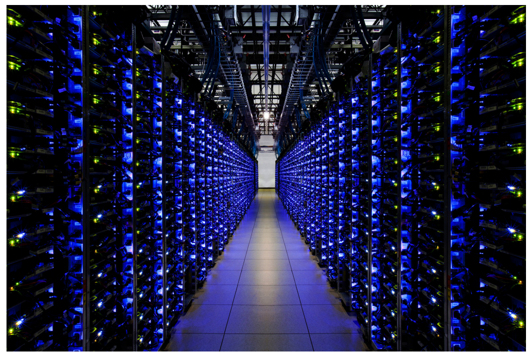
Evaluating Scalability Bottlenecks by Workload Extrapolation

Rong Shi, Yifan Gan, Yang Wang

The Ohio State University

Big data era



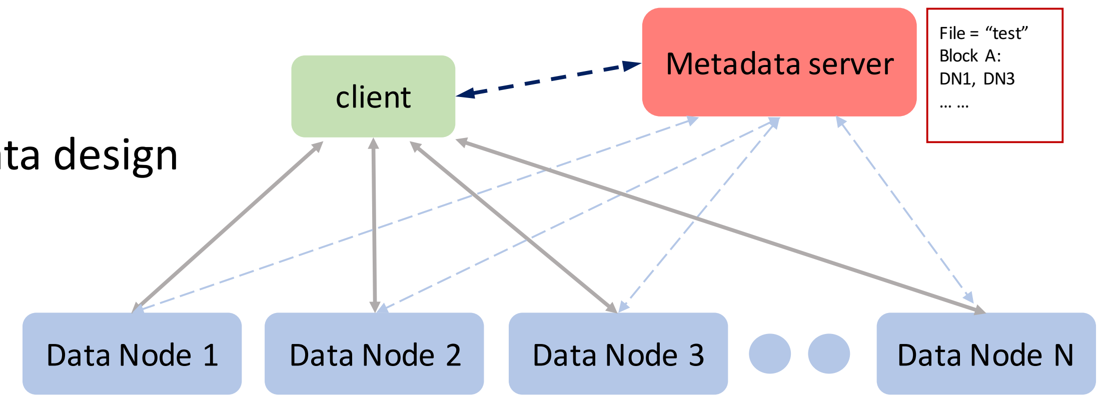
- Data is growing

  Large-scale distributed systems are widely deployed to store and process data

Centralized scalability bottlenecks

- Large-scale distributed systems

  Sharded data + centralized metadata design



- Bottleneck

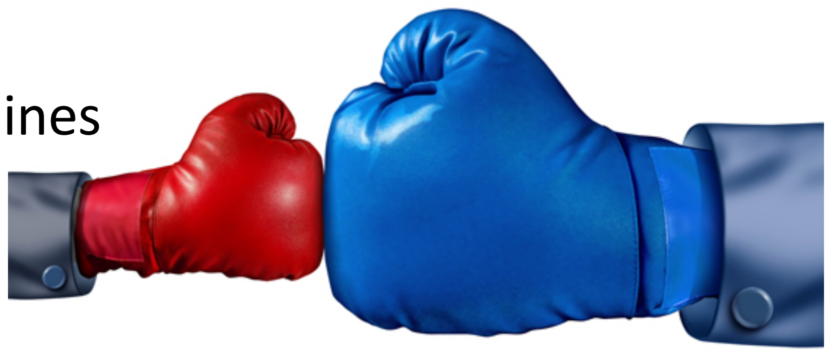
  Metadata server will eventually become the bottleneck as system scale increases

  Investigate bottlenecks to understand and improve system scalability

Evaluate system at large scale

Academia

- Hundreds of machines
- Hundreds of TBs



Public testbeds

limited resources, e.g. CloudLab (315 nodes)

Commercial platforms:

expensive e.g. $100 (100 nodes, 1h)

Industry

- Tens of thousands of machines
- Hundreds of PBs
- Growing fast

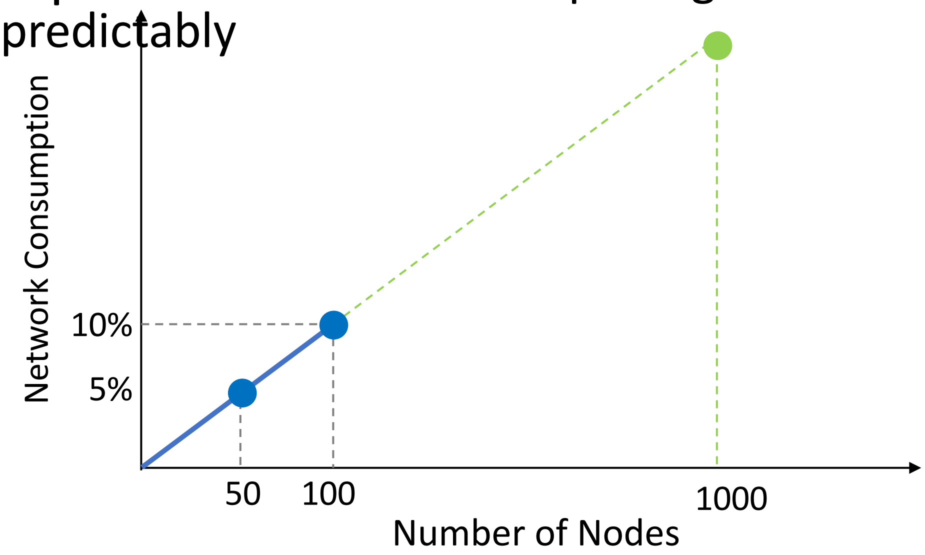
Facebook: >4k servers

Yahoo: 32k cluster

Can we evaluate centralized scalability bottlenecks on small testbeds?

