a msg m from i)

- if m.ts > ts
  - Update local <ts,v>
- Send local <ts,v> to i

Reads must Write!
(2 round-trips)

The world of “fastness”



Algorithm ccFast: Predicate

$$\exists \alpha \in [1, R+1] \text{ s.t.}$$
$$MS = \{s : s.ts = maxTS \ \wedge \ s.views \geq \alpha\} \text{ and } |MS| \geq S - \alpha f$$

Server Protocol (Upon rcv a msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
- else
  - Add i in seen set
- Send local ts and the size of the seen set to i

Reader Protocol (1 phase)

- Send read to all
- Collect <ts,v> and views from S-f servers
- Discover maxTS = max(ts)
- If predicate is true:
  - return maxTS
- else
  - return maxTS-1

Writer Protocol (2 exch)

(Same as ABD)

How many and not which observed <ts,v>

Contributions

Question: Can we be fast when conditions allow it, and switch to a slower mode when conditions may violate atomicity?

- Provide 2 new "multi-speed" algorithms:
  - **ccHybrid**
    - When seen set **at the reader** is below a threshold use ccFast predicate
    - Otherwise perform two round (ABD-like) operation
  - **ohFast**:
    - If the seen **at the server** is below the threshold the server replies directly to the reader and the reader uses the ccFast predicate
    - Otherwise it relays the read before replying
- Complement the algorithms with experimental results.

Algorithm ccHybrid

Server Protocol (Upon rcv a msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - Prop = false
- else
  - Add i in seen set
- If m.ts = ts and i reader
  - prop = true
- Send local ts and the size of the seen set to i

Writer Protocol (1 phase)

(Same as ABD)

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

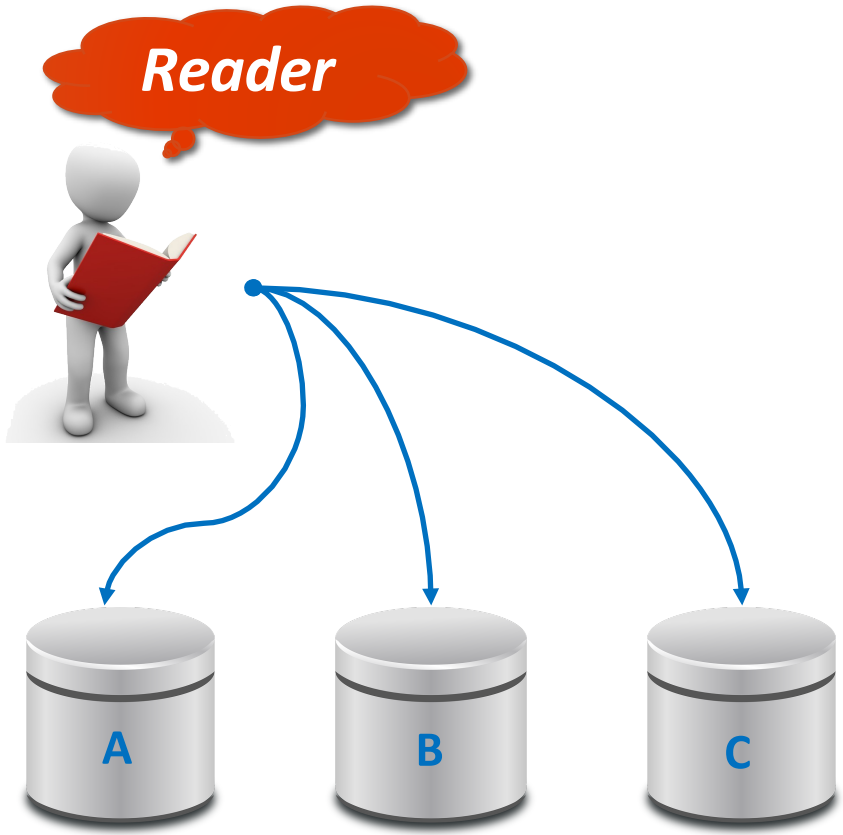
Set of Servers {A,B,C }

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

Communication Exchange 1



Invoker sends a Read Request to all servers

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

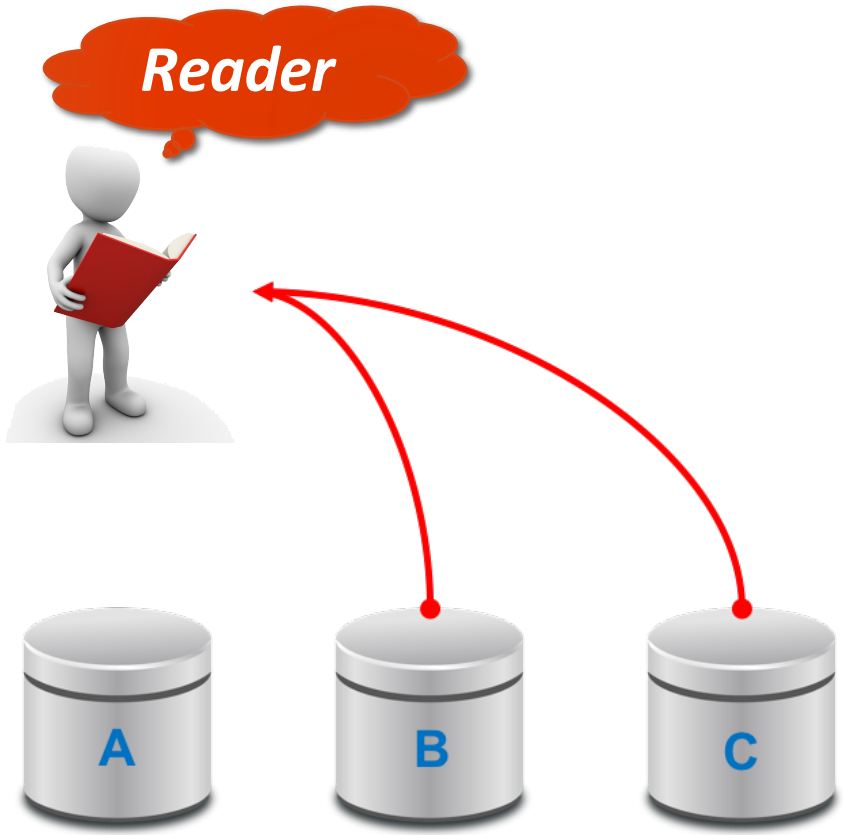
ccHybrid: Visually

Server Protocol (Upon rcv a msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - Prop = false
- else
  - Add i in seen set
- If m.ts = ts and i reader
  - prop = true
- Send local ts and the size of the seen set to i

Servers receive the Read Requests and Update their local Information

ccHybrid: Visually



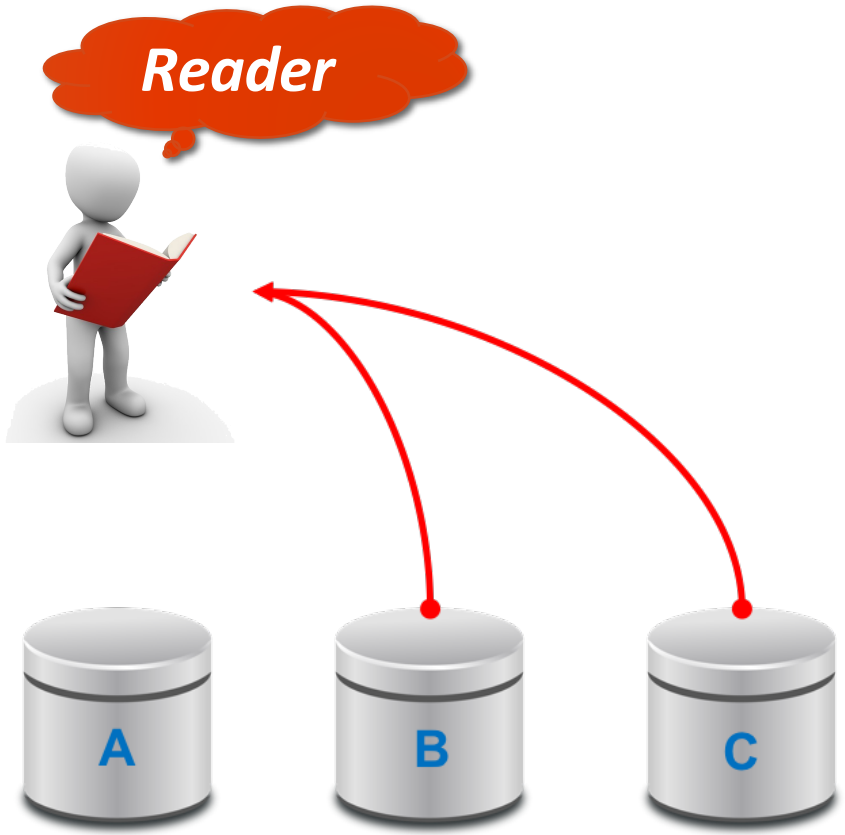
Server Protocol (Upon rcv a msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - Prop = false
- else
  - Add i in seen set
- If m.ts = ts and i reader
  - prop = true
- Send local ts and the size of the seen set to i

