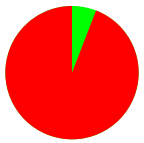
Adaptive Logging:

Optimizing Logging & Recovery Costs in Distributed In-memory Databases

Authors: Chang Yao, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, & Sai Wu

Presented By: Shreejit Nair

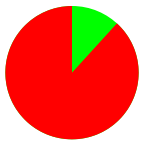


Introduction

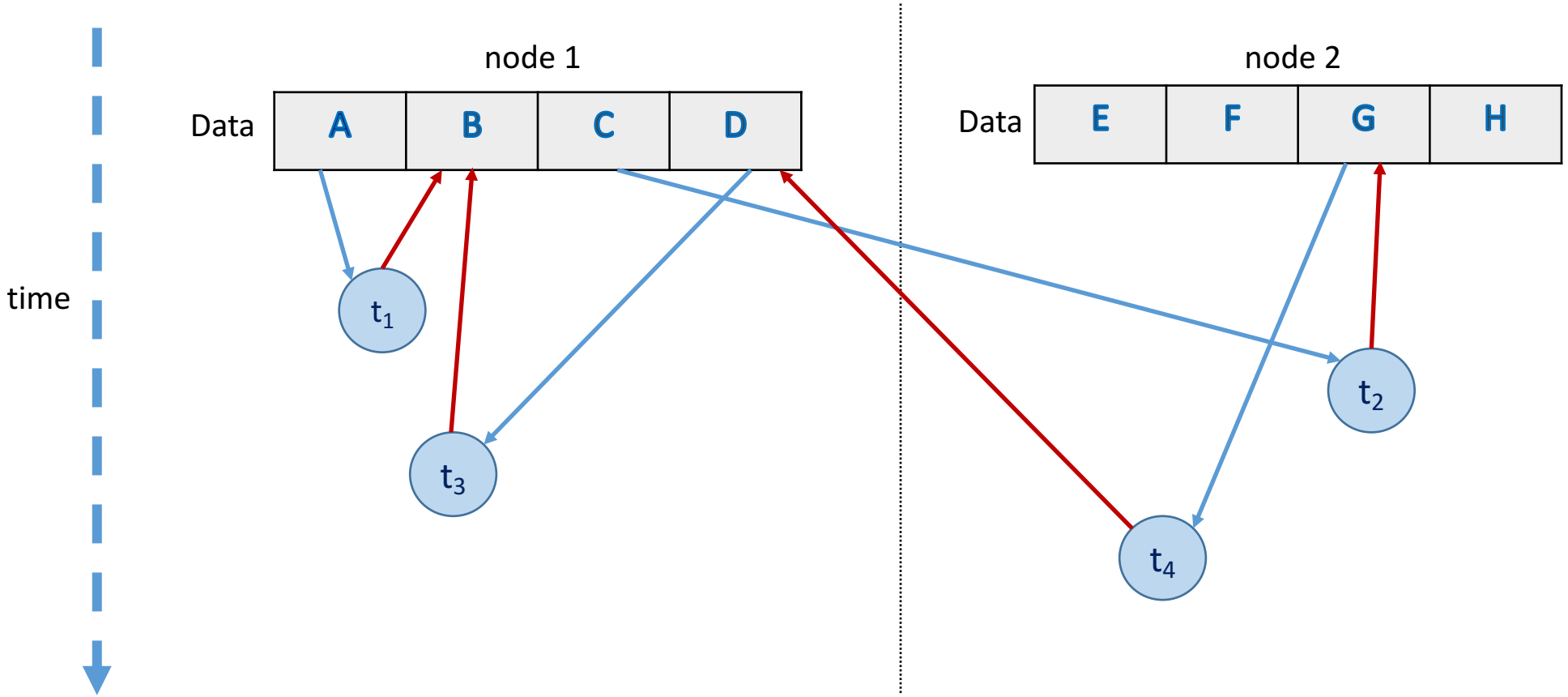
- ARIES Logging: A traditional "heavy-weight" logging technique that involves recording old and new updates to a tuple. Involves significant overhead in writing the log out to disk.

- Command Logging: A more recent approach that maintains only a pointer to the transaction with input parameters giving a compact log size. Involves overhead in replaying the entire transaction back from the log file.

- **Adaptive Logging**: The paper proposes a technique that uses a combination of ARIES style and command logging giving 10x boost to recovery times and transaction throughput comparable to Command logging.



