Presentation by Alireza Tabatabaeian

A Resilient Hierarchical Checkpointing Algorithm for Distributed Systems Running on Cluster Federation

Table of contents

Introduction

- Today, cluster architectures are very widely spread in the research arena and in the industries.
- Resources can be volatile because the nodes can get disconnected when the connection is lost or hardware failure occurs
- Thus, the risks of occurrence of faults
- become very high; these faults would cause failures that prevent the correct execution of the distributed applications
- In distributed systems, fault-tolerance can be ensured by using checkpointing techniques

Terminologies

Terminologies

Distributed Systems

What?

A collection of independent computers (nodes) that appear to the user as a single system.

Key Properties:

concurrency, no global clock, and independent failures.

Example:

Cloud computing platforms like AWS or federated Kubernetes clusters.

Terminologies

Checkpointing

What?

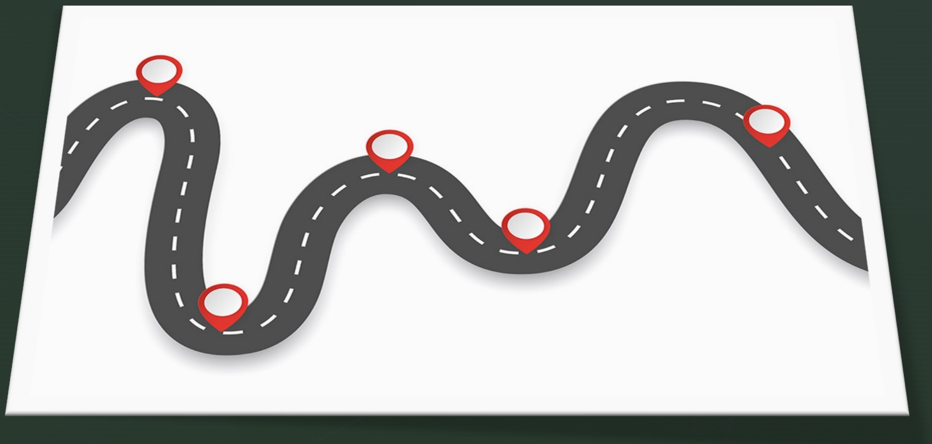
The act of **saving the current state** of a process or system to stable storage so that it can **resume from that point** in case of failure.

Why?

To reduce loss of progress after crashes.

Example:

Like saving a game — you reload from your last save if the game crashes.



Non-blocking Checkpointing
Recovery Line

Terminologies

Cluster

What?

A group of tightly connected computers (often in the same location or network).

How?

Nodes in a cluster share resources and often use **System Area Network (SAN)** to communicate efficiently.

Terminologies

Cluster Federation

What?

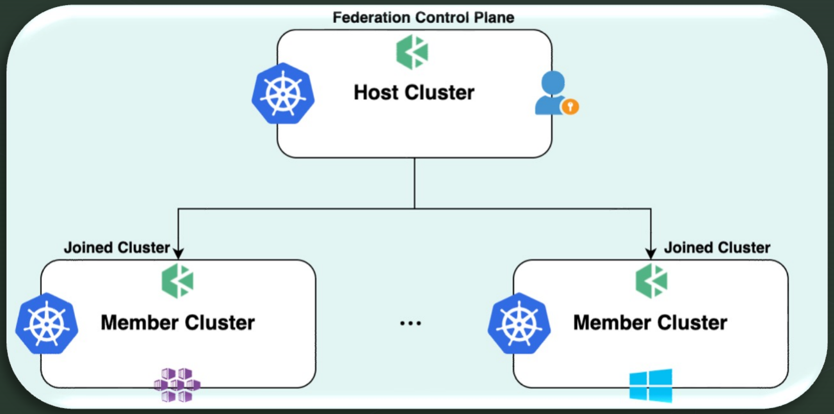
A **union of multiple clusters** connected over a **LAN or WAN**.

Use case:

Used to scale applications across data centers, geographical locations, or institutions.

Example:

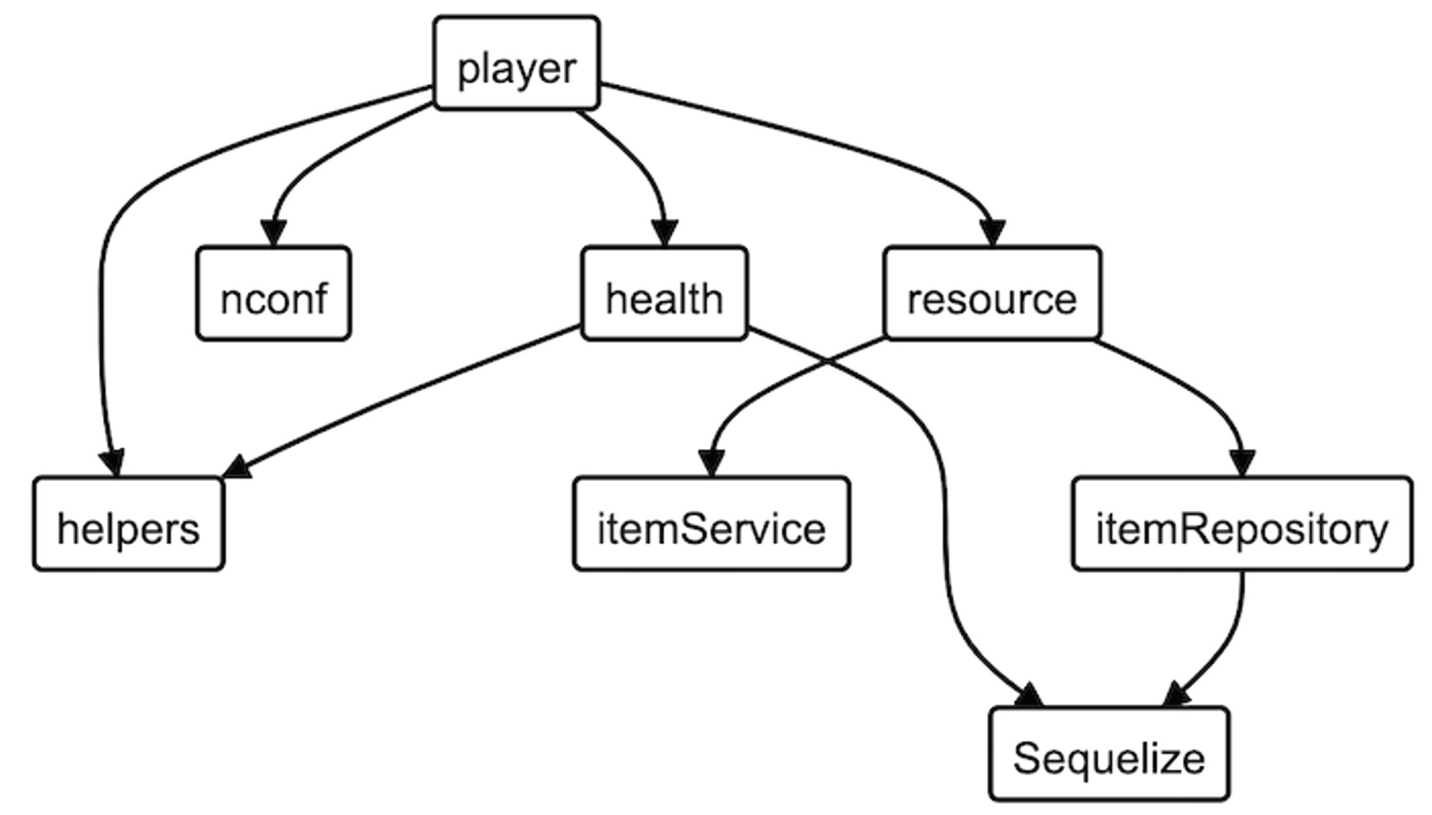
A research computing system where universities share compute clusters.



Fail-Stop Model

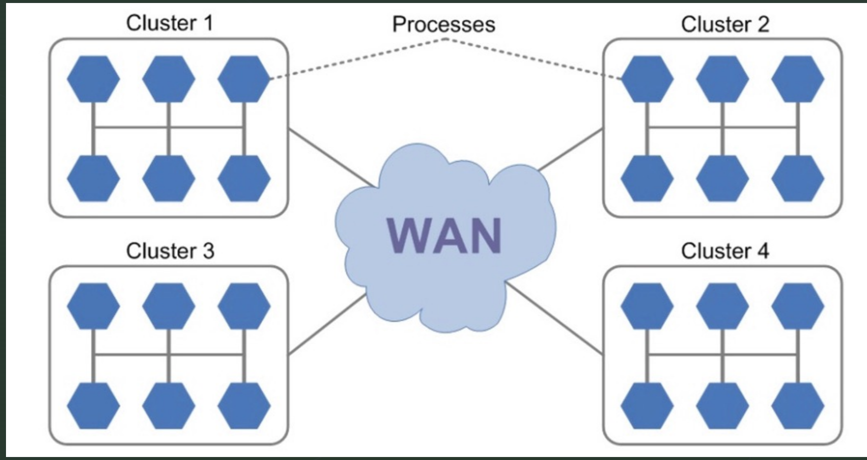
Terminologies

Dependency Graph



System Model

- **Formula**: N = M*K
- **Communications**: SAN (Intra) → LAN/WAN (Inter)
- **Assumptions**: Fail-Stop → No shared memory/clock → Reliable message passing → Any process can initiate checkpointing



Related Works

Related Works

Centralized and Mirroring

Central File Server Checkpointing

- Every process saves its state to a central server.
- **Pros:** Simple and easy to manage.
- **Cons:** Scalability bottleneck and single point of failure.

Checkpoint Mirroring

- Each process duplicates its checkpoint on another node (mirror).
- **Pros:** Fast recovery if original node fails.
- **Cons:** Higher resource usage (bandwidth, storage); doesn’t scale well.