Servers Reply back to the Invoker

ccHybrid: Visually

Communication Exchange 2



Reader collects replies and discovers maxTs, maxVS, and propagate set

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

Reader checks threshold on maxVS

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

If Seen set is above Threshold

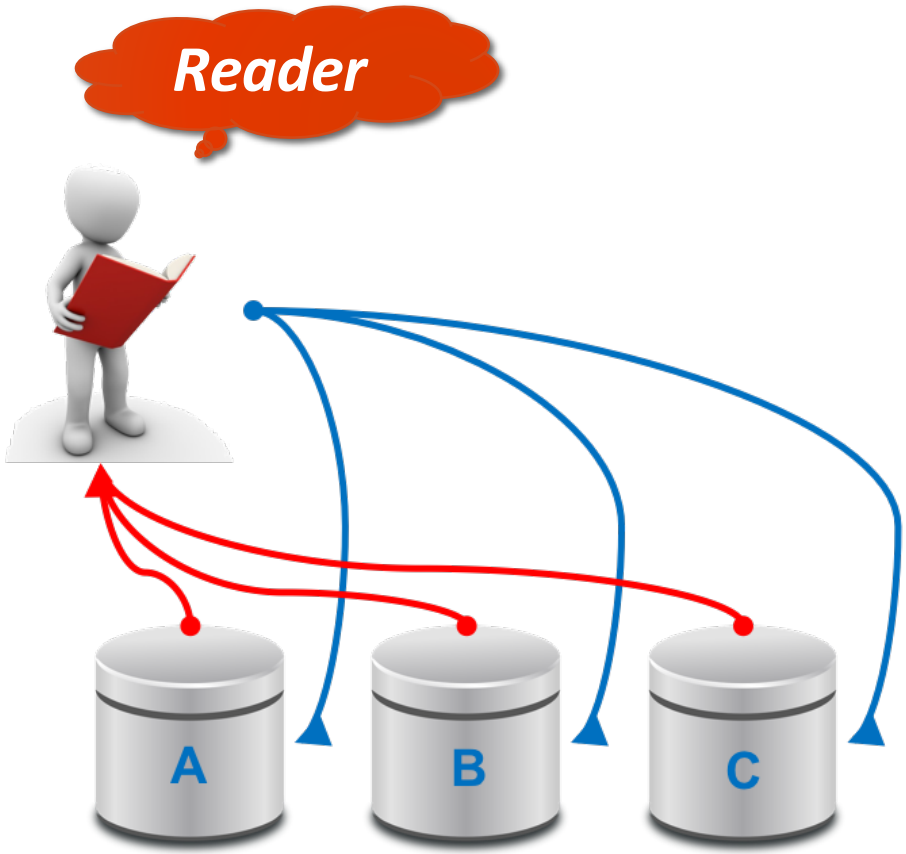
check if propagated to "enough" servers

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

Communication Exchange 3,4



If "few" servers in propagation set then propagate the maxTS

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

Communication Exchange 4

After propagation return maxTS

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

ccHybrid: Visually

Communication Exchange 2

If seen set was bellow threshold

-> read op will be fast

-> check the predicate on what to return

Reader Protocol (2-4 exch)

- Send read to all
- Collect <ts,v>, prop, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
  - prSet = {s:s.prop=1}
- If maxVs > S/f – 2:
  - If |prSet| < f+1:
    - Propagate maxTS to S-f servers
  - return maxTS
- else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Algorithm: ccHybrid

Read Latency: 2 or 4 communication exchanges
(1 or 2 rounds)

Read Message Complexity: $4|S|$

THEOREM: Algorithm ccHYBRID implements an atomic SWMR read/write register.

Algorithm OHFAST

Server Protocol (Upon rcv a **READ** msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If (i reader) and (|seen| > S/f – 2) and securedts=false and Relays[i]<ts
  - Update local information
  - Send relay to all servers
- Else
  - Send local ts and size of seen set and securedts to i

Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Writer Protocol (2 exch)

(Same as ABD)

Algorithm OHFAST

Server Protocol (Upon rcv a **RELAY** msg m from s)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If Relays[i]=m.ts
  - Add s in srvRelay set
- If |srvRelay| > S-f
  - If (ts = m.ts) then set (securedts = true)
  - Send Ack message to the invoker i
- Else
  - Reply back to the relay sender

OHFAST: Visually

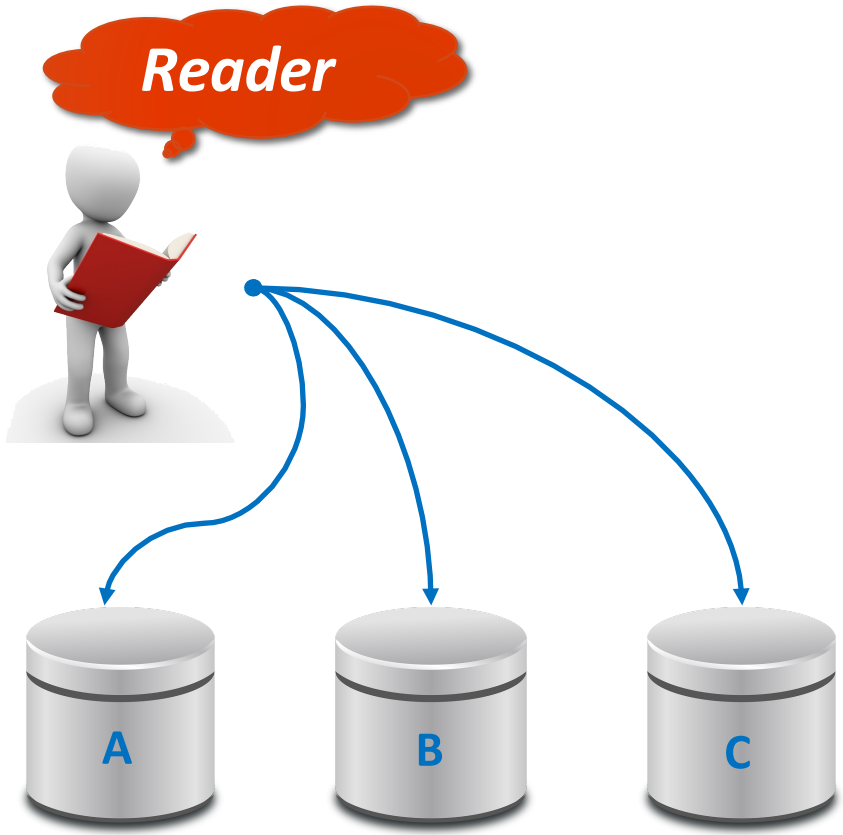
Set of Servers {A,B,C }

Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

OHFAST: Visually

Communication Exchange 1



Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Invoker sends a Read Request to all servers

OHFAST: Visually

Servers receive the Read Requests
Update their local info

Server Protocol (Upon rcv a **READ/WRITE** msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If (i reader) and (|seen| > S/f - 2) and securedts=false and Relays[i]<ts
  - Update local information
  - Send relay to all servers
- Else
  - Send local ts and size of seen set and securedts to i

OHFAST: Visually

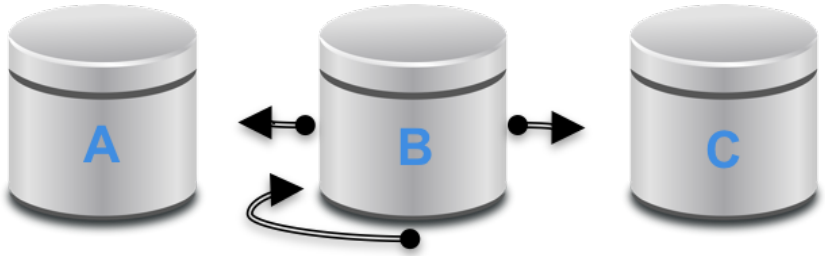
Decide if will relay or not

Server Protocol (Upon rcv a **READ/WRITE** msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If (i reader) and (|seen| > S/f - 2) and securedts=false and Relays[i]<ts
  - Update local information
  - Send relay to all servers
- Else
  - Send local ts and size of seen set and securedts to i

OHFAST: Visually

Communication Exchange 2



If decision was to replay

Server Protocol (Upon rcv a **READ/WRITE** msg m from i)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If (i reader) and (|seen| > S/f - 2) and securedts=false and Relays[i]<ts
  - Update local information
  - Send relay to all servers
- Else
  - Send local ts and size of seen set and securedts to i