Logging Comparison



f(x , y): y = 2x

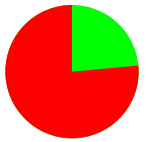
Logging Comparison (Contd)

Table 1: ARIES log

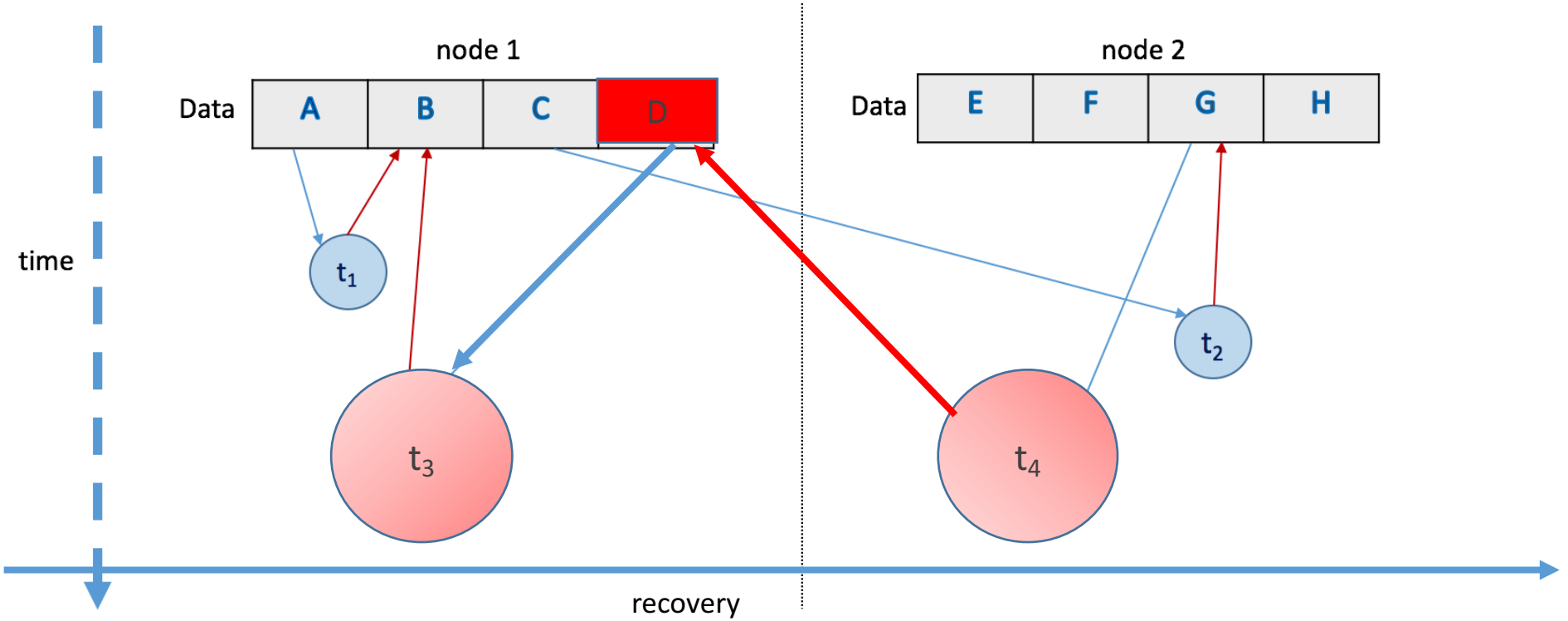
timestamp	transaction ID	parameter	old value	new value
100001	t_1	B	v(B)	2v(A)
100002	t_2	G	v(G)	2v(C)
100003	t_3	B	v(B)	2v(D)
100004	t_4	D	v(D)	2v(G)