**Both methods are useful in small systems but problematic in large federated environments.*

Related Works

Skewed Checkpointing

- Processes take checkpoints at different times to reduce contention and storage load.
- **Evaluation model:** Stochastic analysis of recovery overhead.
- **Result:** Skewed checkpointing was found to be more efficient than centralized or mirrored methods.

*Not all processes need to checkpoint at the same time.

Related Works

Sender-Based Message Logging

Problem addressed

Lost or orphan messages in **inter-cluster recovery**.

How it works

Sender logs every sent message.

Drawbacks

Logging every message causes overhead.

Related Works

Non-blocking coordinated checkpointing

How it works

Clusters take coordinated checkpoints without stopping execution.

Goal

Avoid domino effect and unnecessary rollbacks.

Key Benefit

Even concurrent failures can be recovered without redoing everything.

Related Works

Efficient Recovery Algorithm

Key Feature

Independent of specific cluster federation architecture.

Limitation

Message logging and checkpointing handled separately → higher complexity.

Related Works

Low-Cost Non-Blocking algorithms

Focus

Minimize messages and storage cost

Strategies

- Checkpoint fewest number of processes
- Each cluster manages its own metadata (decentralized)
- Bounded intervals to avoid domino effect

Problem

not yet integrated with optimistic logging.

Related Works

Wrap Up

Term	Meaning
Checkpointing technique	A general strategy or method used to save and recover program state in a distributed system.
Checkpointing algorithm	A specific, formalized step-by-step procedure to implement a technique (e.g., with pseudocode, message handling rules, data structures).

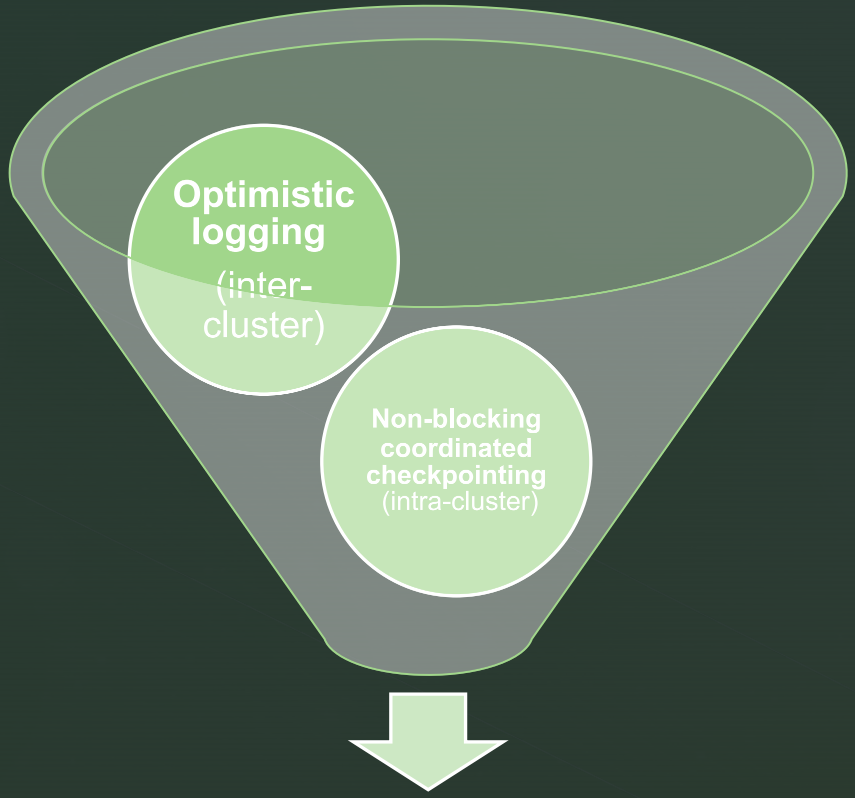
Related Works

Wrap Up

Name/Category	Type	Problem Addressed
Central file server / Mirroring / Skewed	Techniques	How to store checkpoints efficiently (but limited scalability).
Sender-based logging	Algorithm	How to recover from lost **inter-cluster messages**.
Non-blocking coordinated checkpointing	Algorithm	How to avoid blocking & domino effect in recovery.
Hybrid checkpointing (coordinated + comm-induced)	Technique + Algorithm	Handle tightly-coupled applications across clusters.
Efficient architecture-independent recovery	Algorithm	Fast recovery without depending on network layout.
Low-cost non-blocking checkpointing	Algorithm	Reduce number of messages and storage operations.
Domino-effect free concurrent recovery	Algorithm	Crash recovery for concurrent failures in federated clusters.