OHFAST: Visually

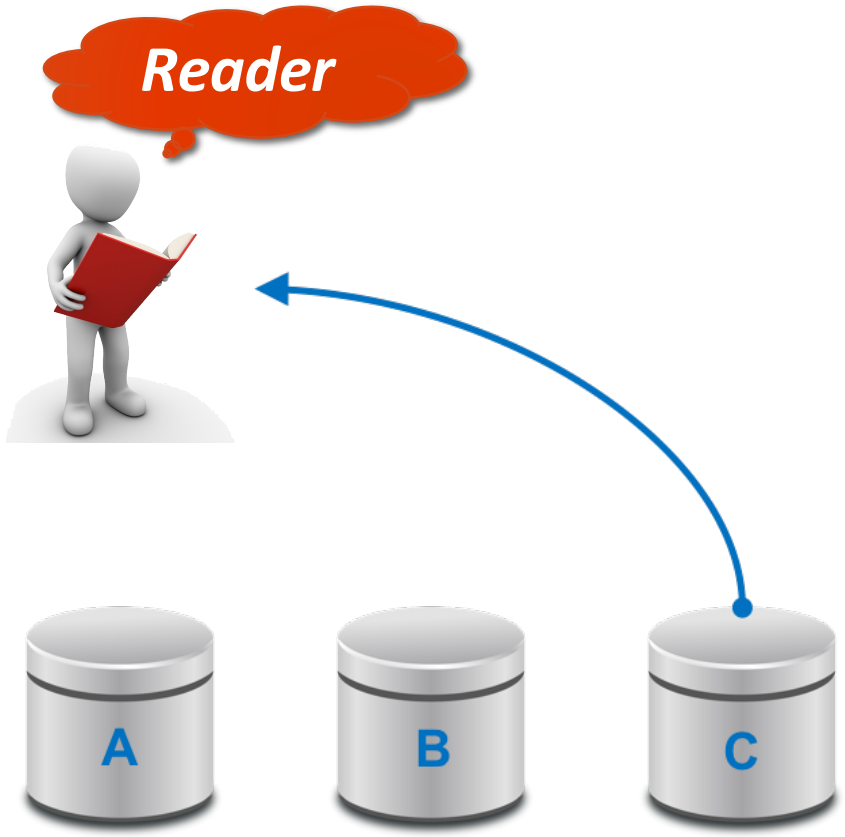
If decision was to replay
Update their local information

Server Protocol (Upon rcv a **RELAY** msg m from s)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If Relays[i]=m.ts
  - Add s in srvRelay set
- If |srvRelay| > S-f
  - If (ts = m.ts) then set (securedts = true)
  - Send Ack message to i
- Else
  - Reply back to sender s

OHFAST: Visually

Communication Exchange 3



Decide if they will send message to the reader

Server Protocol (Upon rcv a **RELAY** msg m from s)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If Relays[i]=m.ts
  - Add s in srvRelay set
- If |srvRelay| > S-f
  - If (ts = m.ts) then set (securedts = true)
  - Send Ack message to i
- Else
  - Reply back to sender s

OHFAST: Visually

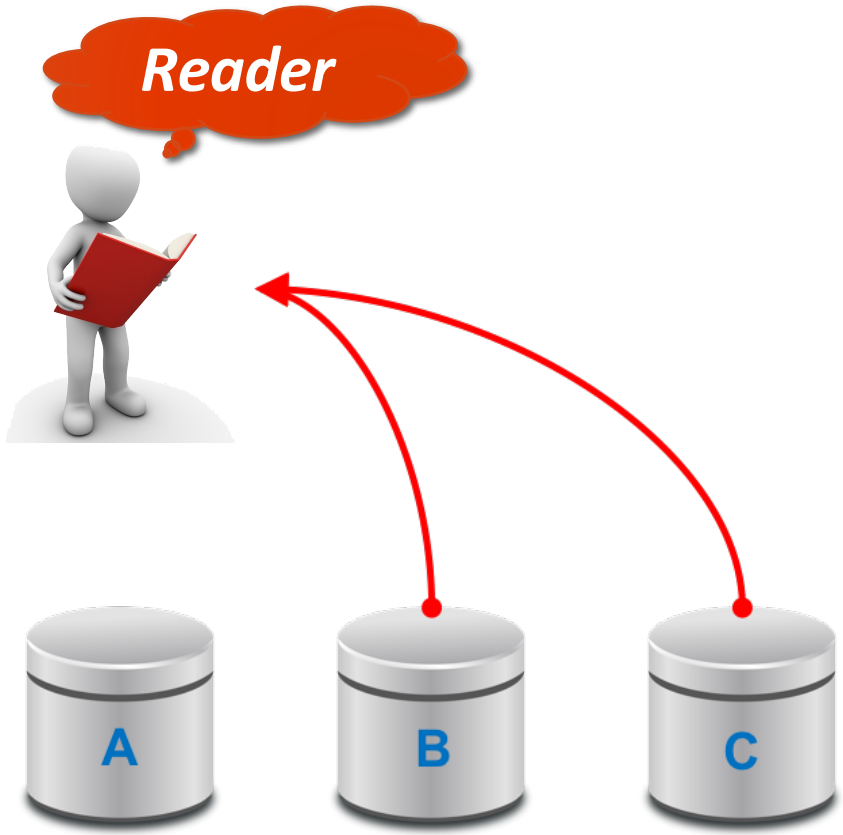
Communication Exchange 3

Decide if they will send message to the reader
Or reply back to the server

Server Protocol (Upon rcv a **RELAY** msg m from s)

- if m.ts > ts
  - Update local info
  - Set seen = {i}
  - securedts = false
- Else
  - add i in seen set
- If Relays[i]=m.ts
  - Add s in srvRelay set
- If |srvRelay| > S-f
  - If (ts = m.ts) then set (securedts = true)
  - Send Ack message to i
- Else
  - Reply back to sender s

OHFAST: Visually



Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Reader Collects messages from the servers from either communication exchange 2 or 3.

OHFAST: Visually

Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Discover Maximum Timestamp and the maximum views

OHFAST: Visually

Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Check if it is "safe" to return the maximum timestamp

OHFAST: Visually

Reader Protocol (2-3 exch)

- Send read to all
- Collect <ts,v>, secured, and views from S-f servers
- Discover
  - maxTS = max(ts)
  - maxVS = max(views)
- If ∃ m s.t. (m.ts=maxTS) and (m.secured=true):
  - return maxTs
- Else:
  - If predicate is true:
    - return maxTS
  - else
    - return maxTS-1

Check if it is "safe" to return the maximum timestamp
Else run the predicate to decide which value to return.

OHFAST READ PROTOCOL

Read Latency: 2 or 3 communication exchanges (1 or 1 ½ rounds)

Read Message Complexity: $|S^2|$

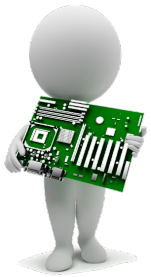
THEOREM: Algorithm OHFAST implements an atomic SWMR read/write register.

Overall Algorithm Comparison

	Computation	Communication	Participation Bounds
ABD [ABD96]	Light	Heavy (4 exch)	Unbounded
FAST [2004]	Heavy (NP-hard)	Light	R < S/f - 2
ccFAST [FNP15]	Light	Light	R < S/f - 2
OhSAM [HNS16]	Light	Moderate (3 exch)	Unbounded
ccHYBRID (here)	Light	Light/Heavy	Unbounded
ohFAST (here)	Light	Light/Moderate	Unbounded

Simulations

We implemented algorithms **ABD** [Attiya, Bar Noy, Dolev 1996], **OHSAM** [Hadjistasi, Nicolaou, Schwartzmann 2016], **SF** [Georgiou, Nicolaou, Shvartsman 2006], **CCHYBRID** and **OHFAST** using the *NS3* discrete event simulator.