Table 2: Command log

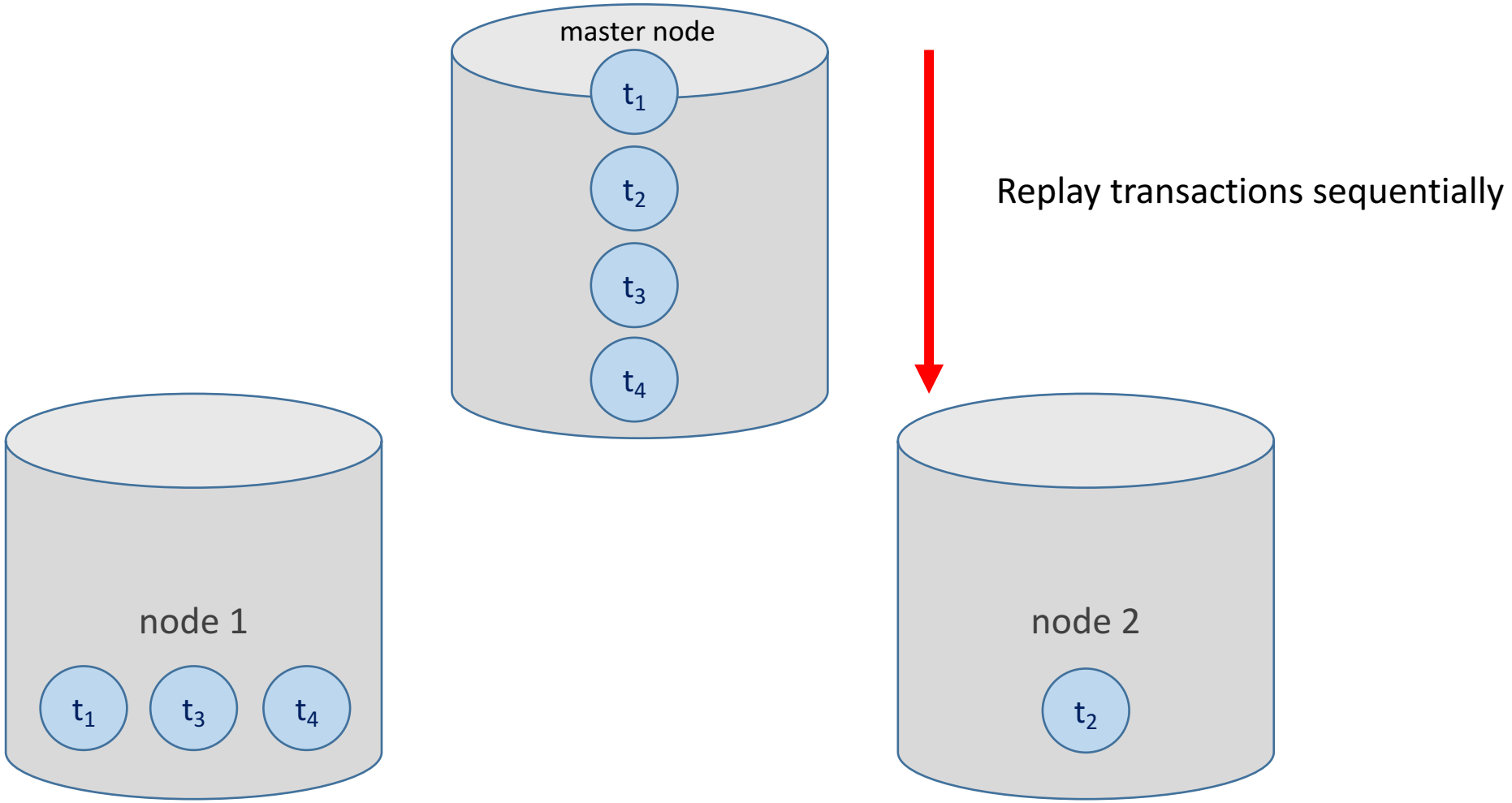
transaction ID	timestamp	stored procedure pointer	parameters
1	100001	$\mathcal{p}$	A,B
2	100002	$\mathcal{p}$	C,G
3	100003	$\mathcal{p}$	D,B
4	100004	$\mathcal{p}$	G,D



