FATE and DESTINI
A Framework for Cloud Recovery Testing

Haryadi S. Gunawi, Pallavi Joshi, Peter Alvaro, Joseph M. Hellerstein, and Koushik Sen

Thanh Do, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau

Dhruba Borthakur

Cloud and failure recovery

- Cloud
  - Thousands of commodity machines
  - *"Rare (HW) failures become frequent"* [Hamilton]
- Failure recovery
  - *"… has to come from the software"* [Dean]
  - *"… must be a first-class op"* [Ramakrishnan et al.]
  - But ... hard to get right

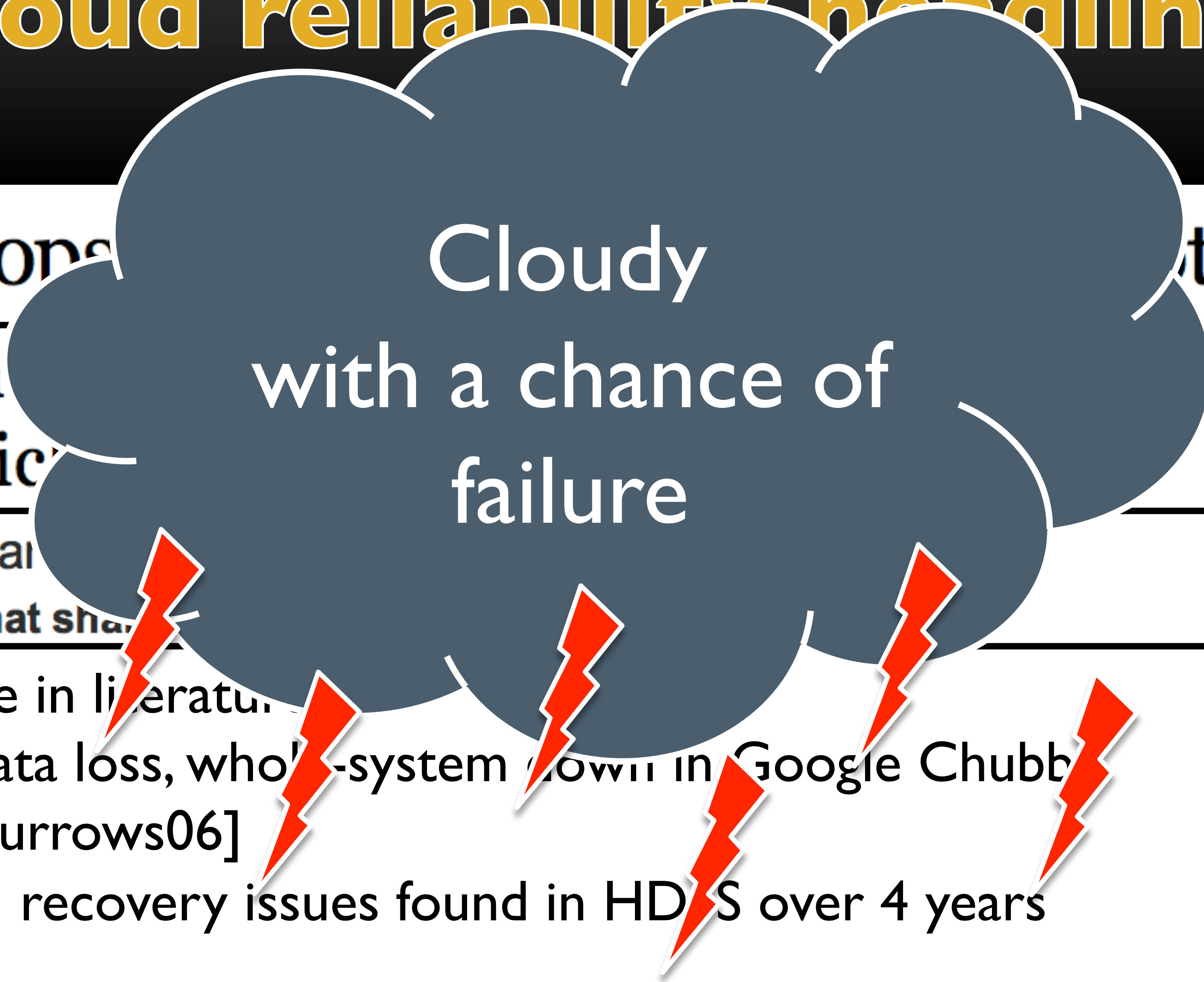
Cloud reliability headlines

Whoops ... tos

When ... ose
Sidekic...

Web sta...
With what sh...

B...
P...



More in literature:
- Data loss, whole-system down in Google Chubby [Burrows06]
- 91 recovery issues found in HDFS over 4 years
- ...

Why?

- Testing is not advanced enough
  - Cloud systems face complex multiple, diverse failures
- Recovery is under-specified
  - Lots of custom recovery
  - Implementation is complex
- Need two advancements:
  - Exercise complex failure modes
  - Write recovery specifications and test the implementation