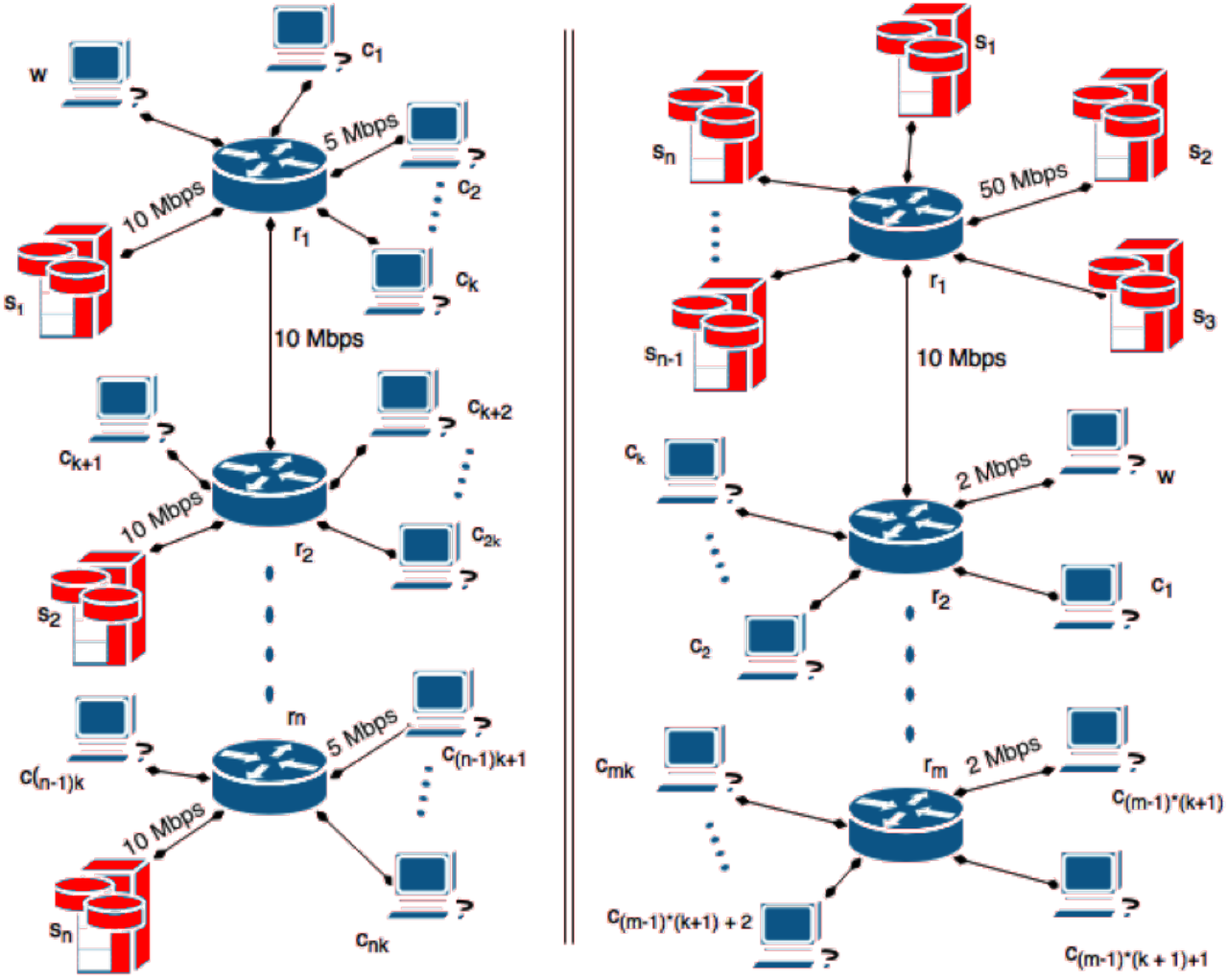


Experimentation Platform:

- Single Writer w, Set of Readers $\mathcal{R}$, Set of Servers $\mathcal{S}$.
- ***f=1*** servers may fail. Introduces high concurrency and inconsistency in the system.
- Communication between nodes is established via ***point-to-point*** bidirectional links implemented with ***DropTail*** queue.
- Two topologies are developed, ***Sparse*** and ***Condensed***.

Topologies

Topologies mainly differ on the deployment of the server nodes.



Scenarios

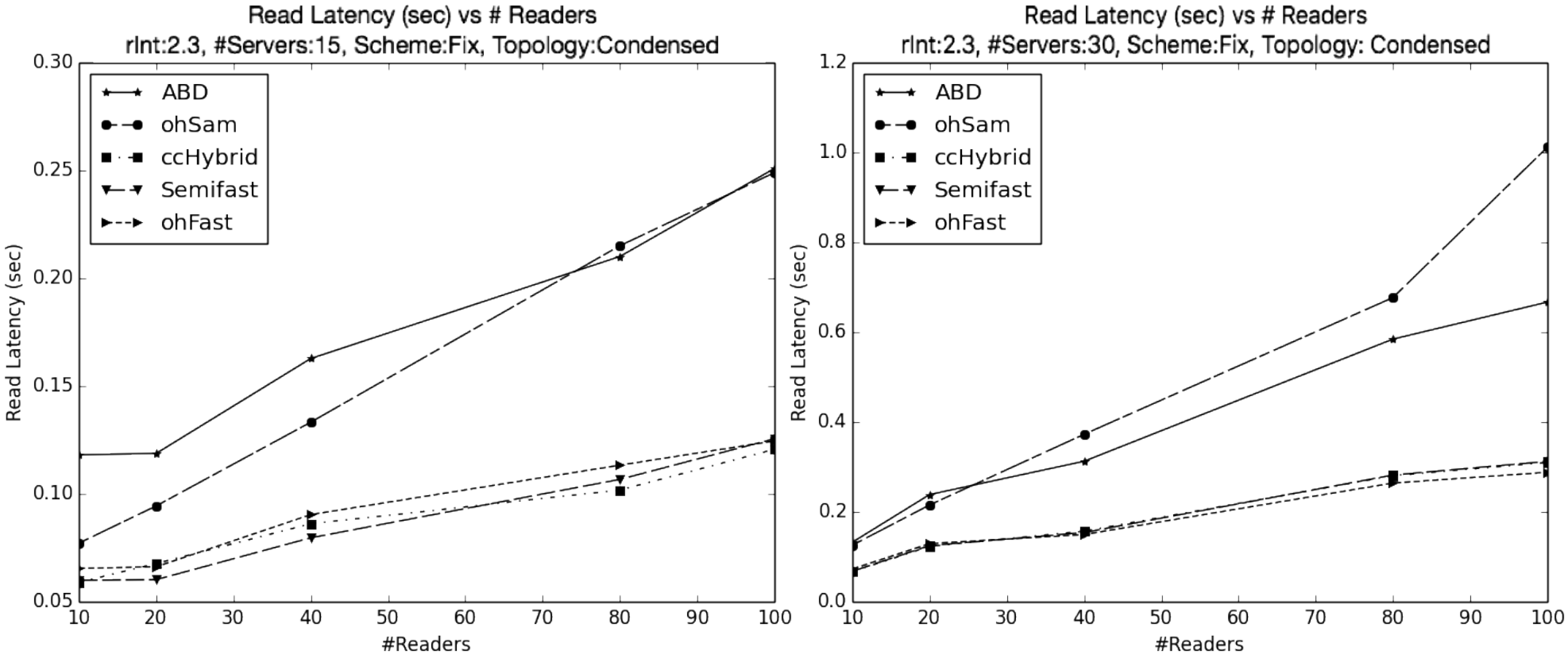
Performance of the algorithms is measured in terms of

- ***communication burden:*** the ratio of the number of fast over slow operations
- ***operation latency:*** the total time it takes for an operation to complete

Scenarios:

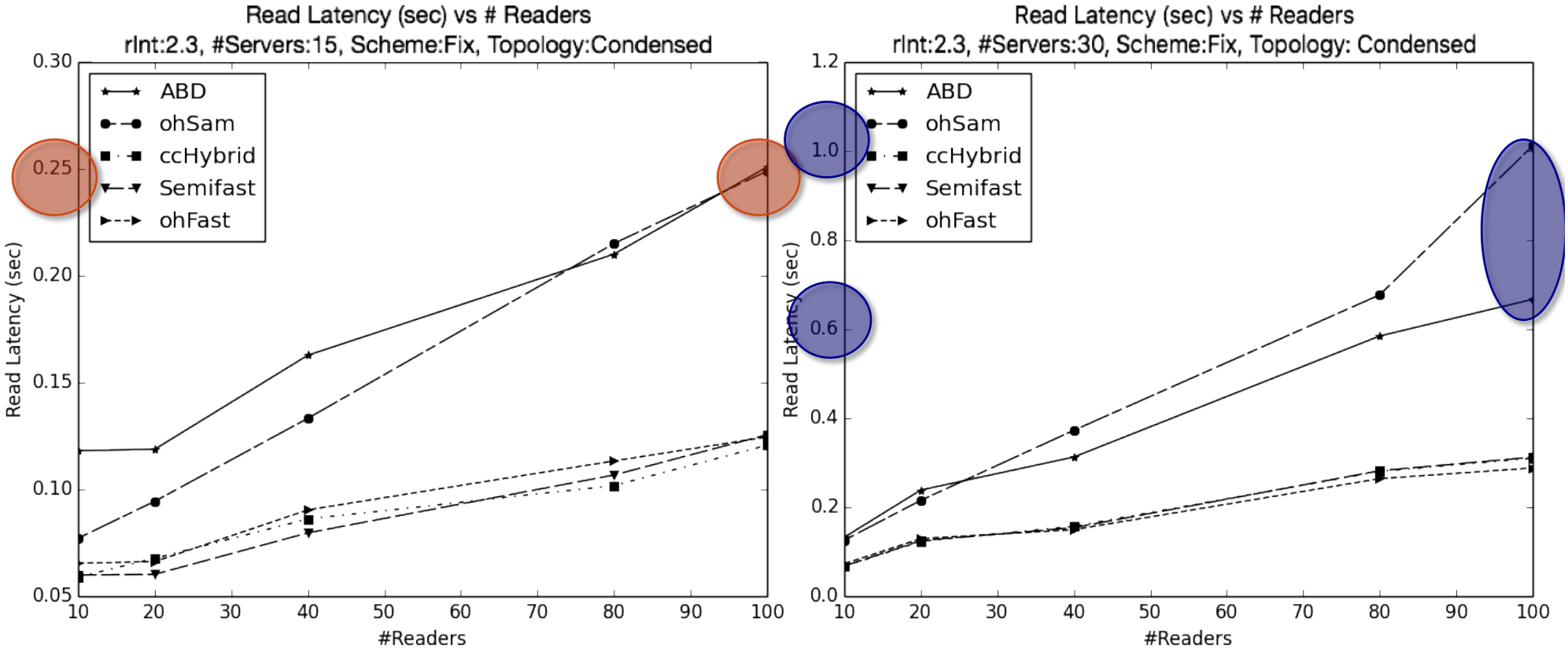
- Test **scalability** of the algorithms as the number of the participants increases, $\mathcal{R} \in [10,20,40,80,100]$, $S \in [10,15,20,25,30]$.
- Test **contention** we specify the frequency of read operations and we run the algorithms for different read intervals ***rInt*** $\in [2.3,4.6,6.9]$
- We define two invocation schemes,
  - **fix** – operations invoked at the read interval *rInt.*
  - **stochastic** – operations invoked at random interval between [1…*rInt*].