Problems with Command Logging

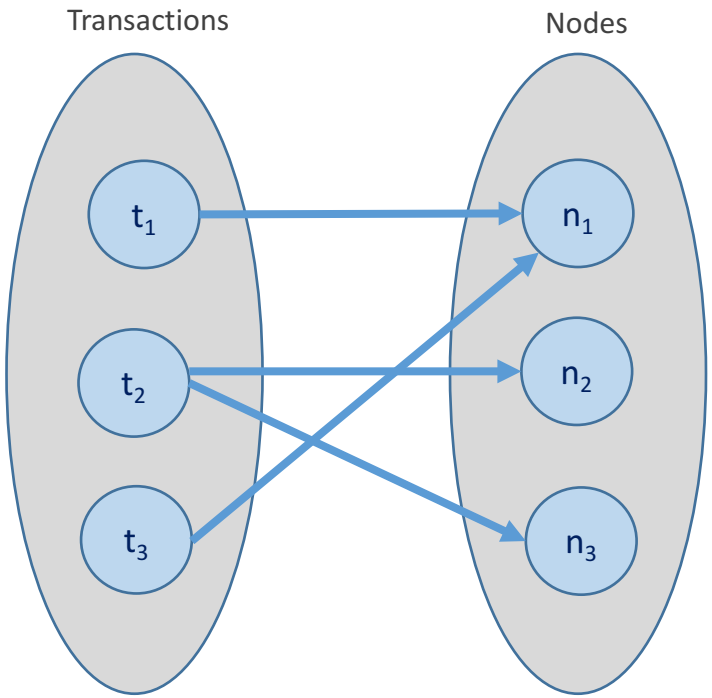


VoltDB Command Logging Implementation



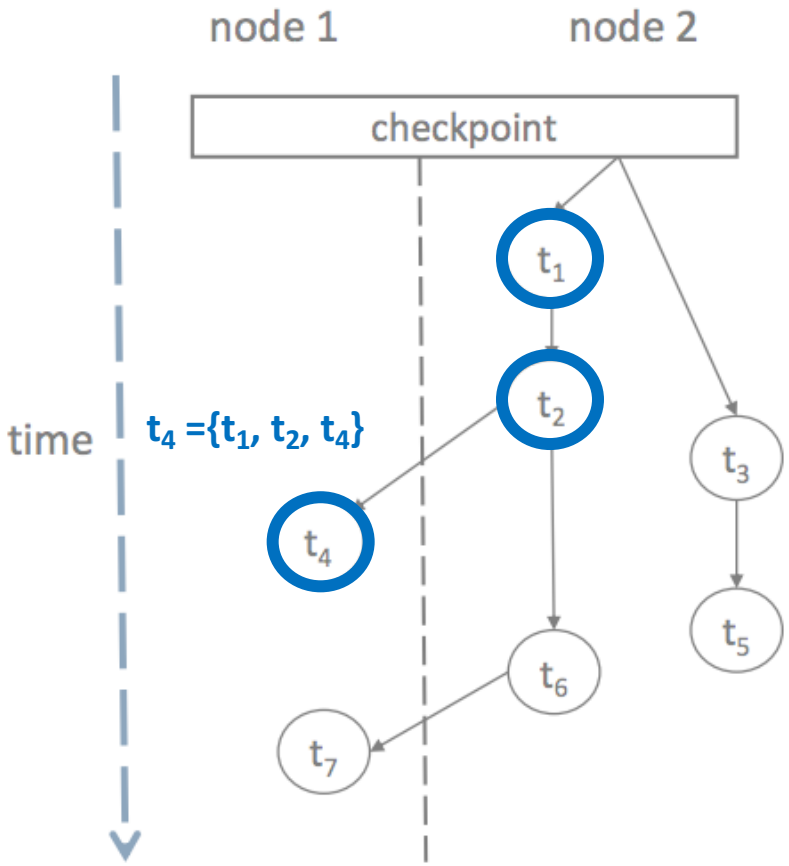
Distributed Command Logging

- Initiate recovery in parallel & replay necessary transactions.

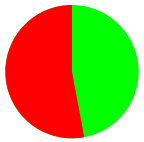


$f(t_i) = n_x$

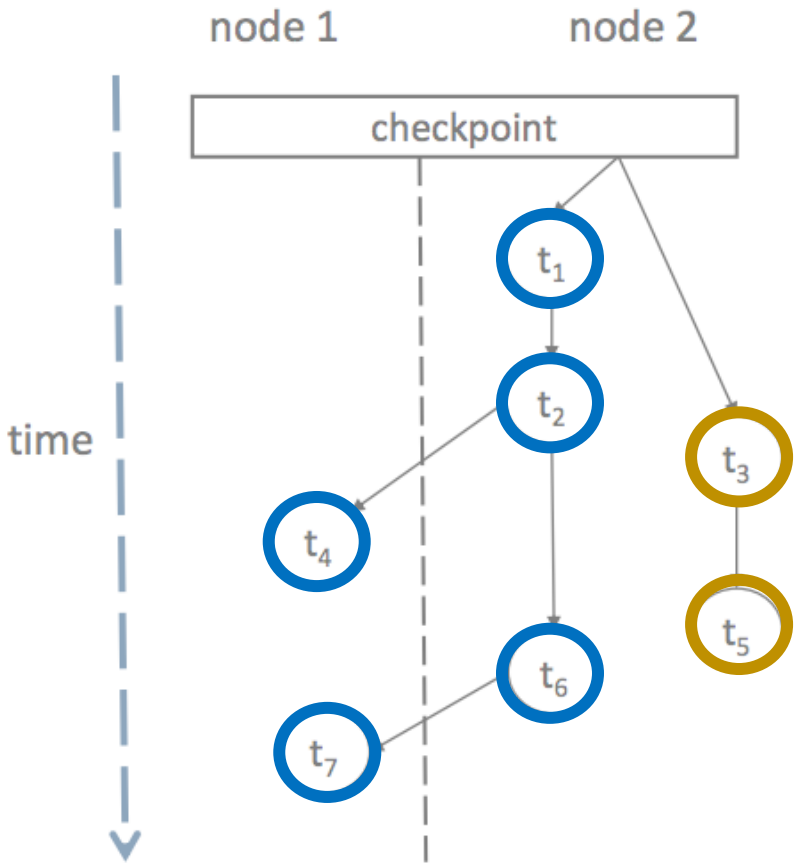
Dependency Graph for Recovery Set (n_x)



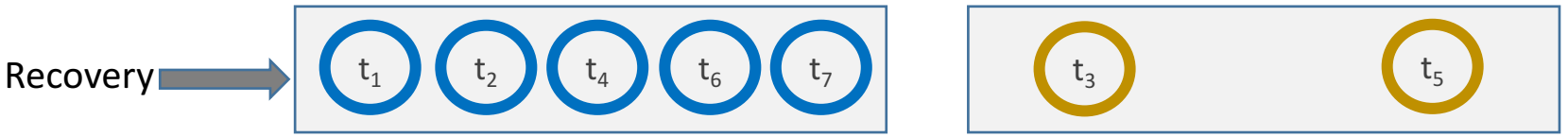
txn	submission time	commit time	stored procedure	parameters
1	1	2	$f_1(p_1)$: $p_1=p_1+1$	x_1
2	2	4	$f_2(p_1,p_2)$: $p_1=p_1-1$, $p_2=p_1+2$	x_1, x_3
3	3	4	$f_3(p_1)$: $p_1=2*p_1$	x_4
4	4	5	$f_4(p_1)$: $p_1=\log(p_1)$	x_3
5	4	5	$f_5(p_1, p_2)$: $p_1=2*p_2$	x_4, x_5
6	5	6	$f_5(p_1, p_2)$: $p_1=2*p_2$	x_6, x_1
7	6	7	$f_1(p_1)$: $p_1=p_1+1$	x_6