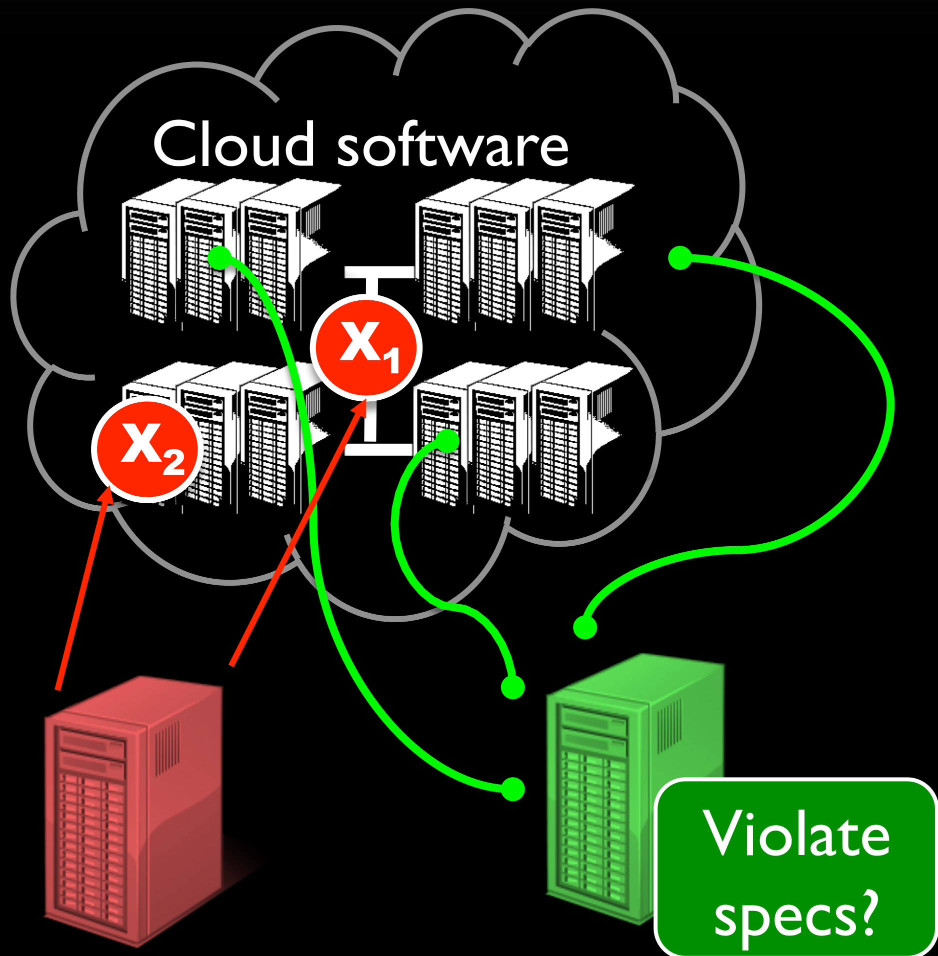
Cloud testing

FATE

Failure Testing Service

DESTINI

Declarative Testing Specifications



Contributions

- FATE
  - Exercise multiple, diverse failures
    - Over 40,000 unique combinations (80 hours)
    - Challenge: combinatorial explosion of multiple failures
  - Pruning strategies for failure exploration
    - An order of magnitude speedup
    - Found the same #bugs
- DESTINI
  - Facilitate recovery specifications
    - Reliability and availability related
  - Clear and concise (use Datalog, 5 lines/check)
  - Design patterns

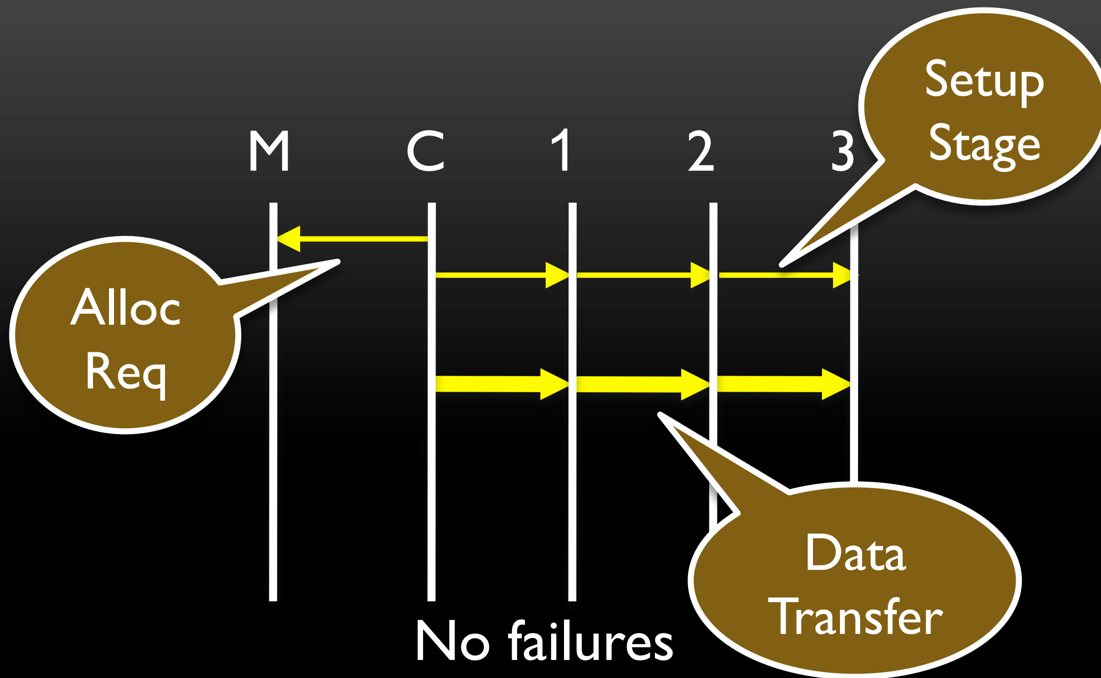
Summary of results

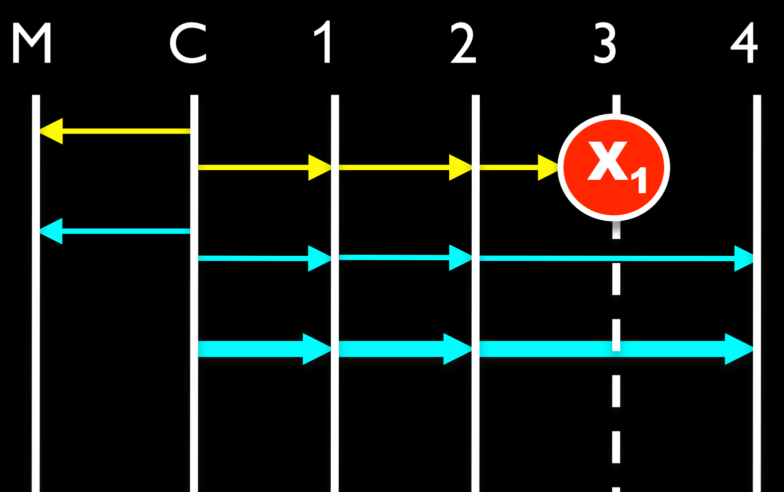
- Target 3 cloud systems
  - HDFS (primary target), Cassandra, and ZooKeeper
- HDFS recovery bugs
  - Found 16 new bugs (+6 in newest)
- Problems found
  - Data loss
    - Buggy recovery wipes out all replicas
  - Unavailability
    - Broken rack-aware policy
    - Can't restart after failures

Outline

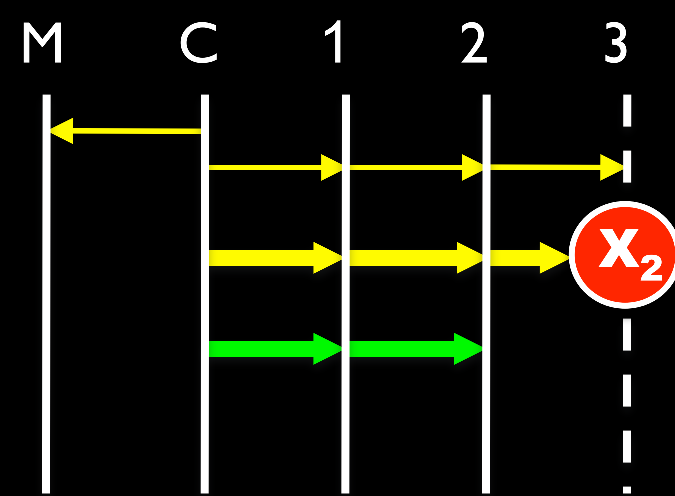
- Introduction
- **FATE**
  - Failure IDs: abstraction for failure exploration
  - Pruning strategies
- DESTINI
- Evaluation
- Conclusion