First approach: resource extrapolation

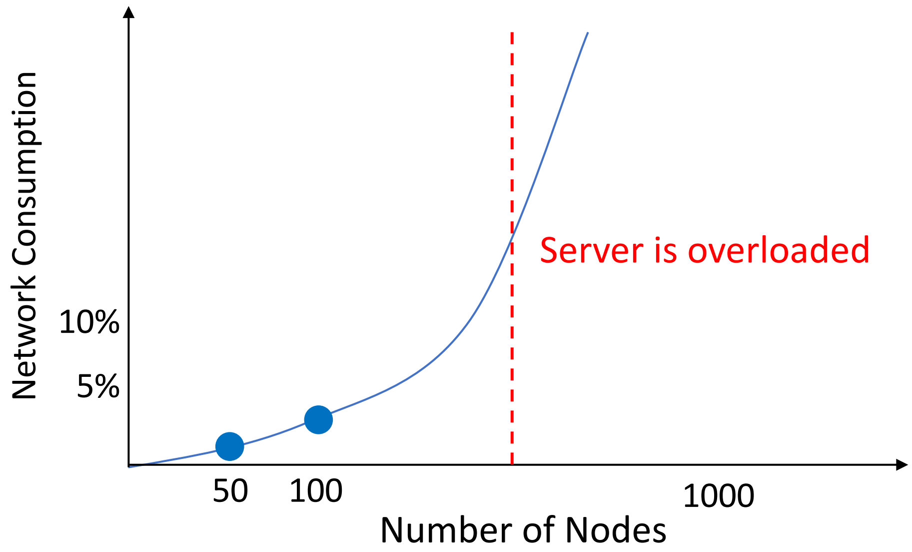
- Measure resource consumption of a bottleneck at small scales
- Predict its resource consumption at a large scale
- Assumption: resource consumption grows linearly with the scale or at least predictably



First approach: resource extrapolation

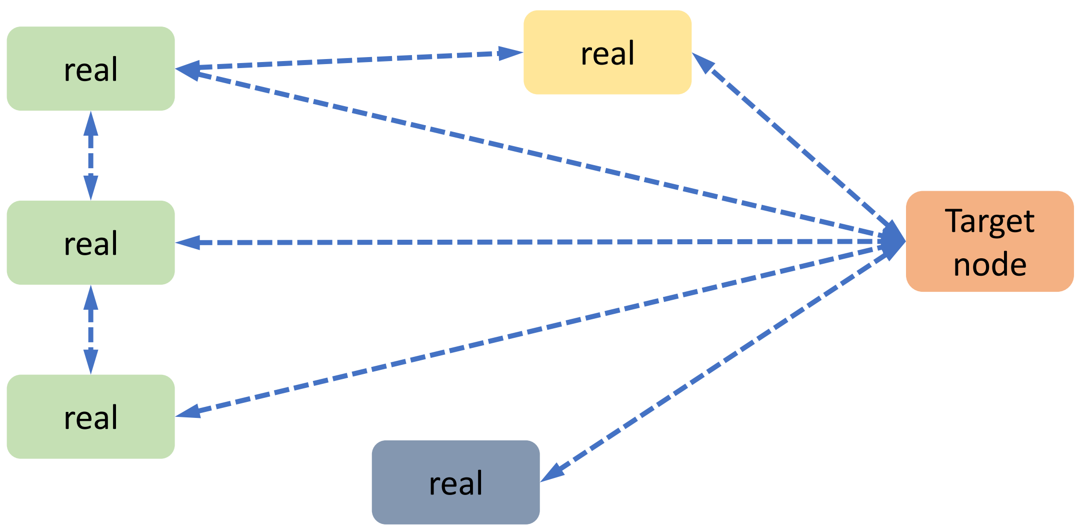
- Assumption can be violated:

- Problem only occur when the system reaches certain limit
- Resource consumption grows super linearly with the scale