Processing Group (Parallel Recovery)

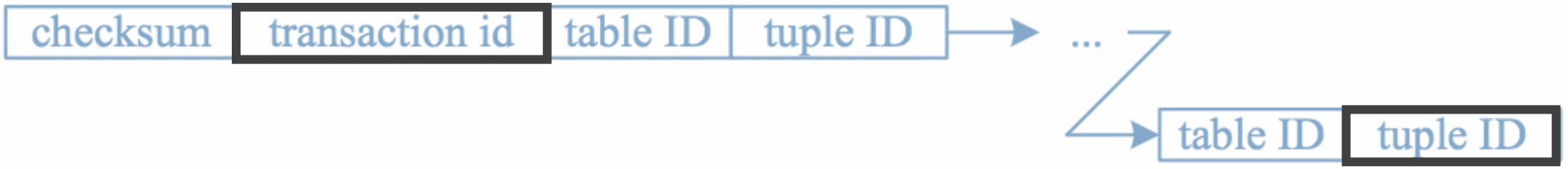


txn	submission time	commit time	stored procedure	parameters
1	1	2	$f_1(p_1)$: $p_1=p_1+1$	x_1
2	2	4	$f_2(p_1,p_2)$: $p_1=p_1-1$, $p_2=p_1+2$	x_1,x_3
3	3	4	$f_3(p_1)$: $p_1=2*p_1$	x_4
4	4	5	$f_4(p_1)$: $p_1=\log(p_1)$	x_3
5	4	5	$f_5(p_1, p_2)$: $p_1=2*p_2$	x_4, x_5
6	5	6	$f_5(p_1, p_2)$: $p_1=2*p_2$	x_6, x_1
7	6	7	$f_1(p_1)$: $p_1=p_1+1$	x_6