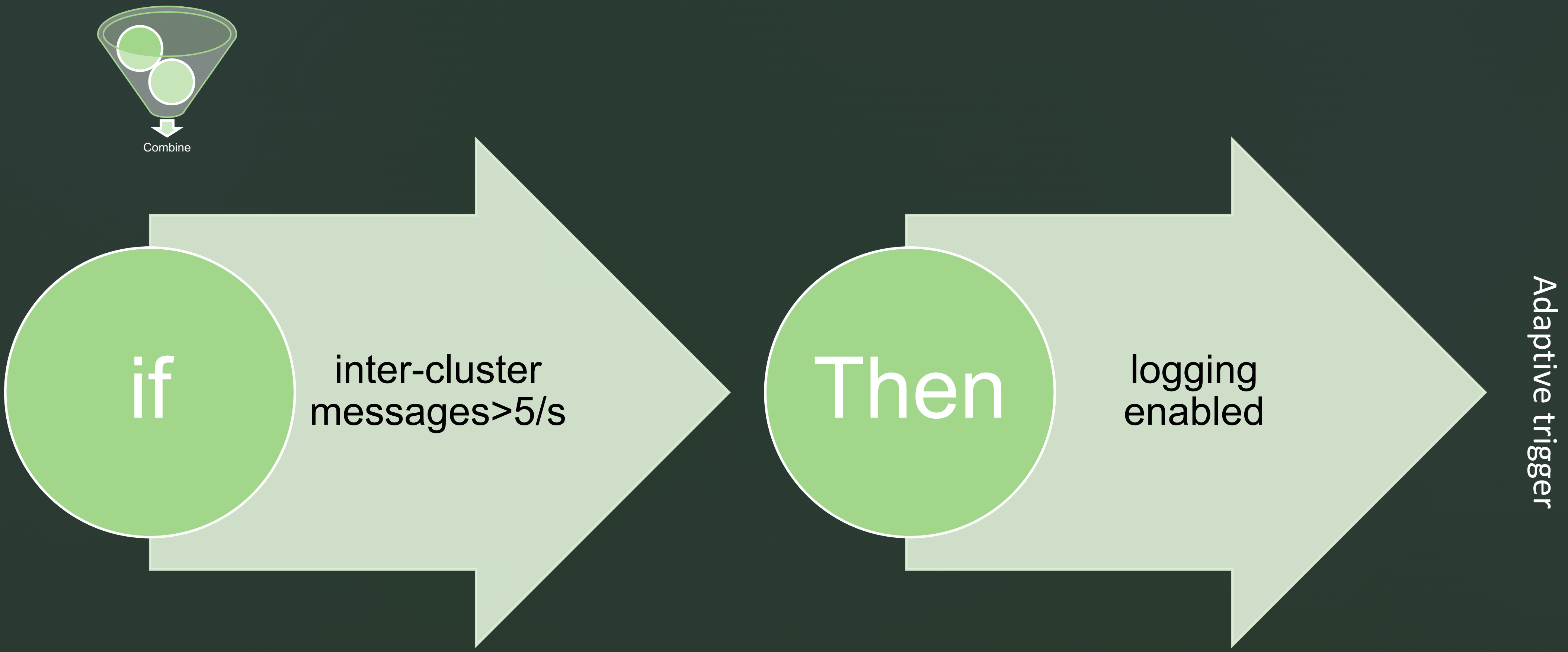
Proposed Algorithm

Proposed Algorithm

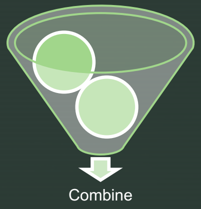


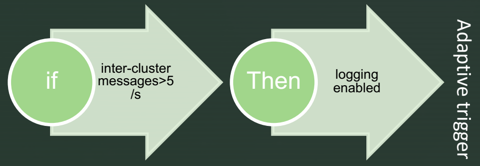
Combine

Proposed Algorithm



Proposed Algorithm

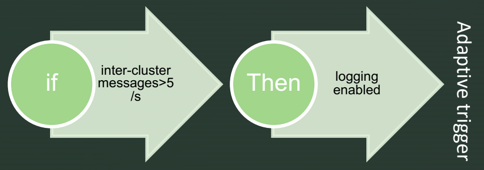


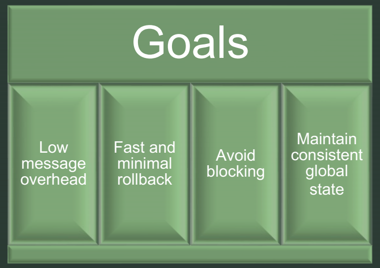


Goals

- Low message overhead
- Fast and minimal rollback
- Avoid blocking
- Maintain consistent global state

Proposed Algorithm





“Clusters independently take notes (checkpoints) while message messengers (between clusters) keep logs only if the chatter becomes too noisy.”

Checkpointing – Algorithm Mechanics

Algorithm Part 1: *Checkpointing Process*

Part executed always by every process on each cluster

Upon receiving a message Msg:
if $(C_{Receiver} \neq C_{Sender})$
Save *Msg to* $InpMsg_{Receiver}$;
Save *Determinant to* $DetMsg_{Receiver}$;
else
if $(Ckpt_{Receiver} < Ckpt_{Sender})$
Save $(TempCkpt_i)$;
$Ckpt_{Receiver} \leftarrow Ckpt_{Sender}$;
endif
endif

Part executed simultaneously on each cluster every 180 s

Part executed by the initiator process Pi
Save $(TempCkpt_i)$;
$Ckpt_i := Ckpt_i + 1$;
for All $(P_x \in DepProc_i)$
Send *Checkpoint Request* $(Ckpt_i, t)$;
endfor
Upon receiving : Answer Request (t)
$Term_i := Term_i + t$;
if $(Term_i = 1)$
for All $(P_x \,/\, x \in [1..k])$
Send *Conformation Request ();*
endfor
endif