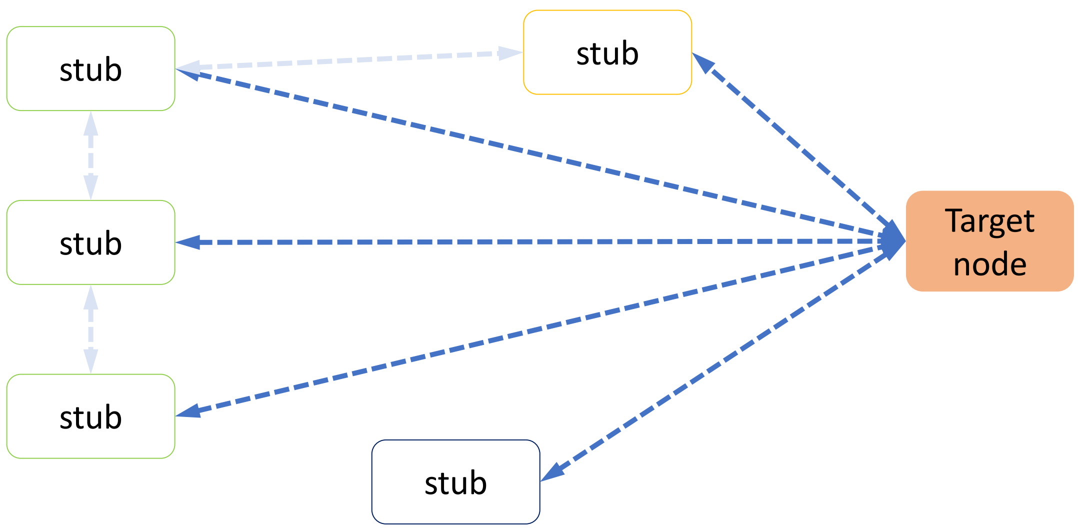
Second approach: stubs

- Run target node in real mode and build stubs to simulate others



Second approach: stubs

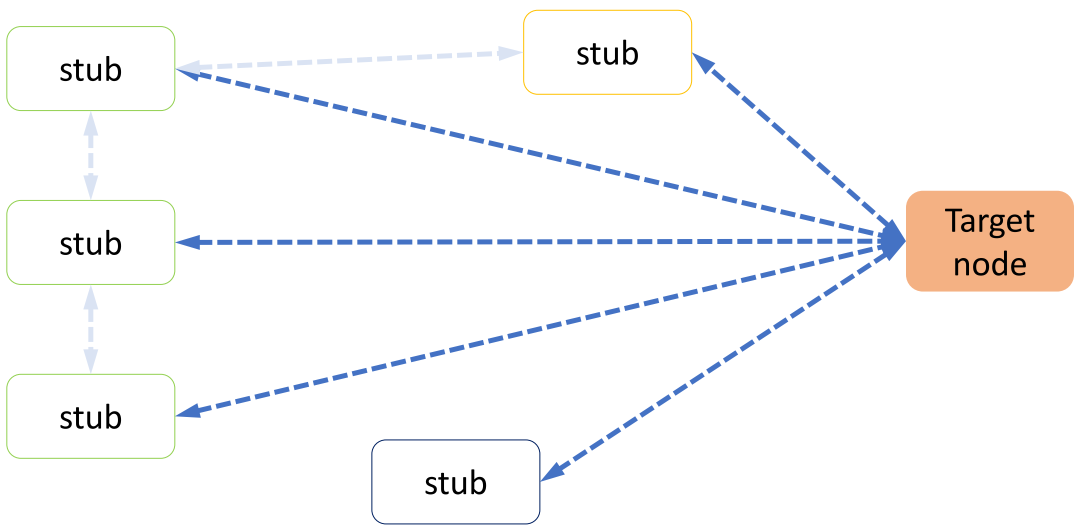
- Run target node in real mode and build stubs to simulate others
  Work well with simple benchmarks, or under specific assumption



Second approach: stubs

- Problem: stubs could be complex

  5GB WordCount to NameNode: 1171 RPCs (19 types), 23592 arguments, 6 type of nodes



Trade-off

- Resource extrapolation
  - Applicable to any system
  - Not really test metadata server under heavy load, which hurts its accuracy
- Stubs
  - Keep the logic of the bottleneck node and thus has good accuracy, assuming stubs are accurate
  - Building accurate stubs is complex unless satisfying certain assumptions

Is it possible to strike a balance between applicability and accuracy?

Is it possible to strike a balance between applicability and accuracy?