HadoopFS (HDFS) Write Protocol





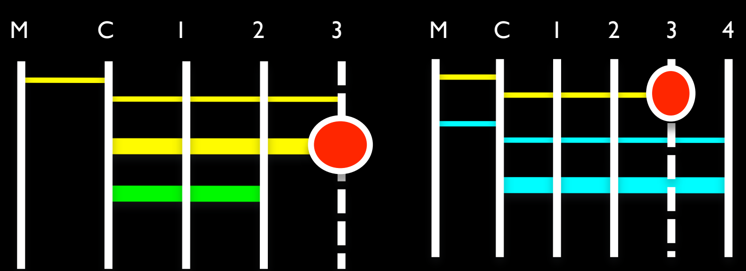
Setup Recovery:
Recreate fresh pipeline (1, 2, 4)

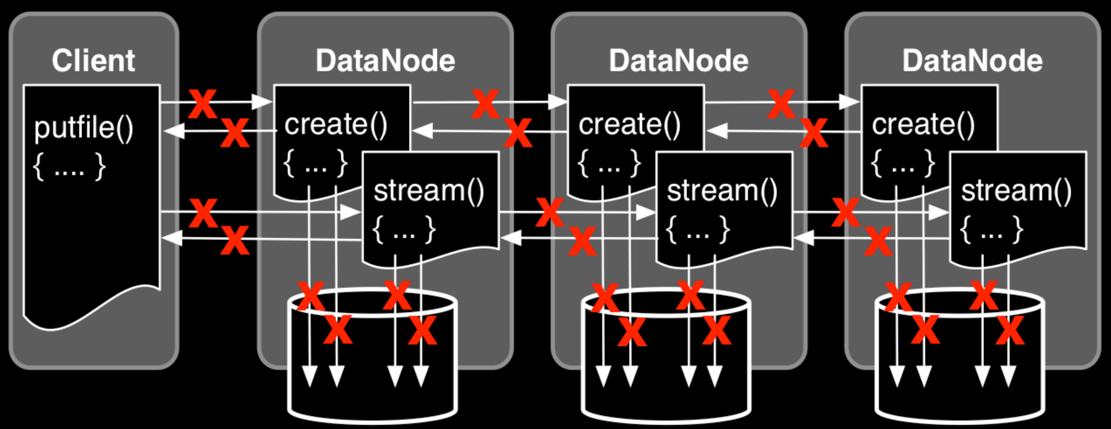


Data Transfer Recovery:
Continue on surviving nodes (1, 2)

Failures and FATE

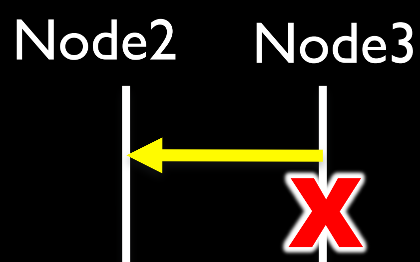
- Failures
  - Anytime: different stages → different recovery
  - Anywhere: N2 crash, and then N3
  - Any type: bad disks, partitioned nodes/racks
- FATE
  - Systematically exercise multiple, diverse failures
  - How? need to "remember" failures – via failure IDs





Failure IDs

- Abstraction of I/O failures
- Building failure IDs
  - Intercept every I/O
  - Inject possible failures
    - Ex: crash, network partition, disk failure (LSE/corruption)



Note:
FIDs
A, B, C, …

I/O information:	OutputStream.read() in BlockReceiver.java
	<stack trace> Net I/O from N3 to N2
	"Data Ack"
Injected failure:	Crash After
Failure ID: 2573	