Part executed by every process Pj in the cluster
Upon receiving : Checkpoint Request $(Ckpt_x, t)$
if $(Ckpt_j < Ckpt_x)$
Save $(TempCkpt_j)$;
$Ckpt_j := Ckpt_j + 1$;
if $(DepProc_j = \varnothing)$
Send *Answer Request (t);*
else
for All $(P_x \in DepProc_j)$
Send *Checkpoint Request* $(Ckpt_j, t)$;
endfor
endif
endif
Upon receiving : Conformation Request ()
$PermCkpt_j \leftarrow TempCkpt_j$;
$InpMsg_j \leftarrow \varnothing$;
$DetMsg_j \leftarrow \varnothing$;
$Term_j \leftarrow 0$;

Fig. 2. Algorithm part 1.

Checkpointing – Algorithm Mechanics

```
Algorithm Part 1: Checkpointing Process
Part executed always by every process on each cluster
  Upon receiving a message Msg:
    if (C_Receiver ≠ C_Sender)
      Save Msg to InpMsg_Receiver;
      Save Determinant to DetMsg_Receiver;
    else
      if (Ckpt_Receiver < Ckpt_Sender)
        Save (TempCkpt_i);
        Ckpt_Receiver ← Ckpt_Sender;
      endif
    endif
Part executed simultaneously on each cluster every 180 s
Part executed by the initiator process Pi
    Save (TempCkpt_i);
    Ckpt_i := Ckpt_i + 1;
    for All (P_x ∈ DepProc_i)
      Send Checkpoint Request (Ckpt_i, t);
    endfor
    Upon receiving : Answer Request (t)
    Term_i := Term_i + t;
    if (Term_i = 1)
      for All (P_x / x ∈ [1..k])
        Send Conformation Request ();
      endfor
    endif
Part executed by every process Pj in the cluster
    Upon receiving : Checkpoint Request (Ckpt_x, t)
    if (Ckpt_j < Ckpt_x)
      Save (TempCkpt_j);
      Ckpt_j := Ckpt_j + 1;
      if (DepProc_j = ∅)
        Send Answer Request (t);
      else
        for All (P_x ∈ DepProc_j)
          Send Checkpoint Request (Ckpt_j, t);
        endfor
      endif
    endif
    Upon receiving : Conformation Request ()
    PermCkpt_j ← TempCkpt_j;
    InpMsg_j ← ∅;
    DetMsg_j ← ∅;
    Term_j ← 0;
```

Fig. 2. Algorithm part 1.

```
Algorithm Part 2: Recovery Process
Part executed by the faulty Process
    for All (P_x / x ∈ [1..n])
      Send Recovery Request ();
    endfor
Part executed by every process Pj in the system
  Upon receiving a Recovery Request ()
  if (C_Receiver = C_Sender)
    Resume execution at the last recorded checkpoint PermCkpt_j;
  else
    Replay the reception event of Msg ∈ InpMsg_j based on DetMsg_j;
  endif
```

Fig. 3. Algorithm part 2.

Recovery Implementation

Upon failures

All processes in the affected **cluster** roll back to last **permanent checkpoint**	Other clusters **replay inter-cluster messages** using determinants from DetMsg

Determinants

{SeqNum, SendTime, ReceiveTime}

Recovery Ensures

No domino effect	Consistent **global state**	Minimum re-execution

Only local data + inter-cluster logs needed → Efficient!

Performance Comparison

Performance Comparison

Metric	Proposed alg	others
Architecture dependant	❌	✅(Mostly)
Domino effect	❌	✅(Often)
Handles concurrent failures	✅	❌ (Often)
Blocking	❌	❌✅
Message complexity	$O(n)$	$O(kn)$, $O(n^2)$, etc
Stable storage trips	✅ Minimal (k)	❌ Higher (k + r)

Conclusion: Efficient and scalable, especially in federated setups.