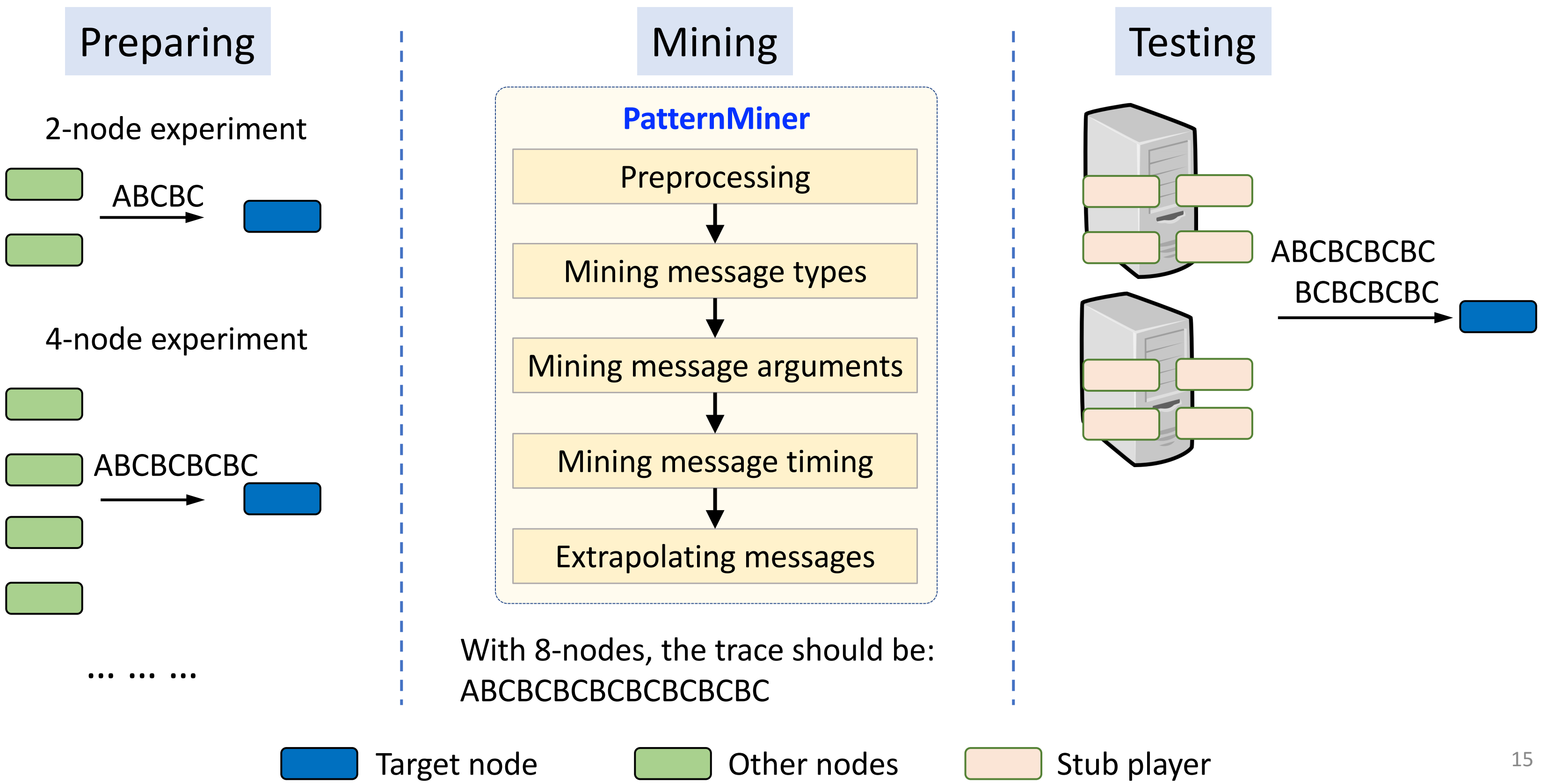
Yes for centralized metadata servers

Key observation: systems at a large scale are often repeating their behaviors at small scales

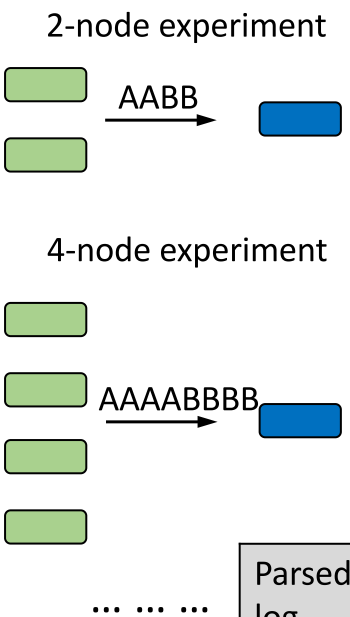
- Users tend to run same job many times with different inputs
- Run same code pieces on many nodes to scale to large number of nodes
- Use loops to adapt code to the growing amount of data

- Provides an opportunity for accurate workload extrapolation
  - Replace data servers with light-weight stubs
  - Extrapolate their output messages to the metadata services

Work flow of workload extrapolation



Preparing



- Requirements of workload logs

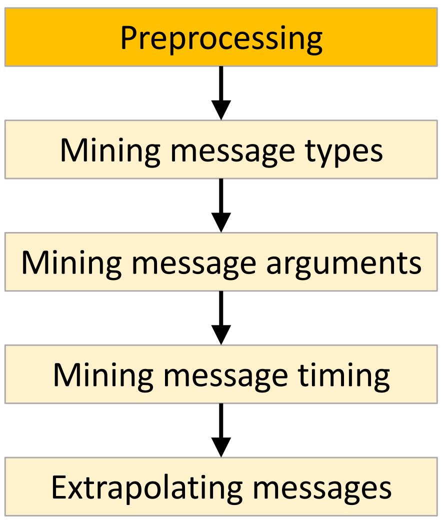
  complete and semantically meaningful

- Light-weight instrumentation

  2017-11-16 13:21:59.012 sender=**29054:32** rpc=**ClientProtocol.getFileInfo**
  Call#**0** Retry#**0** request={**"src": "/terasort/in-4"**}, response={**“”**}