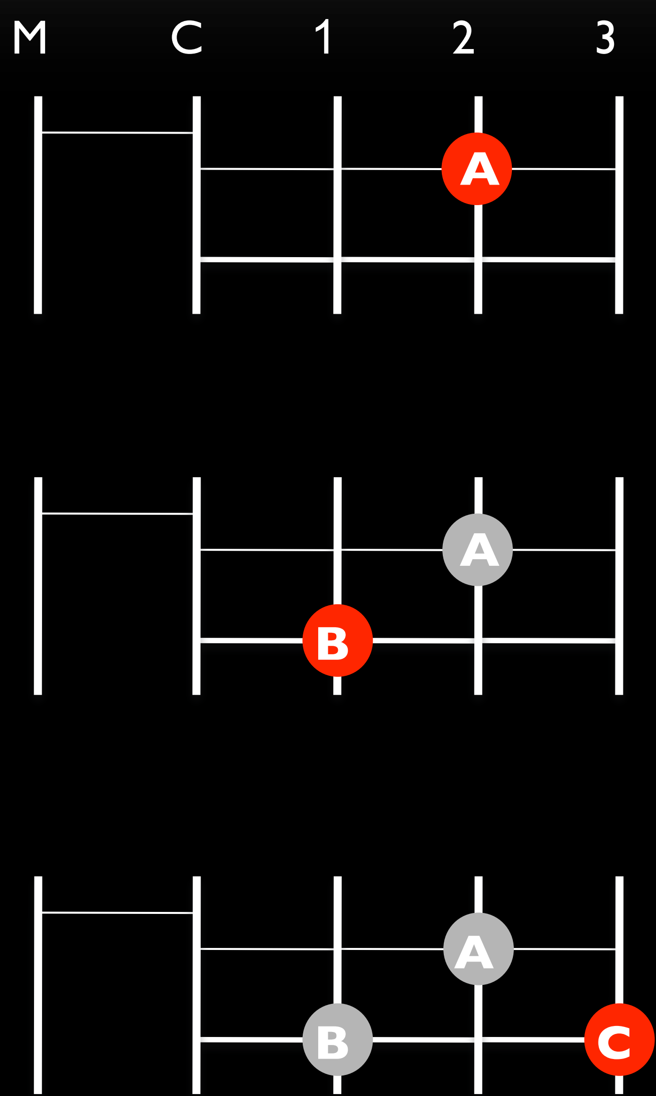
Brute-force exploration

1 failure / run

Exp #1: **A**

Exp #2: **B**

Exp #3: **C**

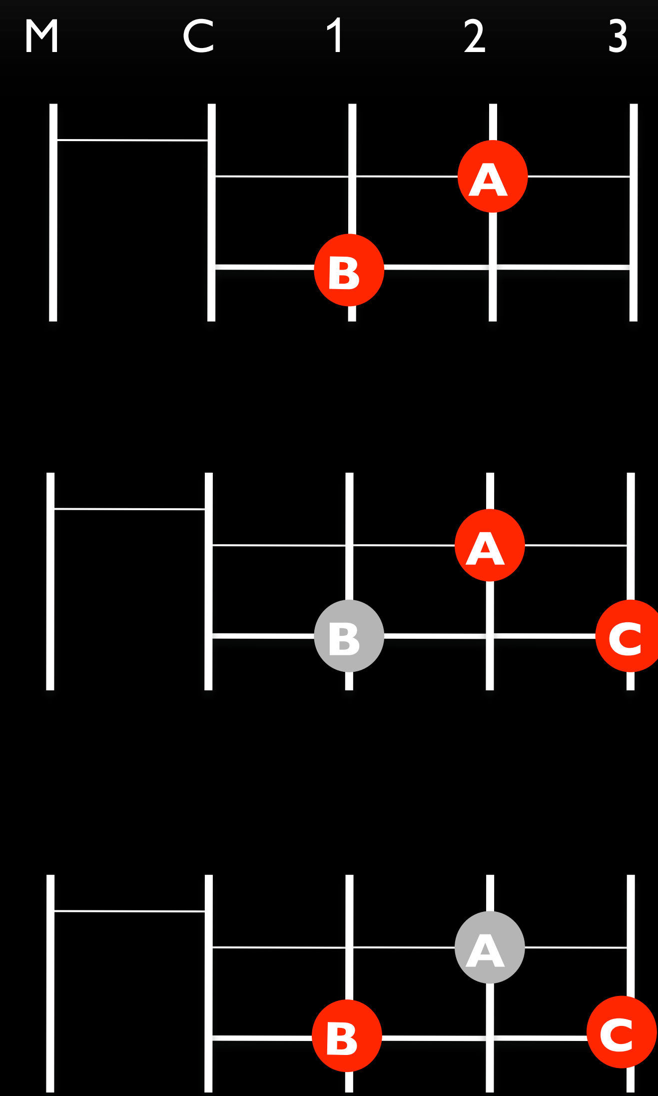


2 failures / run

AB

AC

BC



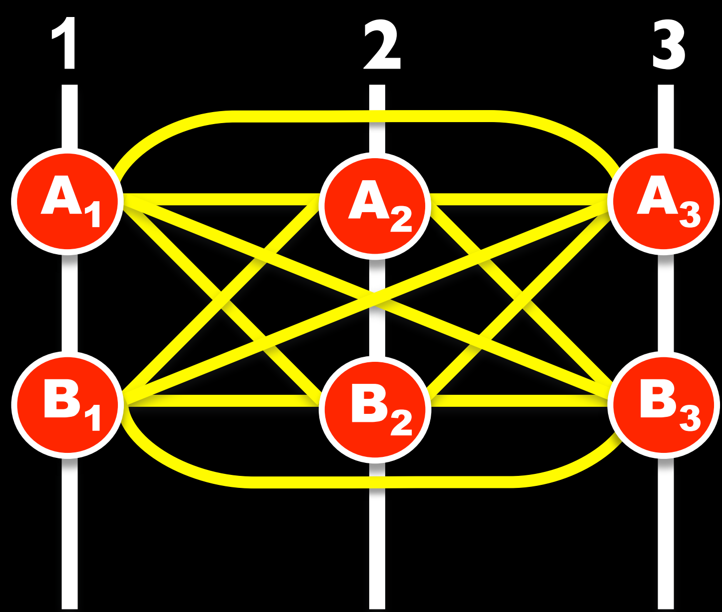
Outline

- Introduction
- **FATE**
  - Failure IDs: abstraction of failures
  - Pruning strategies for failure exploration
- DESTINI
- Evaluation
- Conclusion

Combinatorial explosion

- Exercised over 40,000 unique combinations of 1, 2, and 3 failures per run
  - 80 hours of testing time!

New challenge:
Combinatorial explosion of multiple failures



2 failures / run

```
A1 A2
A1 B2
B1 A2
B1 B2
...
```

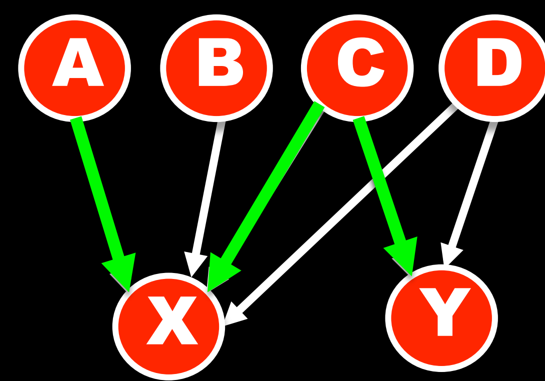
Pruning multiple failures

- Properties of multiple failures
  - Pairwise dependent failure IDs
  - Pairwise independent failure IDs
- **Goal:** exercise distinct recovery behaviors
  - **Key:** some failures result in similar recovery
  - Result: > 10x faster, and found the same bugs

Dependent failures

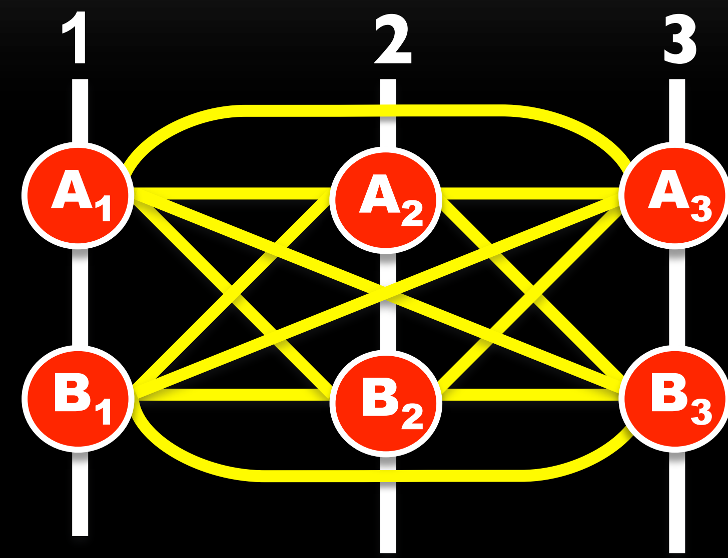
- Failure dependency graph
  - Inject single failures first
  - Record subsequent dependent IDs
    - Ex: X depends on A
  - **Brute-force:** AX, BX, CX, DX, CY, DY
- Recovery clustering
  - Two clusters: {X} and {X, Y}
- Only exercise distinct clusters
  - Pick a failureID that triggers a recovery cluster
  - Results: AX, CX, CY