Empirical Results



Scalability: the increasing number of readers and servers has a negative impact on all the algorithms.

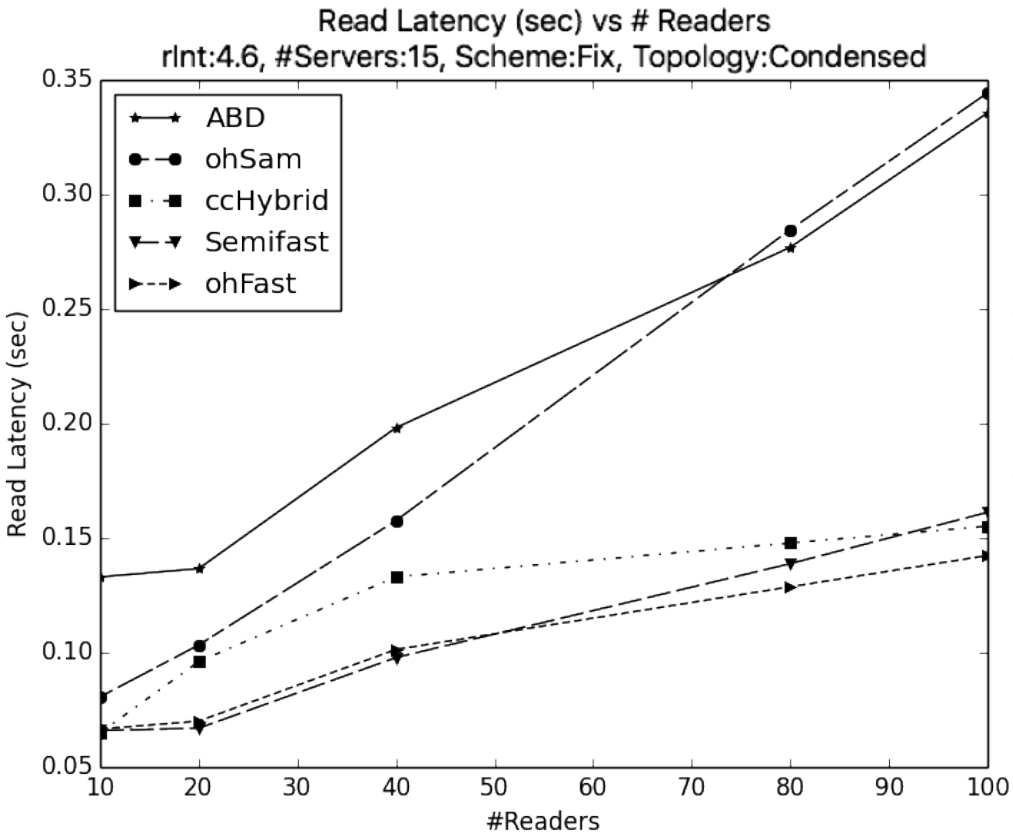
Empirical Results

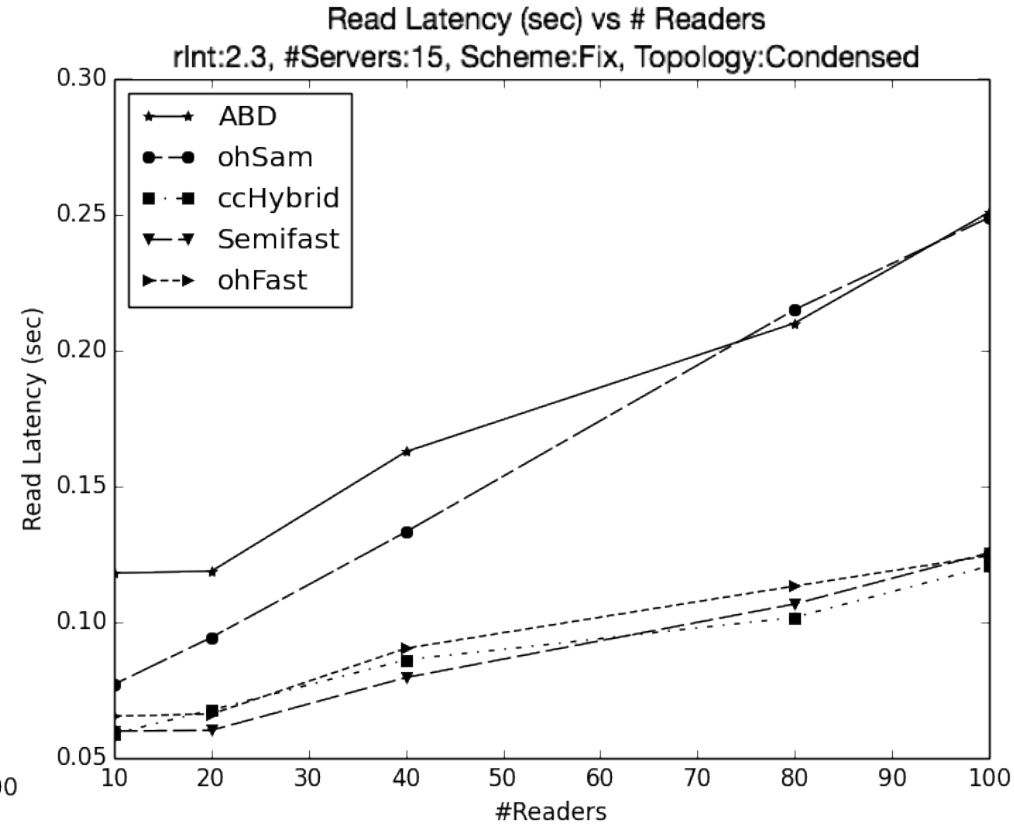


Scalability: the increasing number of readers and servers has a negative impact on all the algorithms.

- The impact is higher on the "single-speed" algorithms **ABD** and **OHSAM**.