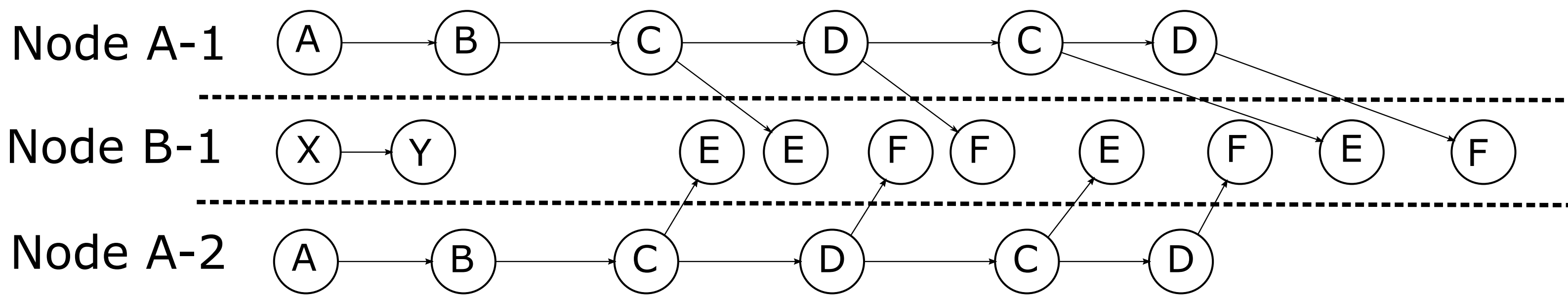
Parsed log	timeStamp	senderID	RPC name	argument_request		argument_reply
	13:21:59.012	29054:32	getFileInfo	type: src	/terasort/in-4	null

Mining using PatternMiner



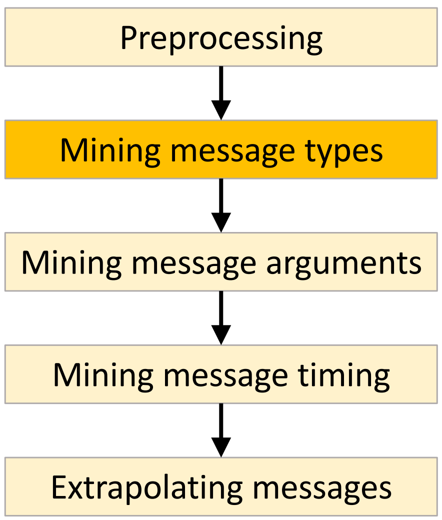
- Separate nodes' logs based on senderID
- Relocate some RPCs by causal ordering
  - Track unique IDs of certain tasks (e.g.block_id, ts)
- Cluster logs by histogram of RPC names

Relocate RPCs based on causal ordering



- E/F in B-1 triggered by C/D in A-1 and A-2
- Repetition is not clear in B-1, due to non-determinism in timing
- Causal ordering technique could alleviate this problem

Mining



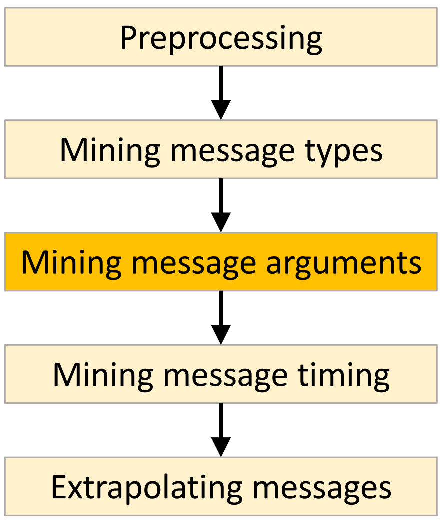
- Detect nondeterministic RPCs
- Identify static and repeated patterns

  Sequence: list of template <type, pattern, repetition>

- Validate key assumption

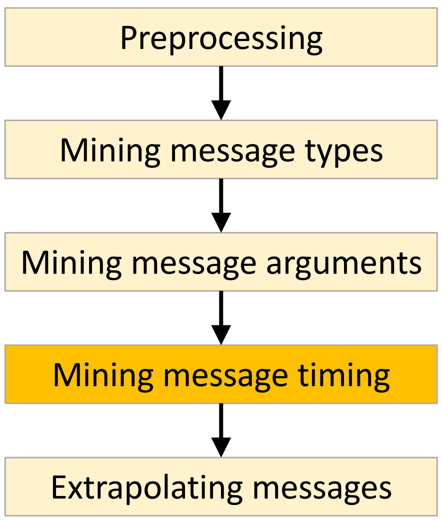
  segment information consistent across experiments, except for repeated segments

Mining



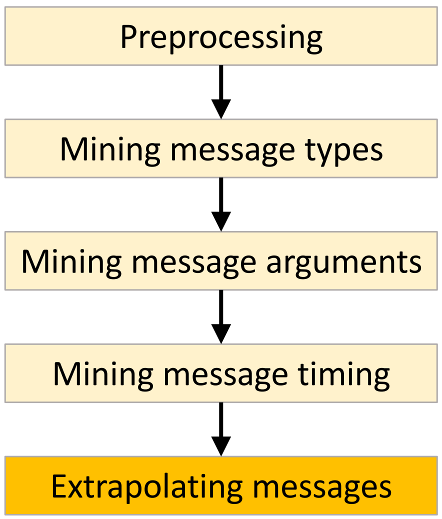
- Context
  - Segment info <type, seg, offset, iter>, environment info
- Regular pattern
  - Constant values, regularly changed values
  - Cross iteration/node summarization
  - Cross experiment validation
- Information flow
  - Values from args/return value of previous RPCs
  - Summarize args pattern and validation