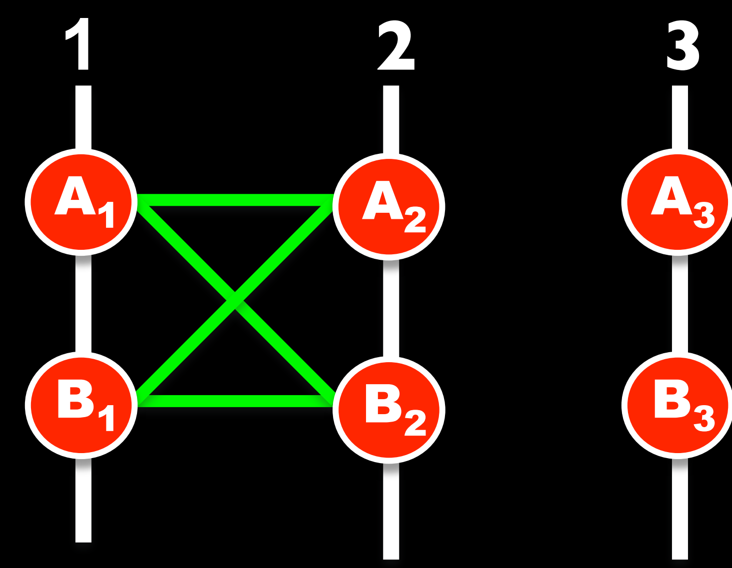
FID	→	Subseq FIDs
A	→	X
B	→	X
C	→	X, Y
D	→	X, Y



Independent failure IDs

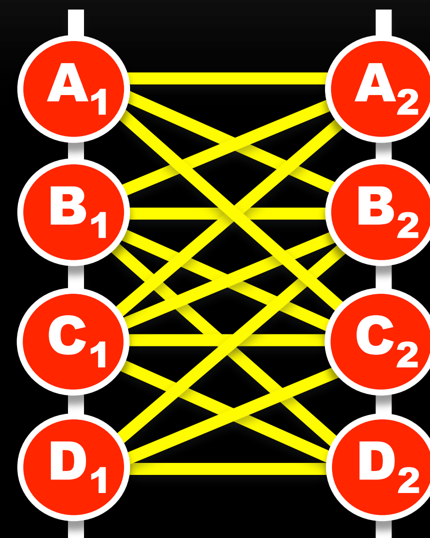
- Independent combinations
  - Ex: FP = 2, N = 3
  - **$FP^2 \times N\ (N-1)$**
- Symmetric code
  - Just pick two nodes
  - $N\ (N-1) \rightarrow 2$
  - **$FP^2 \times 2$**

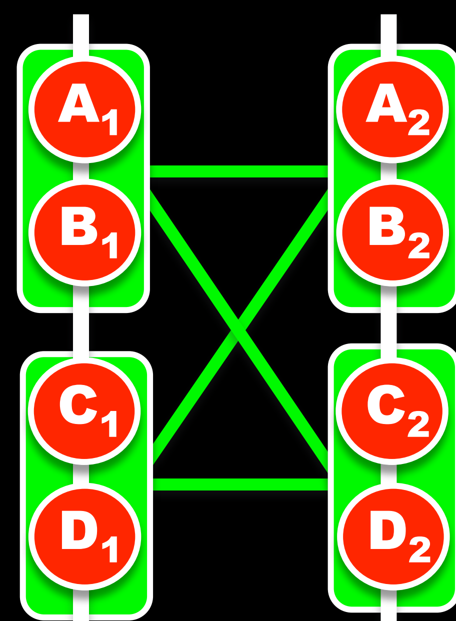




Independent failure IDs

- **FP^2** bottleneck
  - Ex: FP = 4
  - Real example: FP = 15
- Recovery clustering
  - Cluster A and B if: fail(A) == fail(B)
  - Reduce **FP^2** to **$FP^2_{clustered}$**
  - E.g. 15 FPs to 8 $FPs_{clustered}$





FATE Summary

- Contributions
  - Exercise multiple, diverse failures (via failure IDs)
  - Pruning strategies (> 10x improvement)
- Limitations
  - I/O reordering
  - Inclusion of states to failure IDs
  - More failure modes
    - Transient, slow-down, and data-center partitioning

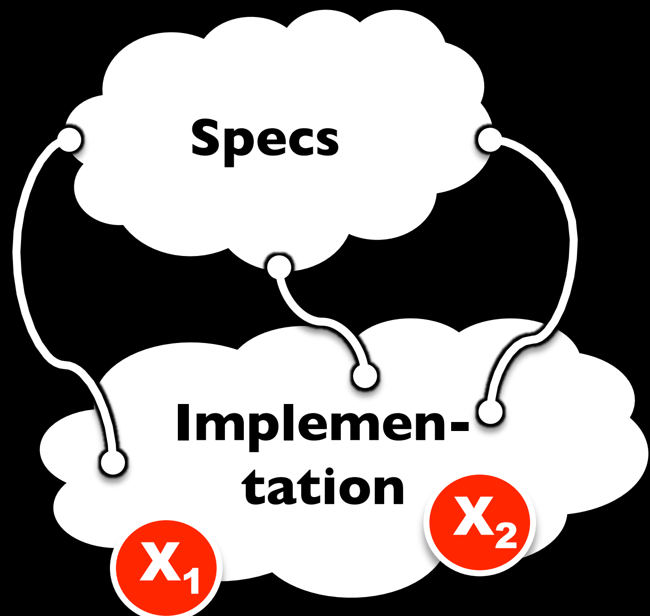
Outline

- Introduction
- FATE
- DESTINI: Declarative Testing Specifications
- Evaluation
- Conclusion

DESTINI: declarative specs

- Is the system correct under failures?
  - Need to write specifications
  - FATE needs DESTINI

> [It is] great to document (in a spec) the HDFS write protocol ...
>
> …, but we shouldn't spend **too** much time on it, … a formal spec may be overkill for a protocol we plan to deprecate imminently.