Simulation Results

Simulation Results

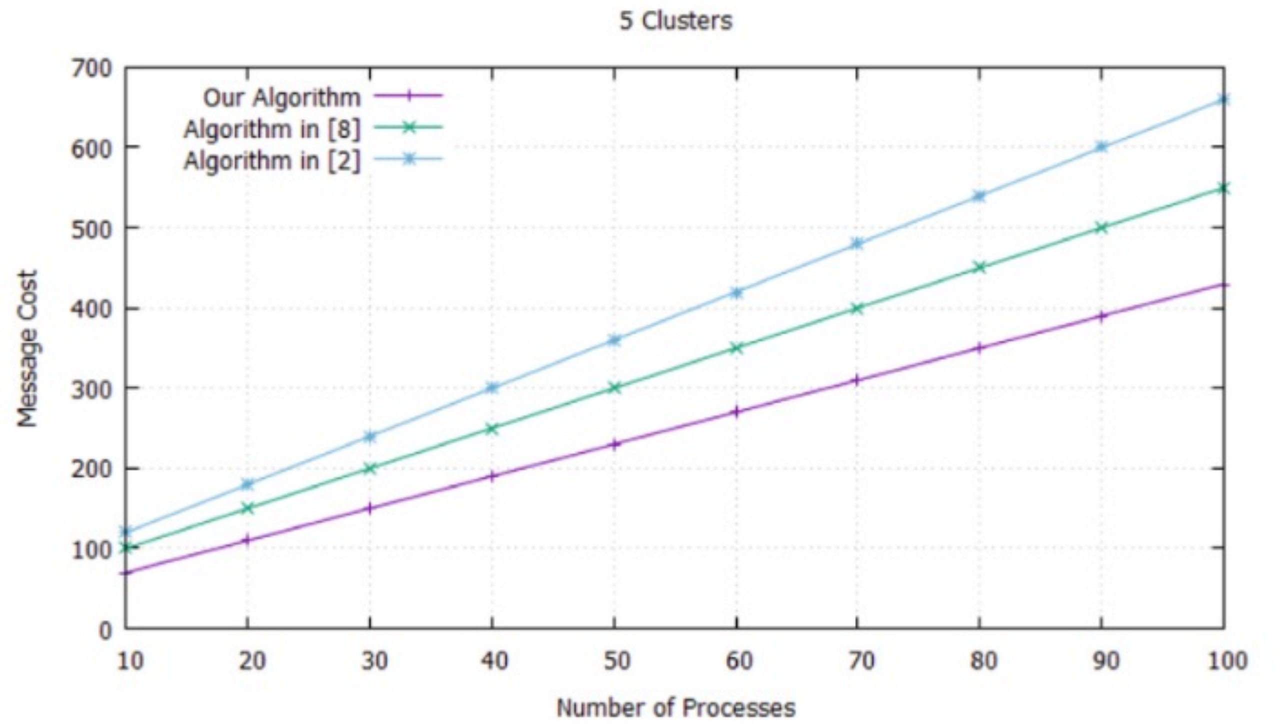
Simulator: **ChkSim**

Clusters tested: 5, 10, 20

Simulation Results

Simulator: **ChkSim**

Clusters tested: 5, 10, 20

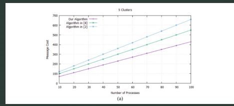


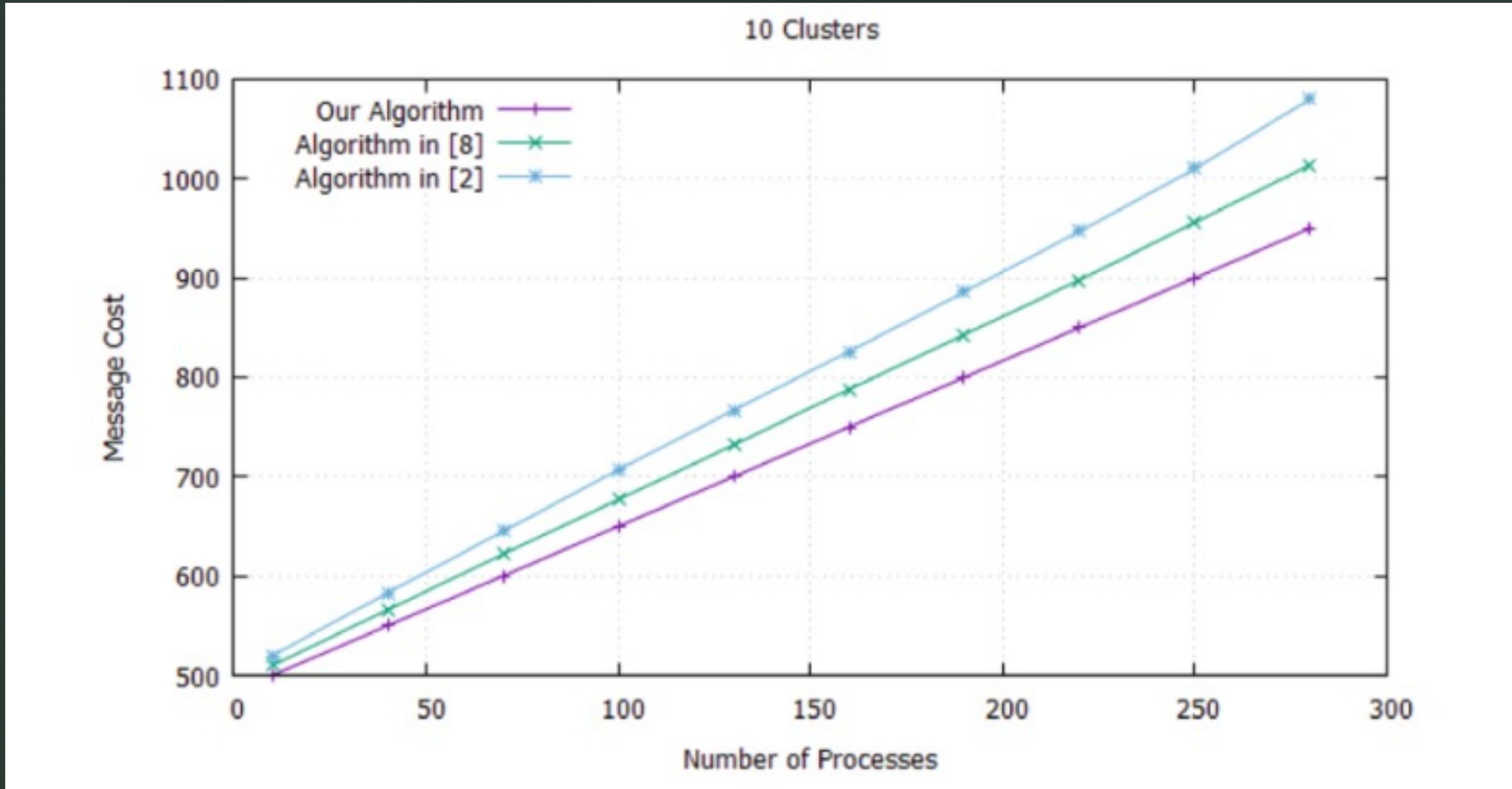
(a)