Mining



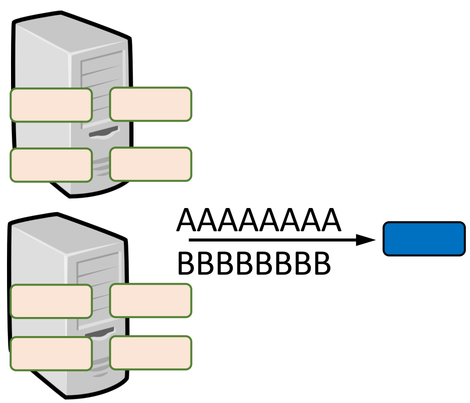
- Time intervals within the same node
  - Compute time-diff and use LR to check
- Starting time of a node
  - e.g. reducer starts after all mappers finish
  - Predefine a set of patterns (e.g. fork, join)

Mining



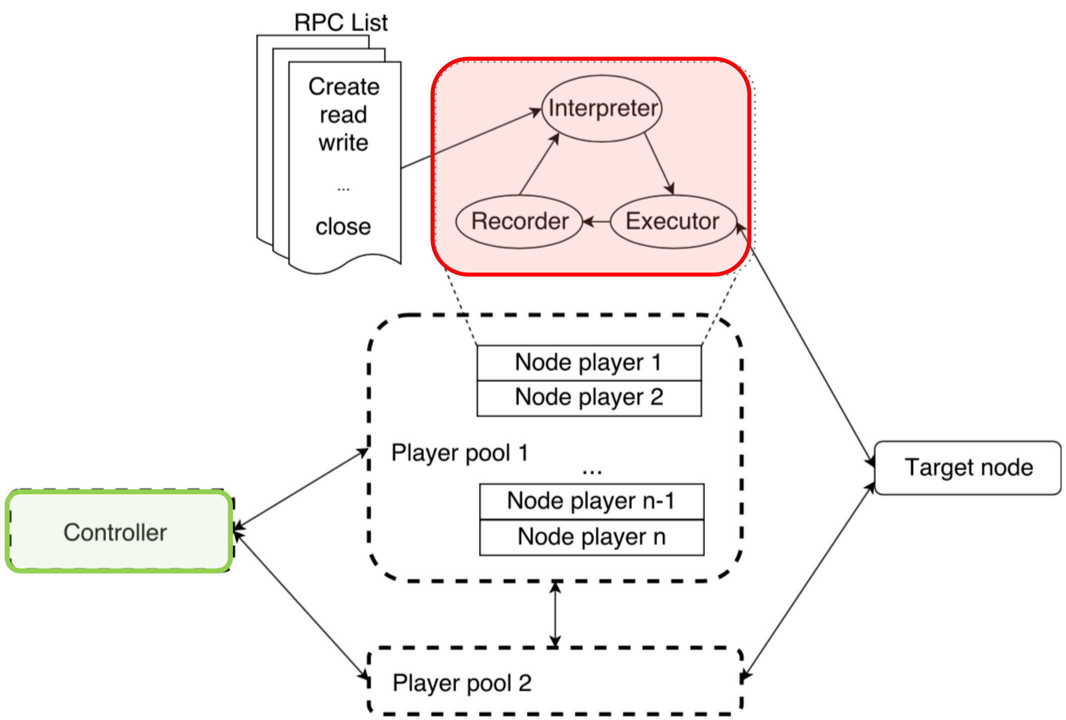
- Specify configuration by developers
- Extrapolate RPC types
  - Predict #iteration for repeated pattern
  - Leave real-time related patterns
- Extrapolate RPC arguments
  - Fill values for regular pattern
  - Fill template for information flow arguments
- Extrapolate timing
  - Directly use extracted interval (insert *sleep()*)
  - Generate rules to predict when a node starts

Testing



- Use stubs to replace all real nodes except the target node, collocate multiple stubs on same machine
- Run target node in real mode

Architecture of simulator



Evaluation

- How well can our approach extrapolate workloads (% predict)?
- How accurate is the extrapolated workload (v.s. real)?
- Can the extrapolated workload help identify performance problems?

Apply our approach to Hadoop and extrapolate workloads for HDFS NameNode and YARN Resource Manager with 4 benchmarks

How well can our approach extrapolate workloads?

	Total	C	R	IF	Unknown	%
NameNode						
WordCount	1371	754	47	541	29	97.88%
TeraSort	3134	1710	92	1278	54	98.28%
KMeans	3178	1773	84	1262	59	98.14%
InvertedIndex	2011	1130	48	800	33	98.36%
Resource Manager						
WordCount	179	108	18	22	31	82.68%

C = Constant, R = Regular pattern, IF = Information flow

How well can our approach extrapolate workloads?