Transaction Footprint

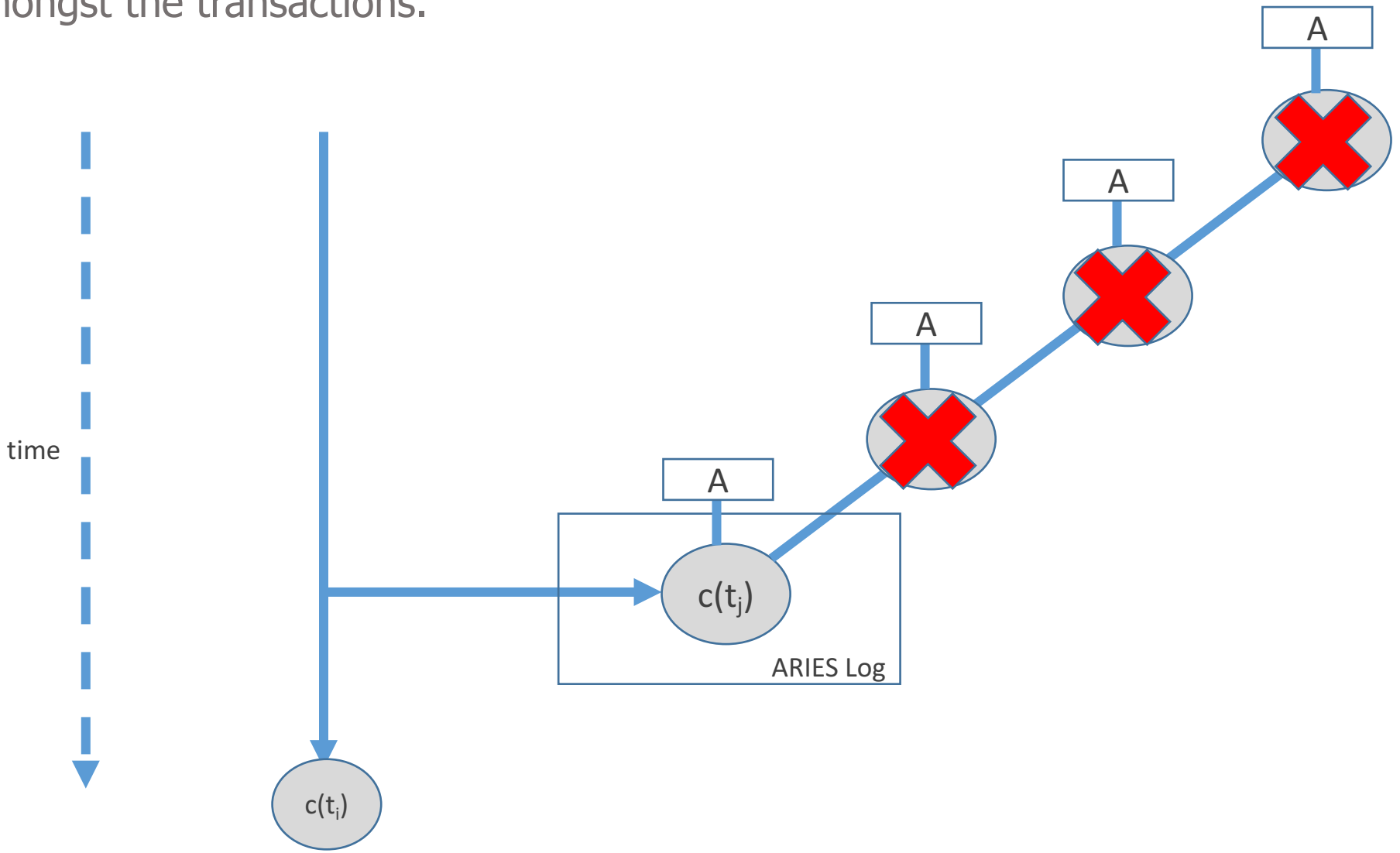
(a) ARIES Log



(b) Footprint Log

Adaptive Logging

- The key bottleneck of distributed command logging is caused by the dependencies amongst the transactions.



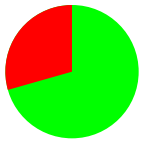
Adaptive Algorithm

Algorithm 6 `Online(Transaction` t_i`, int` $usedQuota$`)`

```
1: int q = getQuota(s(t_i)) - usedQuota
2: if q > 0 then
3:     Benefit b=computeBenefit(t)
4:     if b > τ then
5:         usedQuota++
6:         createAriesLog(t_i)
7:     else
8:         createCommandLog(t_i)
```