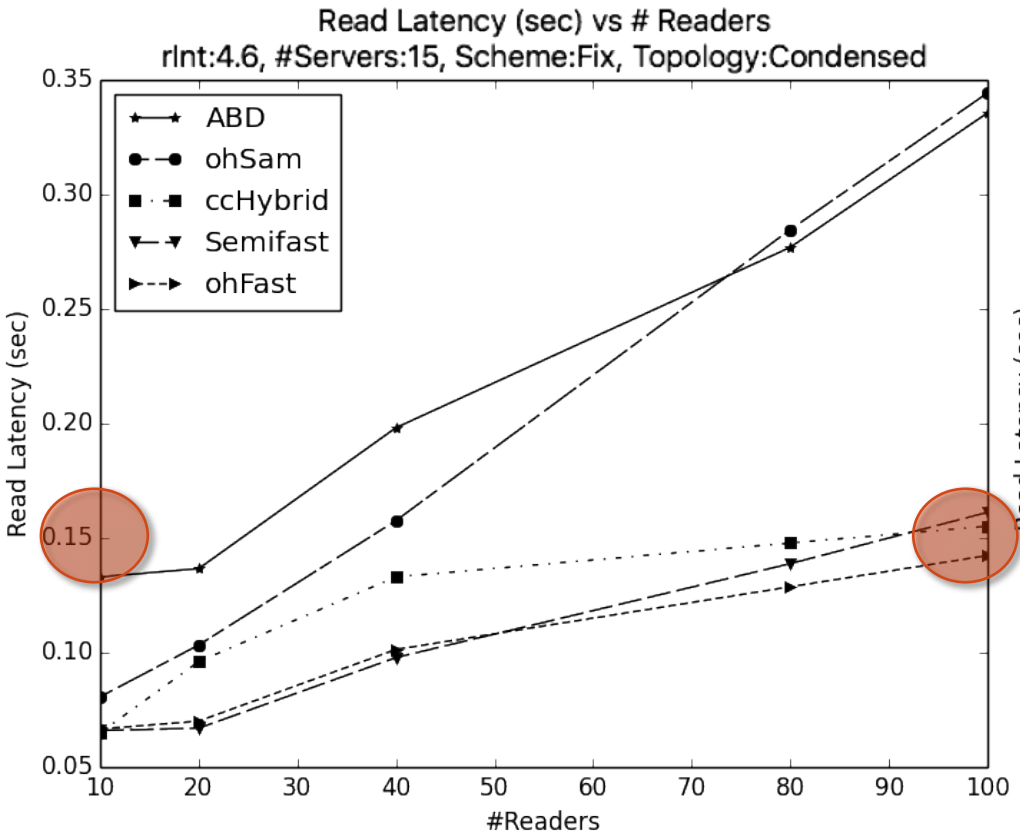
Empirical Results

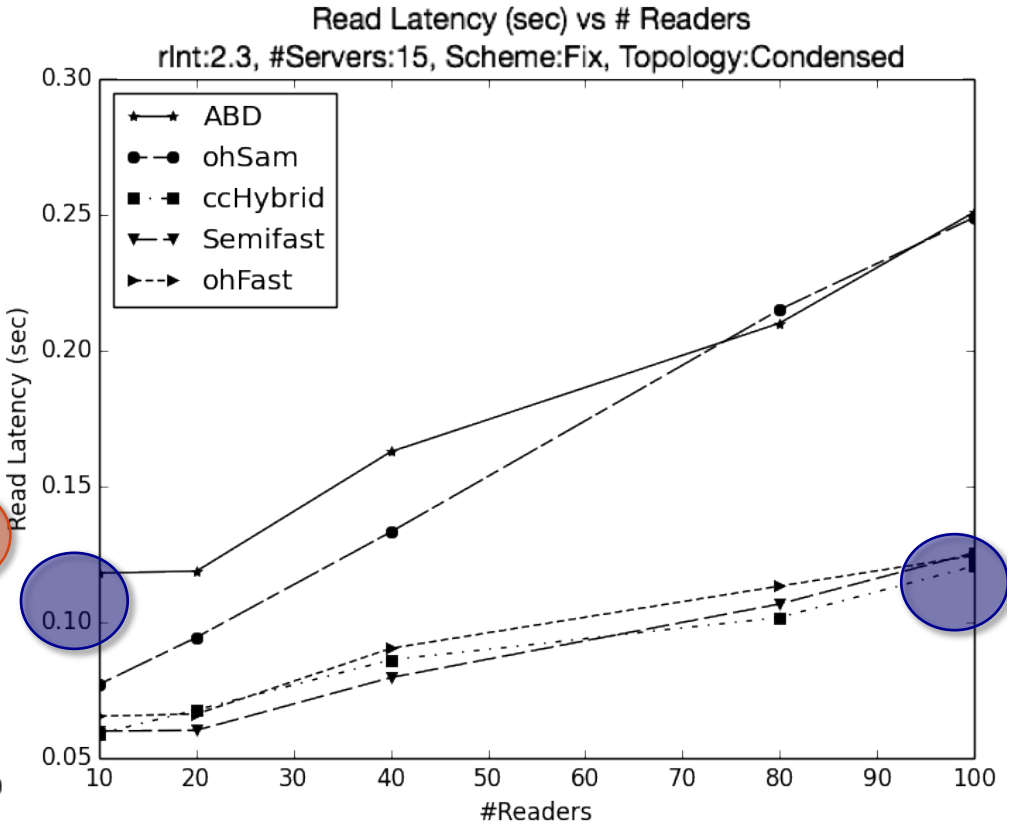




Operation Frequency: "multi-speed" algorithms are affected.

Empirical Results

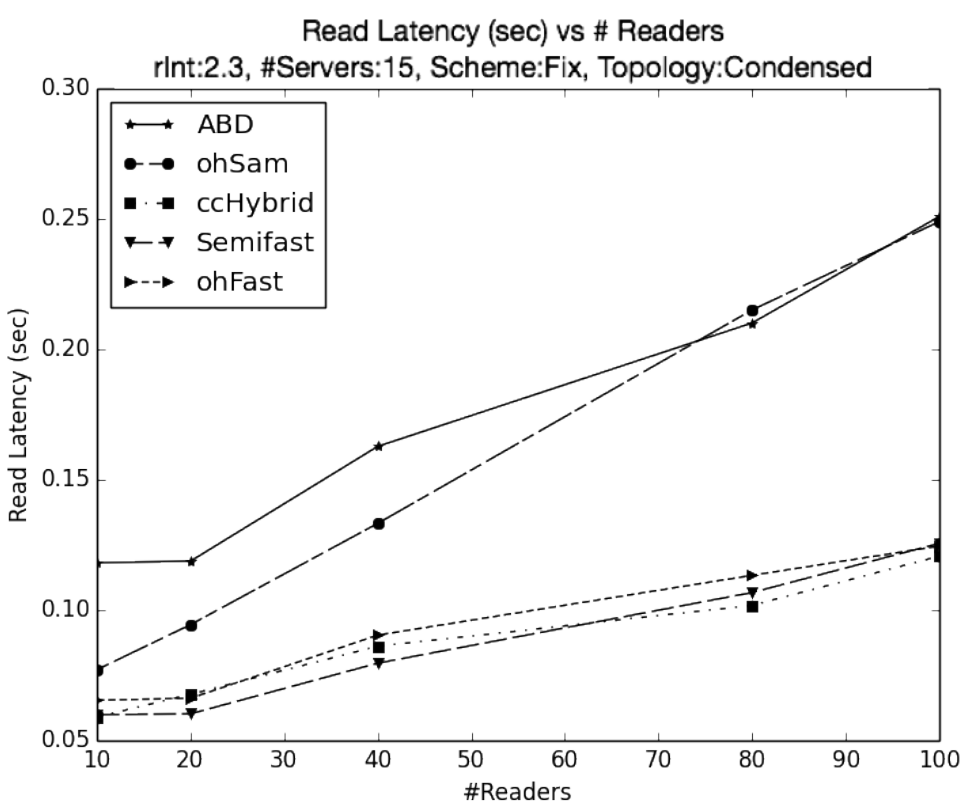


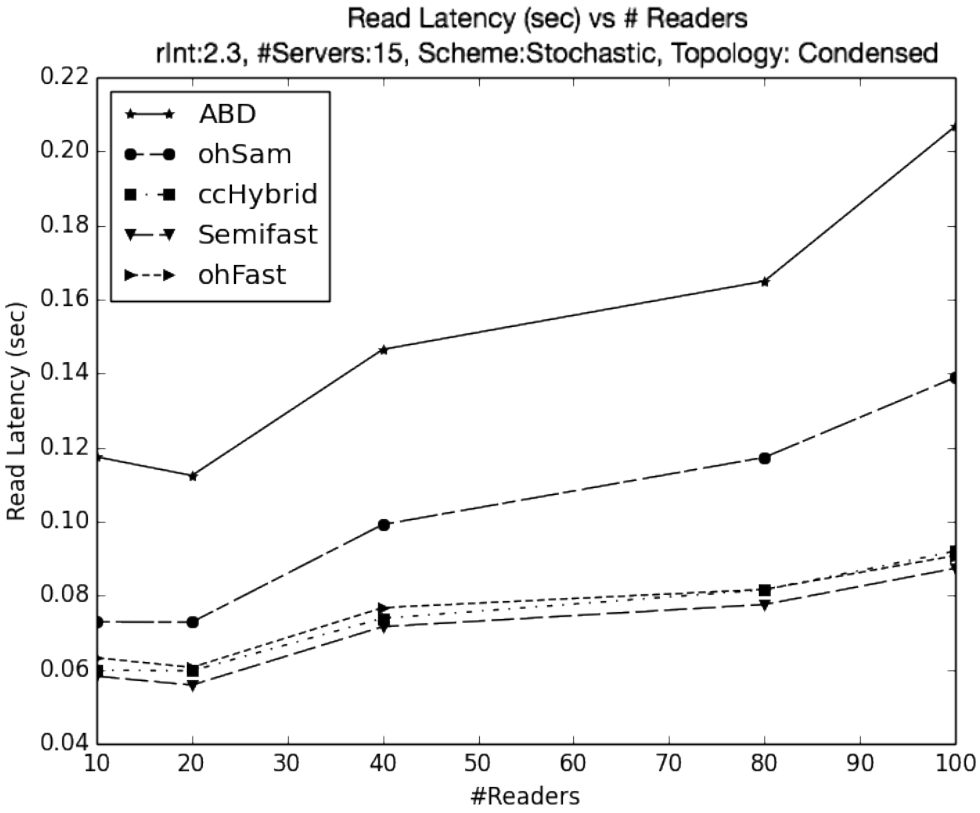


Operation Frequency: "multi-speed" algorithms are affected.