Simulation Results

Simulator: **ChkSim**

Clusters tested: 5, 10, 20



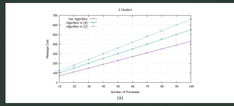


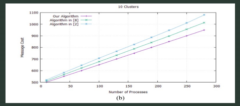
(b)

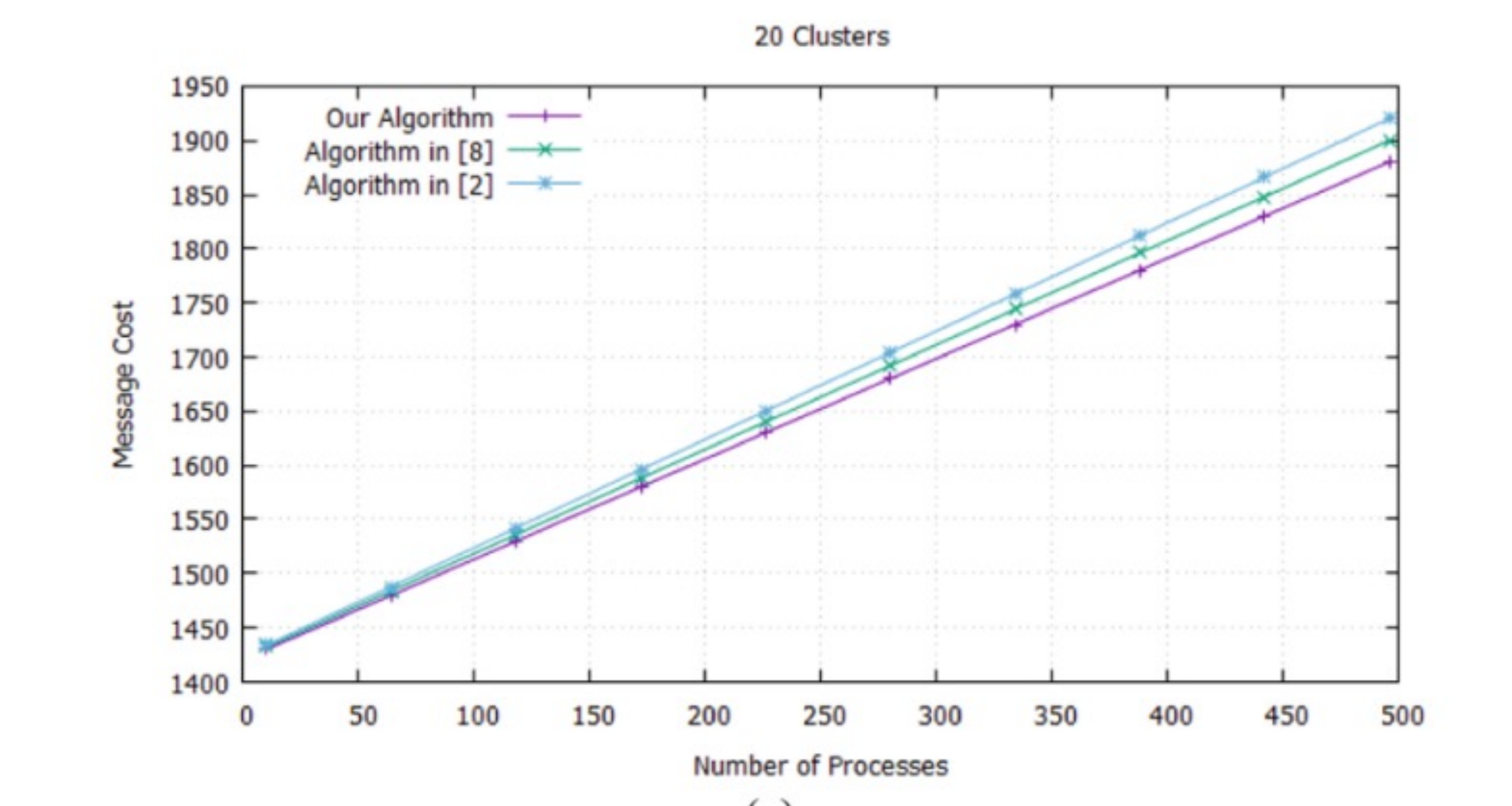
Simulation Results

Simulator: **ChkSim**

Clusters tested: 5, 10, 20





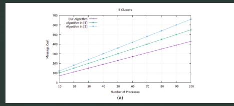


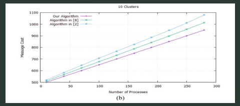
(c)