	Total	C	R	IF	Unknown	%
NameNode						
WordCount	1371	754	47	541	29	97.88%
TeraSort	3134	1710	92	1278	54	98.28%
KMeans	3178	1773	84	1262	59	98.14%
InvertedIndex	2011	1130	48	800	33	98.36%
Resource Manager						
WordCount	179	108	18	22	31	82.68%

C = Constant, R = Regular pattern, IF = Information flow

- Random values (e.g. uuid)
- Timestamp (e.g. contextID, filename)
- Data-dependent (e.g. outputFile size, storage use)

- Easy to handle: Random values (random generator), TS (current time)
- Cannot be accurately estimated: data-dependent values

  - Put estimated values in testing, since NameNode’s performance is not sensitive

How well can our approach extrapolate workloads?

	Total	C	R	IF	Unknown	%
NameNode						
WordCount	1371	754	47	541	29	97.88%
TeraSort	3134	1710	92	1278	54	98.28%
KMeans	3178	1773	84	1262	59	98.14%
InvertedIndex	2011	1130	48	800	33	98.36%
Resource Manager						
WordCount	179	108	18	22	31	82.68%

- Port information (e.g. rpc_port)
- Specific files (e.g. size, creation time)
- Task progress (e.g. 20%)
- Argument of ***Allocate()* call**

C = Constant, R = Regular pattern, IF = Information flow

Allocate()

Report states (in most cases): predictable

Ask for new resource: write code to simulate internal logics

How well can our approach extrapolate workloads?

	Total	C	R	IF	Unknown	%
NameNode						
WordCount	1371	754	47	541	29	97.88%
TeraSort	3134	1710	92	1278	54	98.28%
KMeans	3178	1773	84	1262	59	98.14%
InvertedIndex	2011	1130	48	800	33	98.36%
Resource Manager						
WordCount	179	108	18	22	31	82.68%

C = Constant, R = Regular pattern, IF = Information flow

PatternMiner: **semi-automatic** tool

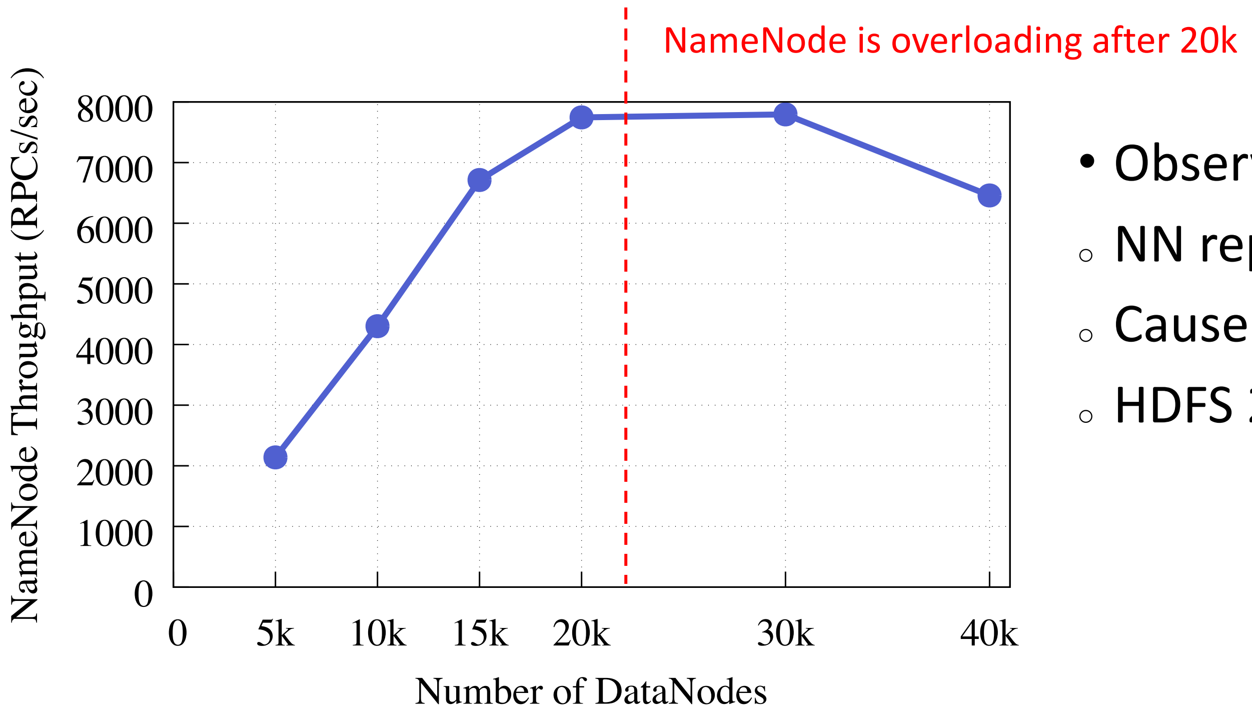
Developers need to handle nondeterminstic events and unknown argument patterns

How accurate is the extrapolated workload?