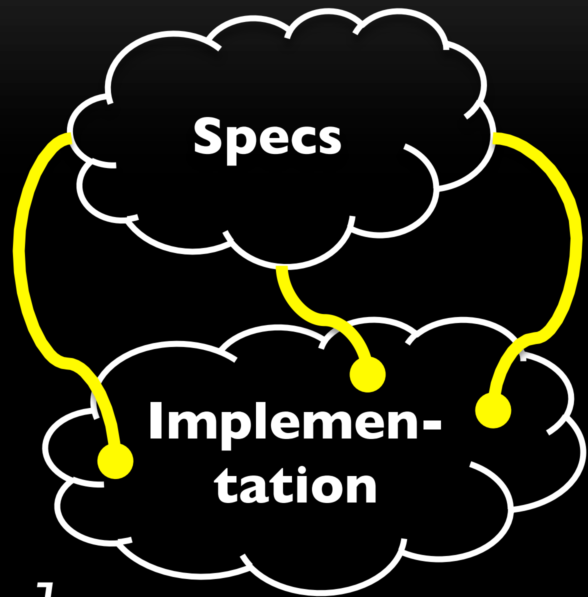


Declarative specs

- How to write specifications?
  - Developer friendly (clear, concise, easy)
- Datalog: a declarative relational logic language
  - Easy to express logical relations
  - (just for writing specifications)

Specs in DESTINI

- How to write specs?
  - Violations
  - Expectations
  - Facts
- How to write recovery specs?
  - *"... recovery is under specified" [Hamilton]*
  - Precise failure events
  - Precise check timings
- How to test implementation?
  - Interpose I/O calls (lightweight)
  - Deduce expectations and facts from I/O events



Specification template

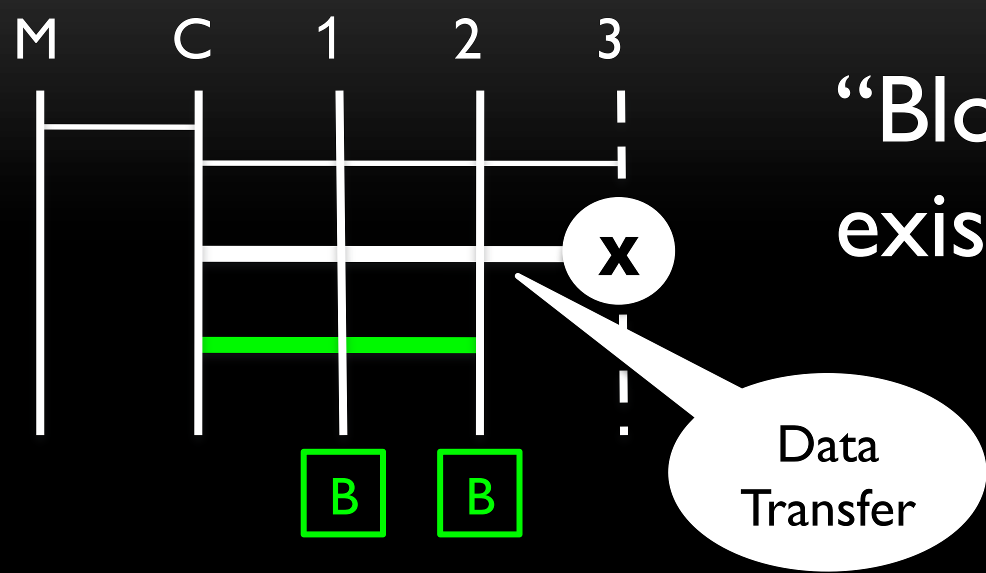
"Throw a violation if
an expectation is different from
the actual behavior"

violationTable(…) :-
expectationTable(…),
NOT-IN actualTable(…)

Datalog syntax:

head() :- predicates(), …
:- derivation
, AND

Data transfer recovery



"Block replicas should exist in surviving nodes"

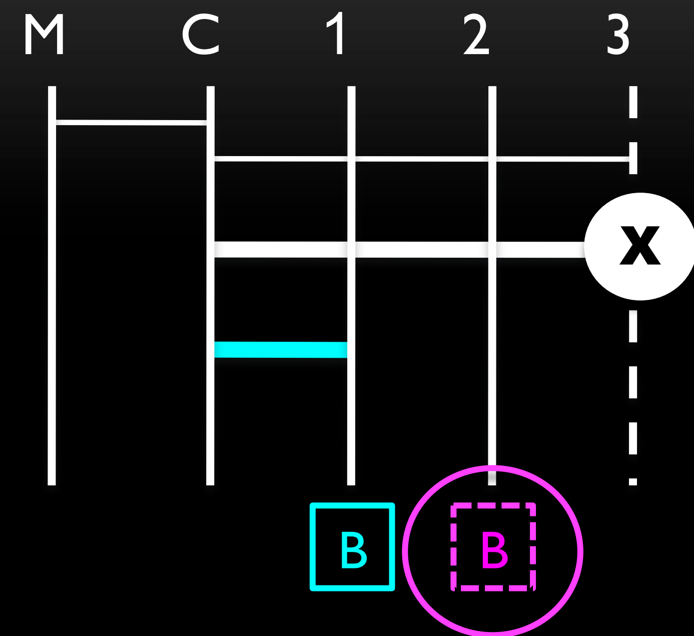
incorrectNodes (Block, Node)	

expectedNodes (Block, Node)	
B	Node 1
B	Node 2

actualNodes (Block, Node)	
B	Node 1
B	Node 2

incorrectNodes(B, N) :- expectedNodes(B, N), NOT-IN actualNodes(B, N);