$$quota(k) = P(fail\_time < k)\frac{B_{i/o}}{W^{aries}}$$

$$b_i^{opt} = (\mathbb{N}R^{cmd}\sum_{\forall a_x \in A} P(a_x) - R^{aries}) \times \frac{1}{W^{aries}}$$



Experimental Evaluation

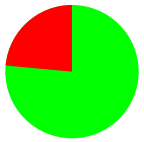
- **ARIES** – ARIES logging.
- **Command** – command logging
- **Dis-Command** – distributed command logging
- **Adapt-x** – adaptive logging

All the experiments are conducted on an in-house cluster of 17 nodes.

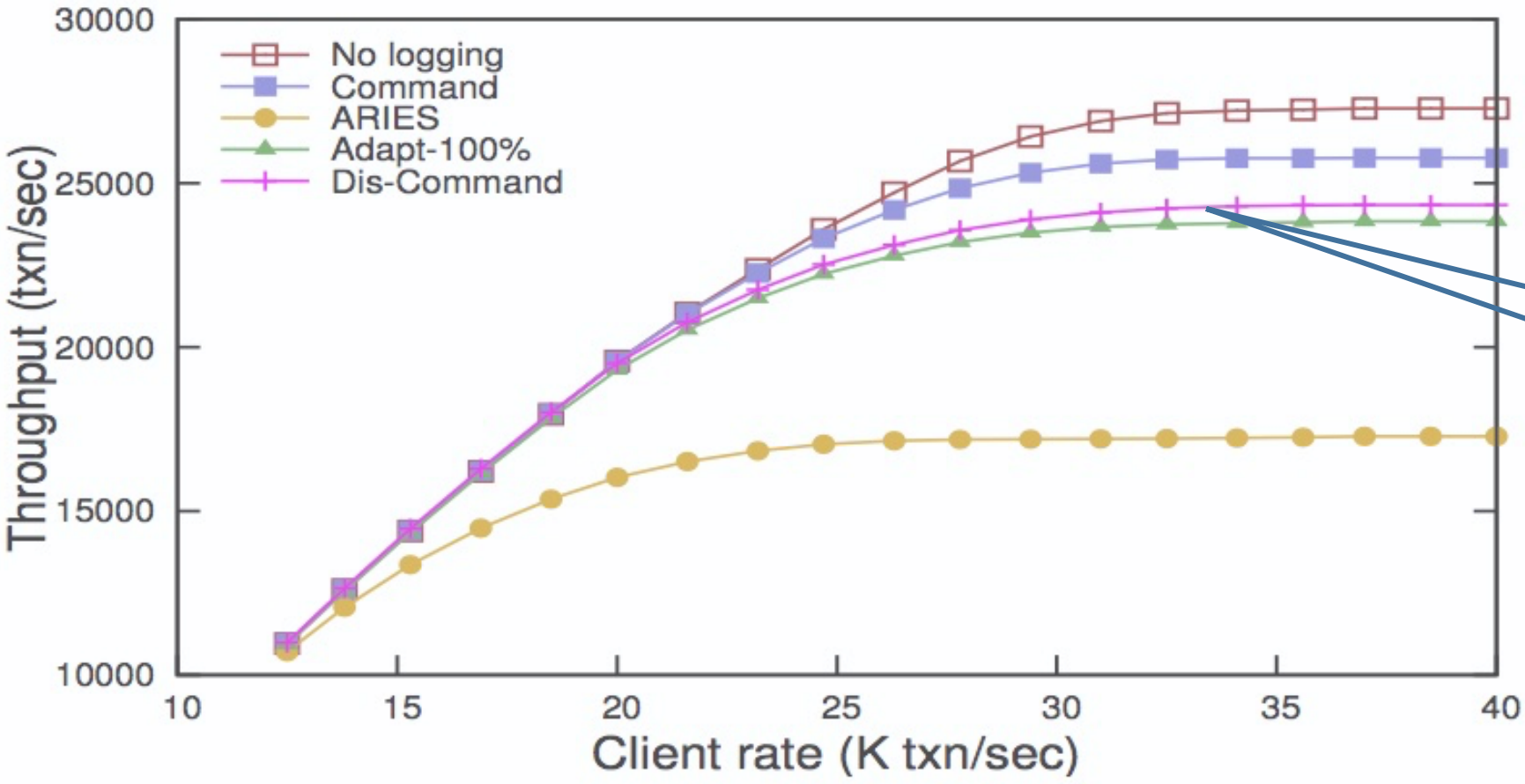
The head node is a powerful server equipped with an Intel(R) Xeon(R) 2.2 GHz 24-core CPU and 64 GB RAM.