- Experiments: run WordCount and TeraSort on 500 nodes in Microsoft Azure
  - Record their traces to NameNode and Resource Manager
  - Run small-scale experiments, mine patterns, extrapolate workload of 500 nodes

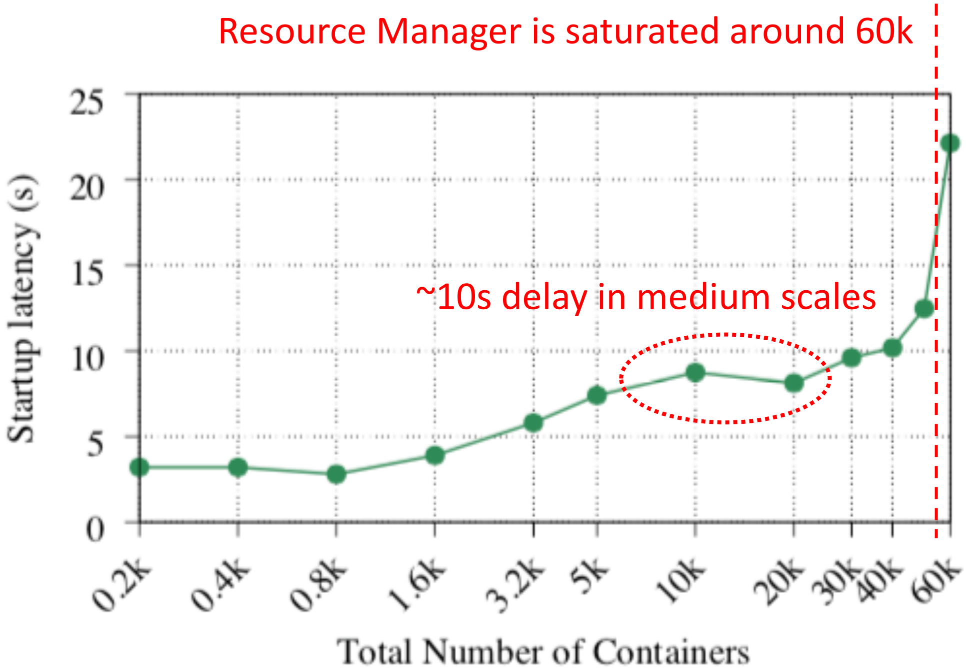
- Validation
  - RPC Sequences: match, except 3 failed DNs leading to differences on a few mappers
  - RPC arguments: match
  - Time interval: difference is within 10% for 90% (NN) and 99% (RM) of the intervals
  - Start time of nodes: match

Can the extrapolated workload help identify performance problems?



- Observe one correctness issue
  - NN report DNs as failed
  - Cause: burst traffic
  - HDFS 2.8 add lifeline protocol

Can the extrapolated workload help identify performance problems?



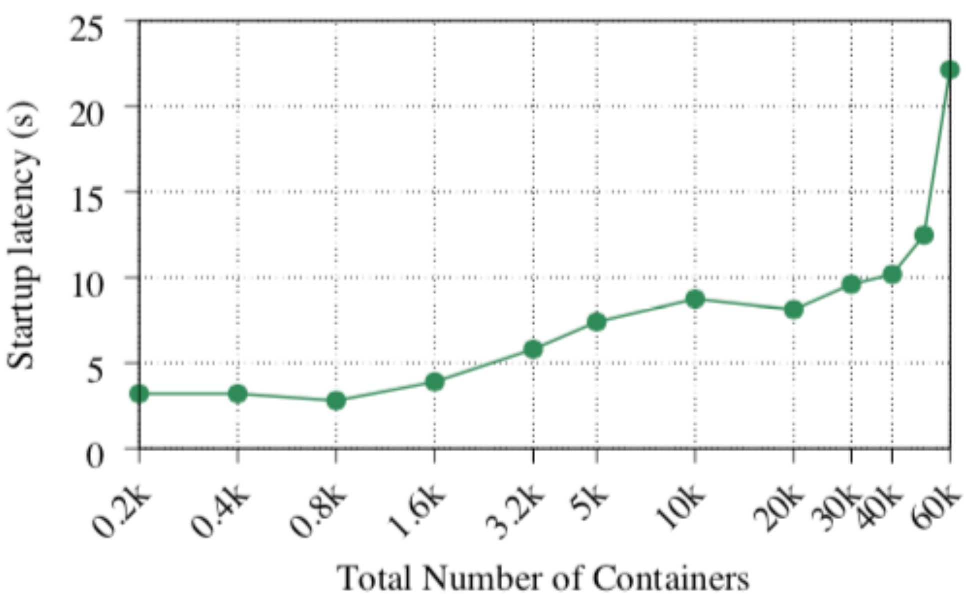
- Startup latency

  Register to RM -> Get all Containers

- Grows steadily, increase sharply around 60k

- ~10s delay (start application) is problematic for short tasks

Can the extrapolated workload help identify performance problems?



- Observe over-subscription issue
  - App may get more containers than it asks for
  - Cause: race condition of last batch allocation and request
  - Spark: gives back containers

Related work

- Evaluating system at large scale
  Industry: Facebook’s Kraken [Veer OSDI’16], …
  Stub: Exalt [Wang NSDI’14], Scale Check [Lees HotOS’17], ...
  Dynamometer [Linkedin]

- Workload extrapolation
  Analyze workloads in the past to predict workload in the future [Oly ICS’02], ...

- Log analysis
  Infer causal relationship between events [Zhao OSDI’16], ...

Conclusion

- Testing a scalability bottleneck is challenging

- Our solution:

  Test a bottleneck node by extrapolating a workload that the target node would observe at a large scale

https://github.com/OSUSysLab/HadoopMetadataBench