Recovery bug



incorrectNodes (Block, Node)	
B	Node 2

expectedNodes (Block, Node)	
B	Node 1
B	Node 2

actualNodes (Block, Node)	
B	Node 1

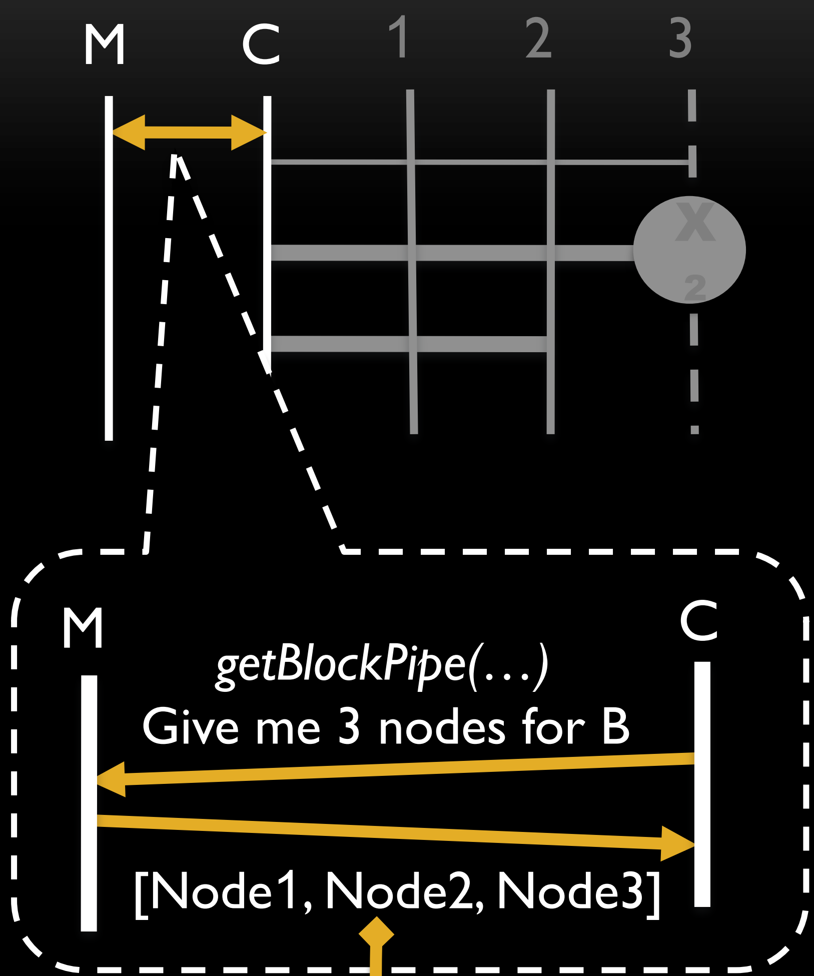
incorrectNodes(B, N) :- expectedNodes(B, N), NOT-IN actualNodes(B, N);

Building expectations

- Ex: which nodes **should** have the blocks?
  - Deduce expectations from *I/O events (italic)*

expectedNodes (Block, Node)	
B	Node 1
B	Node 2
B	Node 3

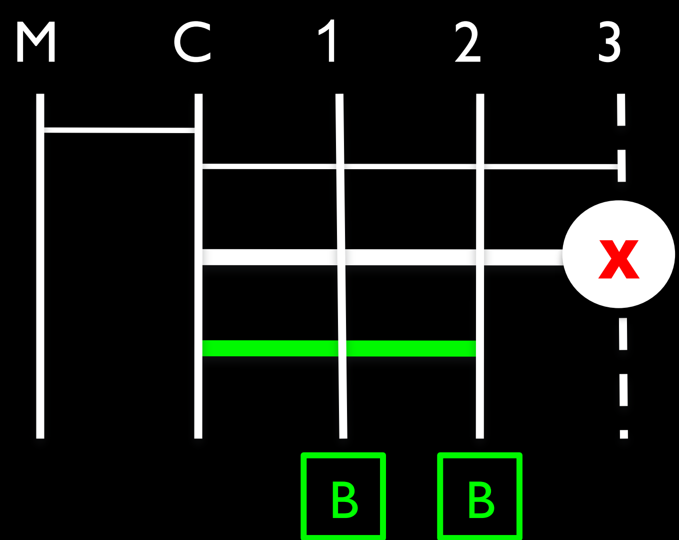
expectedNodes (B, N) :-
    getBlockPipe (B, N);



#1: incorrectNodes(B, N) :- expectedNodes(B, N), NOT-IN actualNodes(B, N);

Updating expectations

expectedNodes (Block, Node)	
B	Node 1
B	Node 2
~~B~~	~~Node 3~~



DEL expectedNodes (B, N) :-
expectedNodes (B, N),
fateCrashNode (N)

DESTINI
needs
FATE

#1: incorrectNodes(B, N) :- expectedNodes(B, N), NOT-IN actualNodes(B, N);

#2: expectedNodes(B, N) :- *getBlockPipe*(B,N);

Precise failure events

DEL expectedNodes (B, N) :-
expectedNodes (B, N),
fateCrashNode (N),
writeStage (B, Stage),
Stage == "Data Transfer";

Precise failure events →

#1: incorrectNodes(B,N)	:- expectedNodes(B,N), NOT-IN actualNodes(B,N)
#2: expectedNodes(B,N)	:- *getBlockPipe*(B,N);
#3: expectedNodes(B,N)	:- expectedNodes(B,N), fateCrashNode(N), **writeStg** (B,Stage), Stage == "DataTr"

#4: **writeStg**(B,"DataTr")	:- writeStg (B,"Setup"), nodesCnt(Nc), acksCnt (Ac), Nc==Ac
#5: nodesCnt (B, CNT<N>)	:- pipeNodes (B, N);
#6: pipeNodes (B, N)	:- *getBlockPipe* (B, N);
#7: acksCnt (B, CNT<A>)	:- setupAcks (B, P, "OK");
#8: setupAcks (B, P, A)	:- *setupAck* (B, P, A);

Violation and check-timing

```
#1:
incorrectNodes(B, N) :-
    expectedNodes(B, N),
    NOT-IN actualNodes(B,
    N),
    completeBlock (B);
```

- **Recovery ≠ invariant**
  - If recovery is ongoing, invariants are violated
  - Don't want false alarms
- Need precise check timings
  - Ex: *upon block completion*

DESTINI Summary

- Support recovery specs
  - Reliability and availability related
  - Clear and concise (use Datalog)
- Design patterns
  - Add detailed specs
  - Write specs from different views (global, client, …)
  - Incorporate diverse failures (crashes, rack partitions)
  - … more in the paper