- They perform at least one slow operation per write.
- rInt=4.6 is closer to wInt=4.0 -> more concurrent reads to the write -> thus more slow reads.

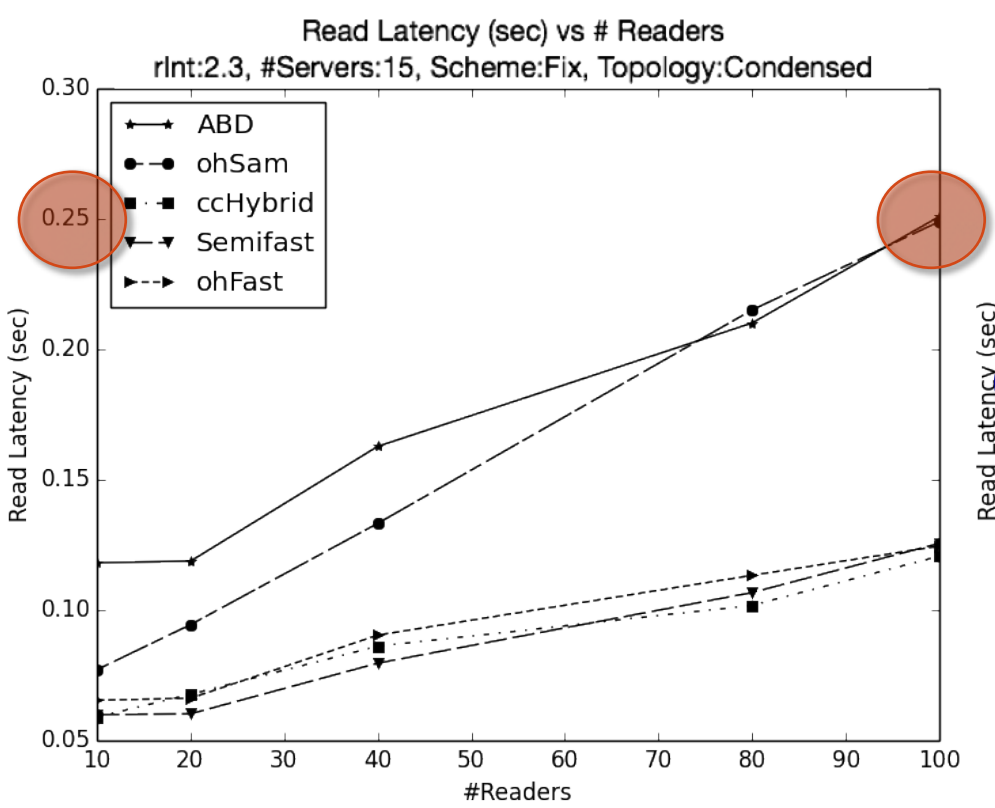
Empirical Results

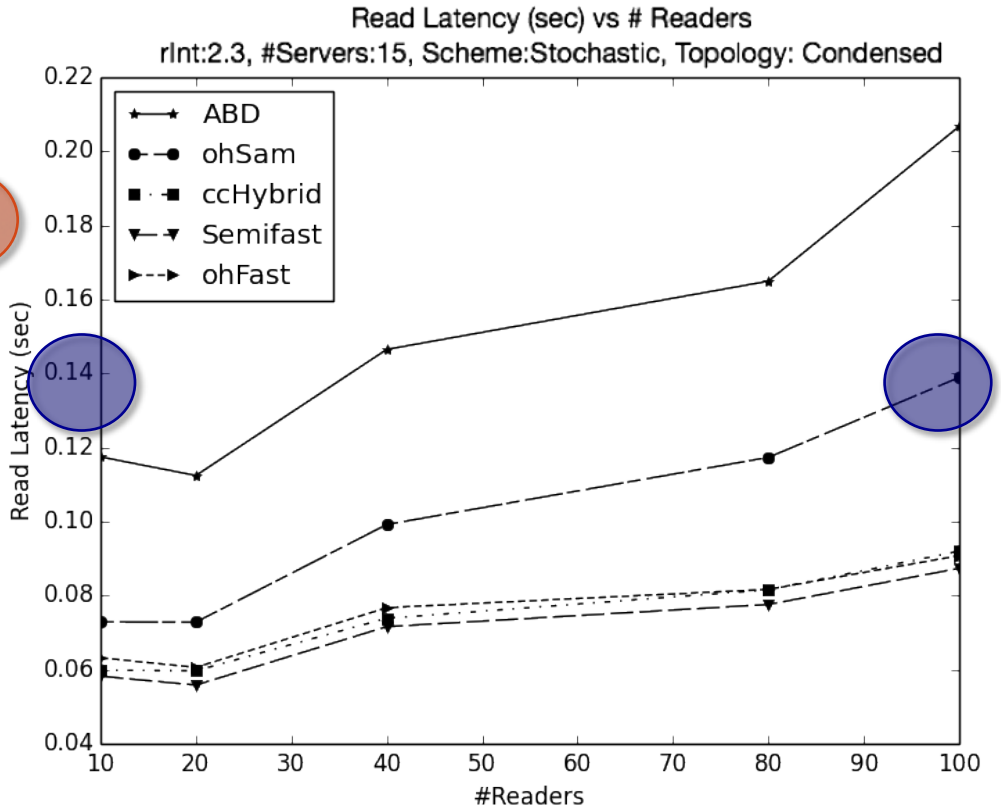




Concurrency Scheme: "single-speed" algorithms **ABD** and **OHSAM** are used as points of reference as they have same computation and communication in both schemes.

Empirical Results

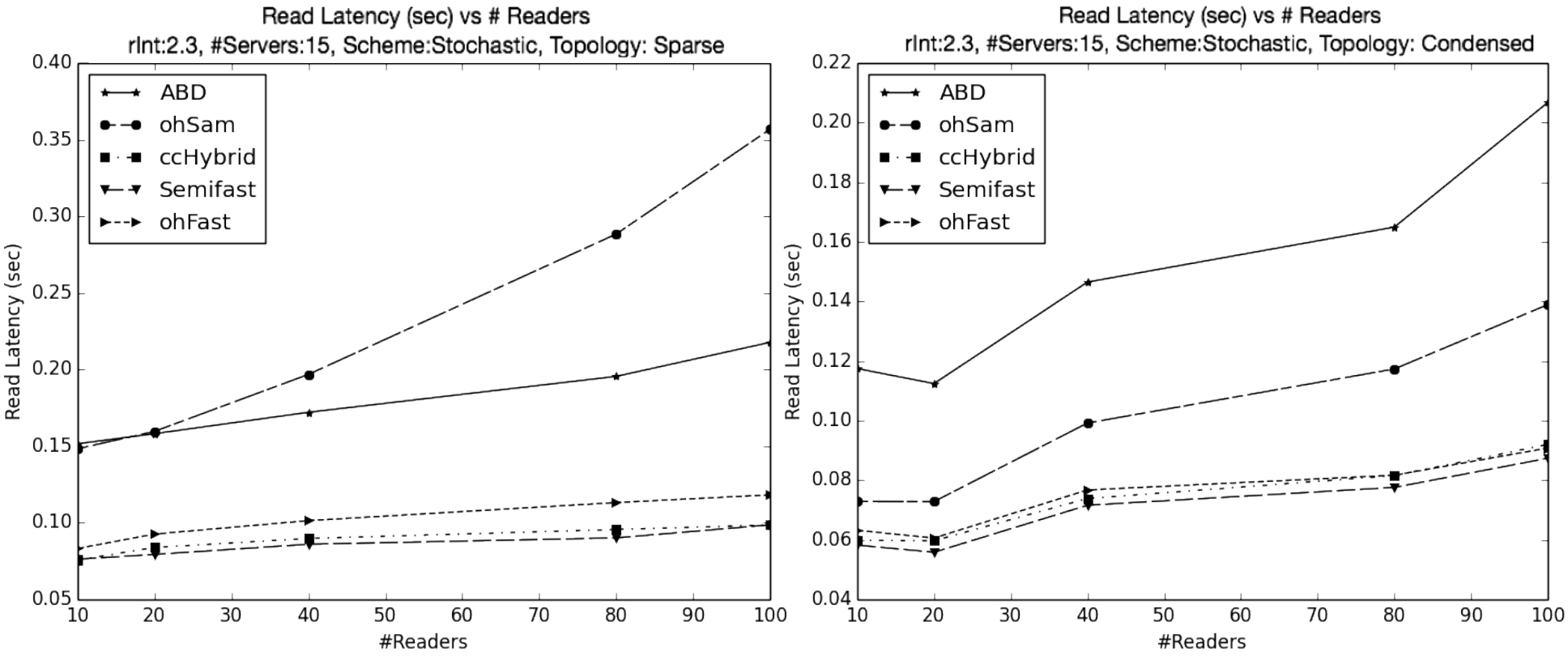




Concurrency Scheme:

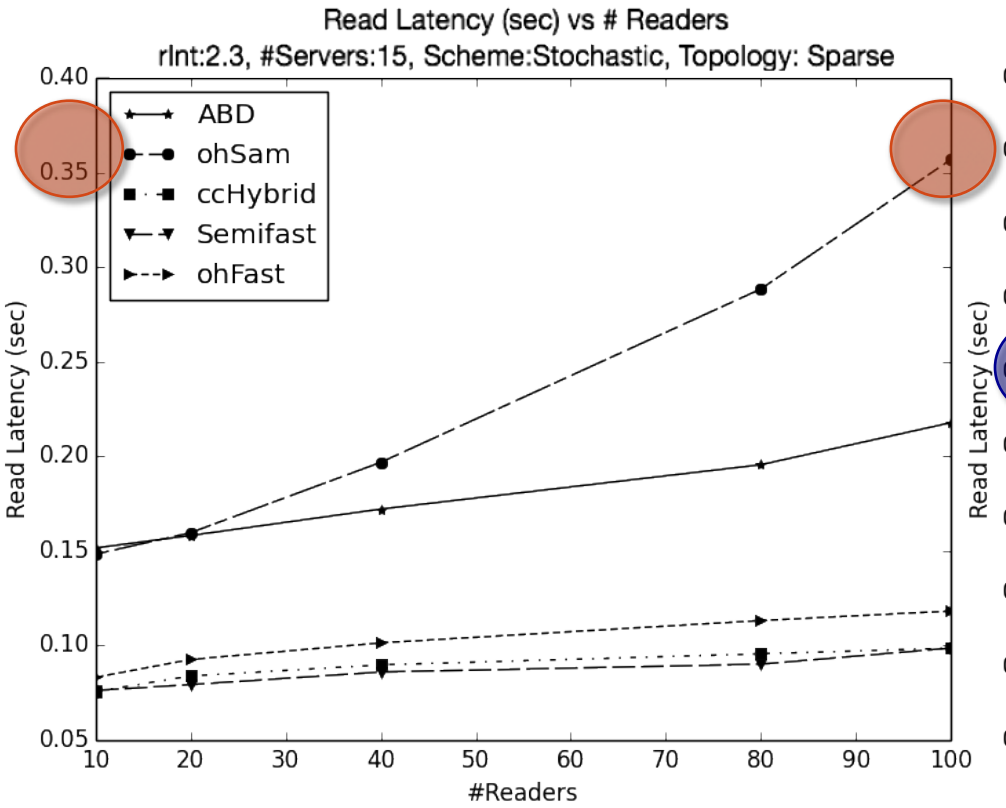
- Fix scheme introduces congestion in the network.
- Stochastic scheme -> distributes the invocation time intervals
  -> reducing the network congestion -> reducing operation latency.

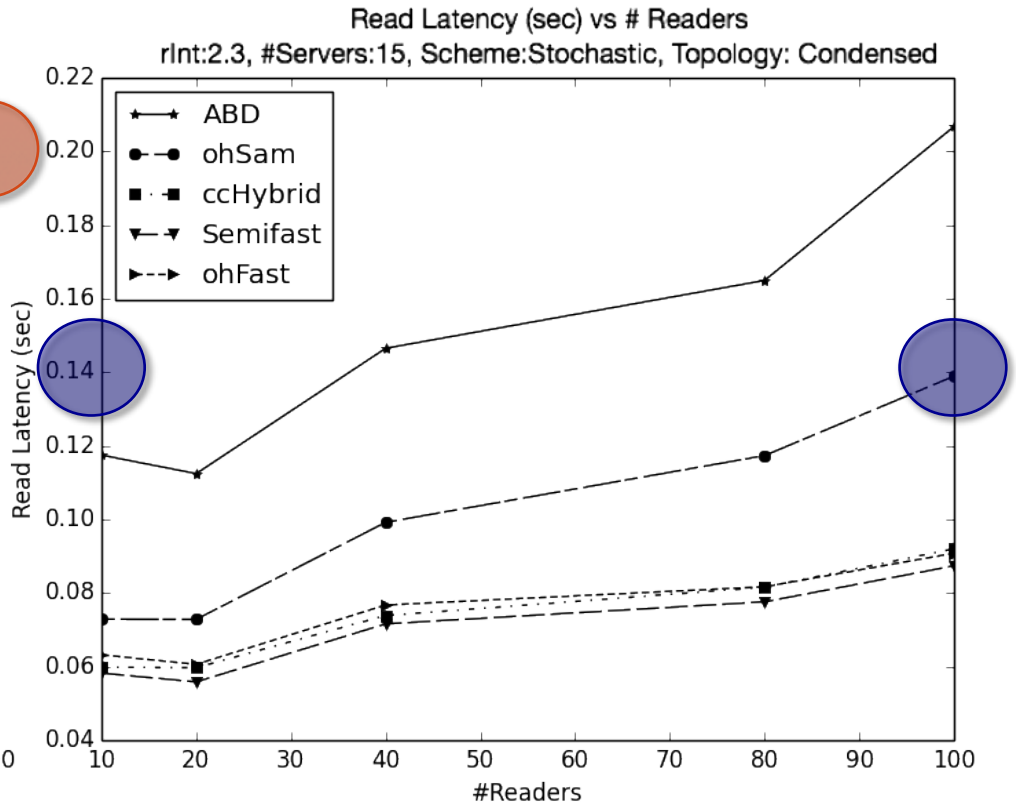
Empirical Results



Topology has an impact on the performance and the efficiency of all the algorithms. "Single-speed" algorithms **ABD** and **OHSAM** are affected the most.

Empirical Results





- Algorithms **OHSAM** and **OHFAST** perform much better in condensed topologies -> need to exchange messages between the servers during a relay phase.
- **OHFAST** outperforms **OHSAM** -> Do not perform operation relays in every read operation.

Conclusions

We presented,

- ✓ Algorithm **CCHYBRID** for the SWMR setting using ***one*** or ***two*** communication rounds
- ✓ Algorithm **OHFAST** for the SWMR setting using ***one*** or ***one-and-a-half*** communication rounds
- ✓ Simulation results show that both algorithms outperform all slow operation algorithms as well as "multi-speed" implementations that have high computation demands.

Neither algorithm imposes constrains on the number of readers and/or the writers.

Our developments take us closer to "***practical***" implementations of atomic read/write objects in the message passing environment.

Thank you!

Related Work

- Attiya, Bar Noy, and Dolev (96) provided the first algorithm – ABD – that implements single-writer/multiple-reader (SWMR) atomic objects.
  - Lynch and Shvartsman (97) presented extensions for MWMR memory, where both Read/Write operations take 4 communication exchanges.
- Dutta et al. (2004) give a SWMR implementation where each operation involves 1 communication round.
  - This is possible only when the number of readers r is bounded with respect to the number of servers s and the server failures f, $r < (s/f) - 2$.
- Fernández et al. (2015) has shown that that although the result of Dutta et. al (2004) is efficient in terms of communication, it requires processes to evaluate a computationally hard predicate (NP-Hard).
  - They proposed algorithm ccFast that allows operations to terminate with linear computation overhead but under the same participation constraints (Dutta 2004)

Related Work

- Georgiou et al. (2008) in order to achieve fast read operations, they introduced "quorum views" to examine the distribution of the latest value among the replicas.
- Georgiou et al. (2009) use the same predicate as Dutta et. al (2004) but on virtual nodes (i.e. sets of readers).
  - Both works trade communication for scalability.
- Hadjistasi et al. (2016) presented an algorithm, Oh-SAM, where each read operation takes one and a half rounds to complete.
  - No bounds assumed on the participation.
  - Negligible computation - algorithm relies on basic comparisons.
  - Algorithm is optimal in terms of communication when no constraints are imposed.

CONTRIBUTIONS

In this work we focus in improving the ***practicality*** of Single-Writer Multiple-Reader (SWMR) atomic read/write register algorithms.

- We seek low communication and computation costs.

In particular,

- Introduce a "multi-speed" algorithm, **CCHYBRID**, that allows operations to terminate in ***one*** or ***two*** communication rounds and does not impose any constraints on the participation.
- Combine techniques to obtain a "multi-speep" algorithm, **OHFAST**, that allows ***one*** and ***one-and-a-half*** round operations with unbounded participation.
- Complement the algorithms with experimental results.