The compute nodes are blades, each with an Intel(R) Xeon(R) 1.8 GHz 4- core CPU and 8 GB RAM.

H-Store is deployed on a cluster of 16 compute nodes with the database being partitioned evenly.

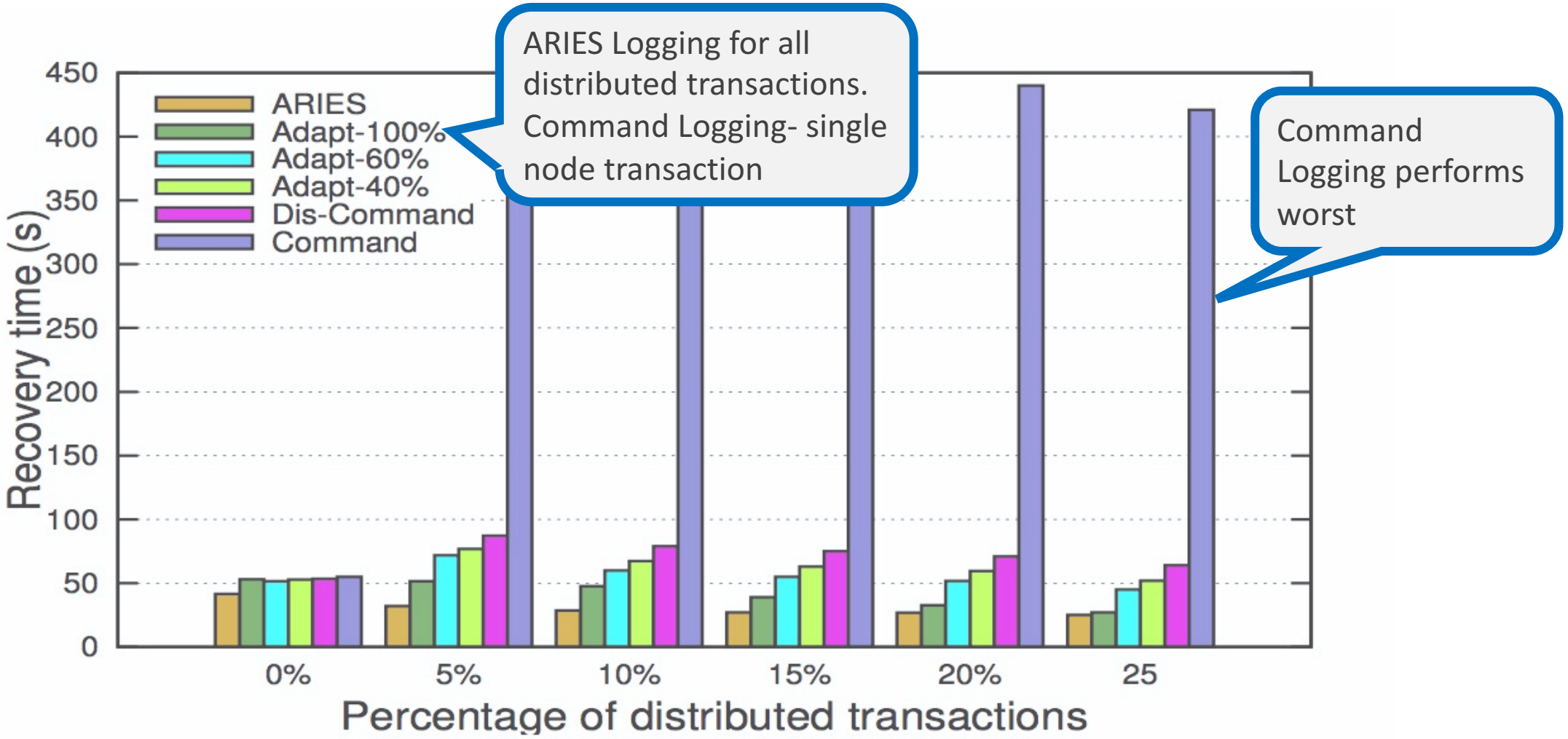


Throughput



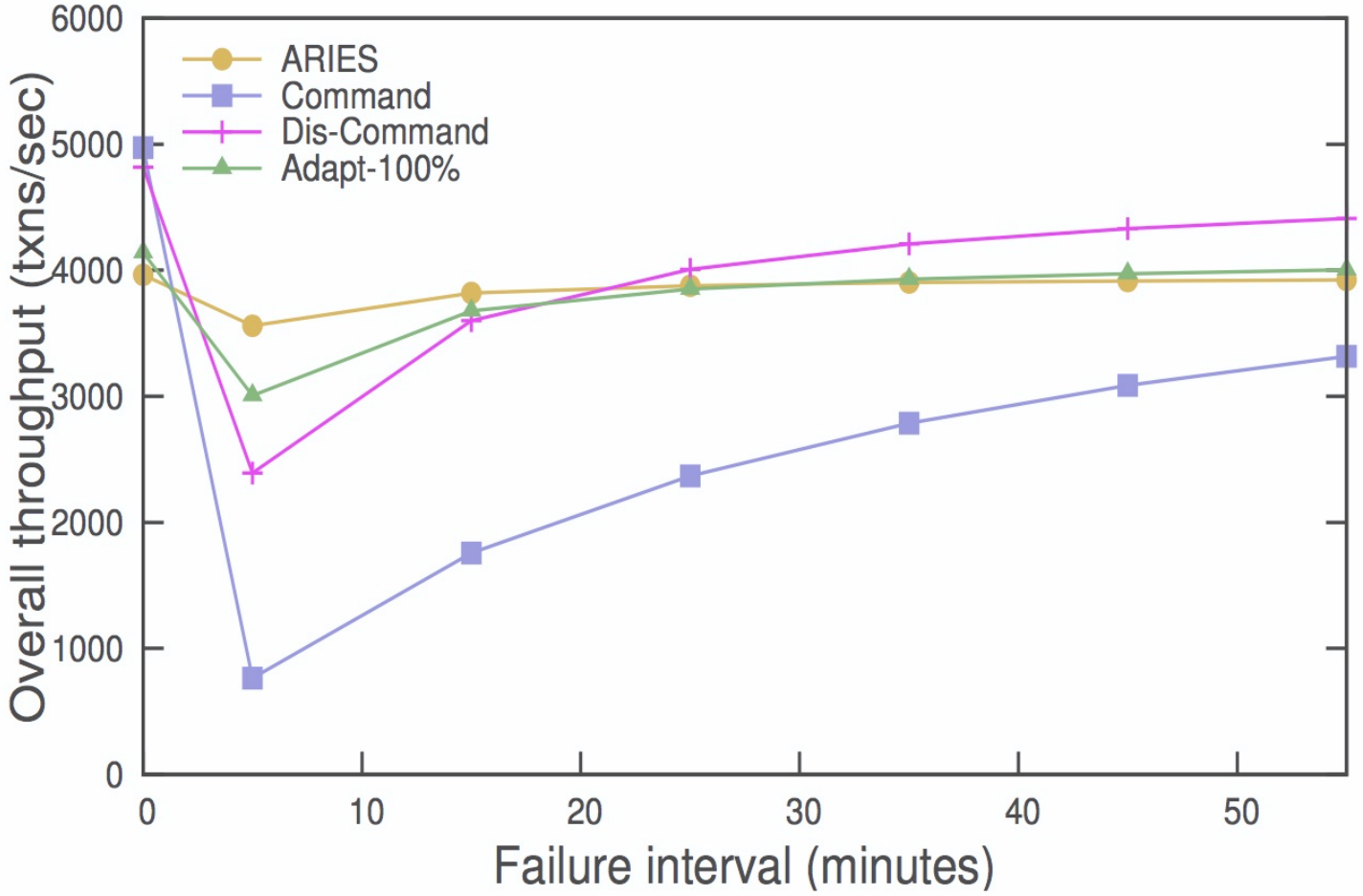
(a) Throughput without distributed transactions

Recovery



(a) 1 minute after the last checkpoint

Performance (Distributed Transactions)



(c) Overall throughput with 20% distributed transactions

Conclusion

- For in-memory databases, command logging shows better performance than ARIES logging.
- Command Logging significantly increases the recovery time due to sequential re-execution of the transactions.
- Adaptive Logging aims to achieve an optimized trade off between the runtime performance of transaction processing and the recovery performance upon failures.