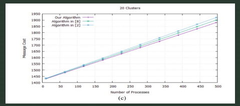
Simulation Results

Simulator: **ChkSim**

Clusters tested: 5, 10, 20







consistently **uses fewer messages**

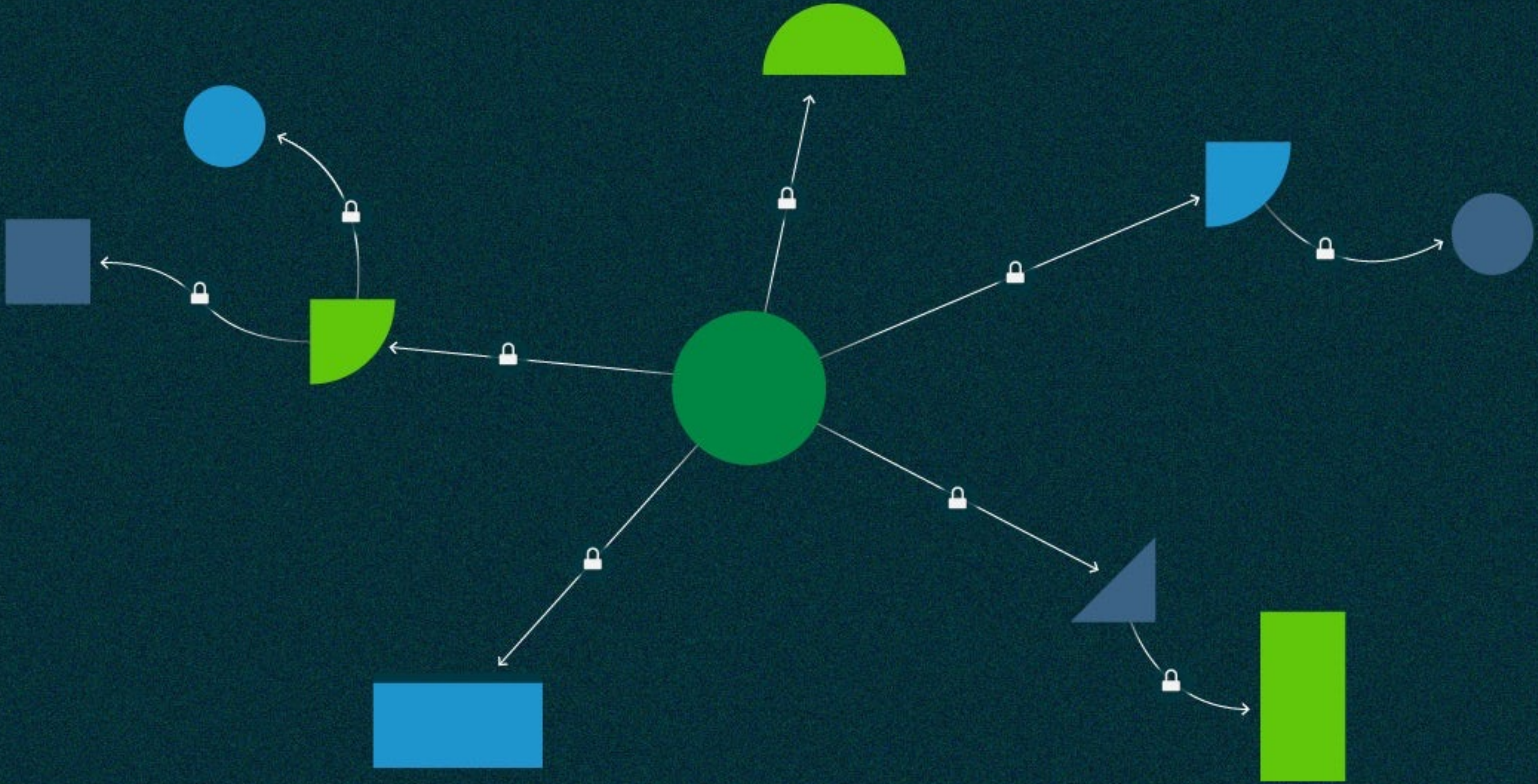
efficient when: Large number of processes/Clusters

Conclusion

Conclusion

- A **resilient hierarchical checkpointing algorithm** for federated clusters
- Combines best of coordinated and logging-based recovery
- **Avoids blocking**, **low message overhead**, and **domino-effect free**
- Suitable for large-scale, fault-prone systems

Thanks



Presented by Alireza Tabatabaeian