Outline

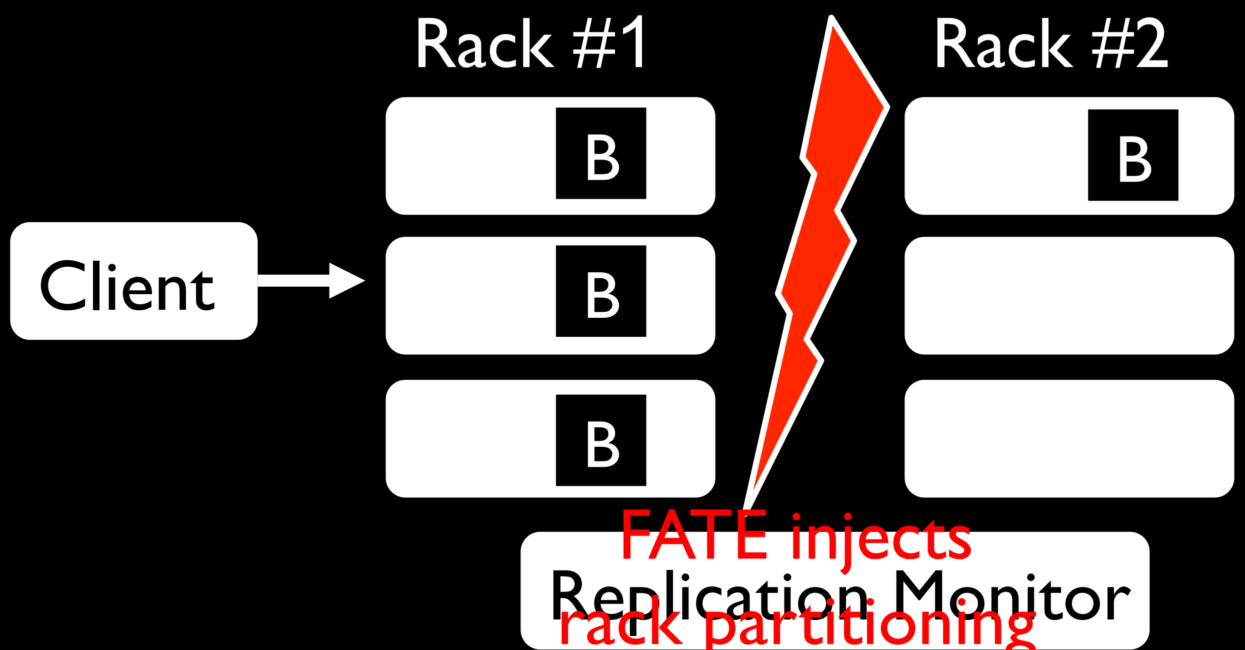
- Introduction
- FATE
- DESTINI
- Evaluation and conclusion

Evaluation

- Implementation complexity
  - ~6000 LOC in Java
- Target 3 popular cloud systems
  - HDFS (primary), ZooKeeper, Cassandra
- HDFS recovery bugs
  - Found 22 new bugs
    - 8 bugs due to multiple failures
    - Data loss, unavailability bugs
  - Reproduced 51 old bugs

Availability bug

"If multiple racks are available (reachable), a block should be stored in a minimum of two racks"



Availability bug!
#replicas = 3,
locations are not checked
B is not migrated to R2

Availability bug

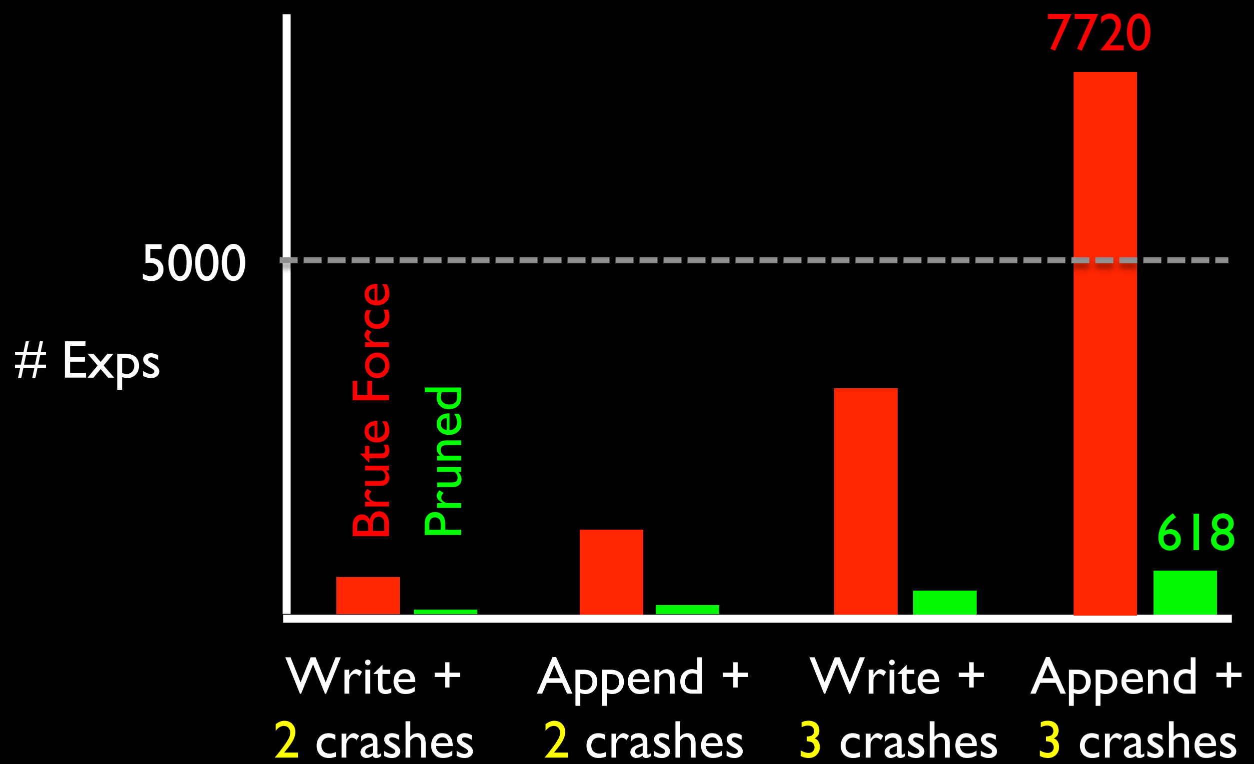
"If multiple racks are available (reachable),
a block should be stored in a minimum of two racks"

```
errorSingleRack(B) :- rackCnt(B,Cnt), Cnt==1, blkRacks(B,R), connected(R,Rb),
                      endOfReplicationMonitor (_);
```

errorSingleRack	←	rackCnt	blkRacks	connected
B		B, 1	B, R1	R1, R2

Pruning Efficiency

- Reduce #experiments by an order of magnitude
  - Each experiment = 4-9 seconds
- Found the same number of bugs
  - (by experience)



Specification simplicity

Framework	#Chks	Lines/Chk
D3S [NSDI '08]	10	**53**
Pip [NSDI '06]	44	**43**
WiDS [NSDI '07]	15	**22**
P2 Monitor [EuroSys '06]	11	**12**
DESTINI	74	**5**

- Compared to other related work

Conclusion

- Cloud software systems
  - Must deal with HW failures
- FATE and DESTINI
  - Explore multiple, diverse failures systematically
  - Facilitate concise recovery specifications
  - A unified framework
    - FATE needs DESTINI
    - DESTINI needs FATE
- Real-world adoption in progress

Thank you!
Questions?

Berkeley Orders Of Magnitude

http://boom.cs.berkeley.edu

http://cs.wisc.edu/adsl