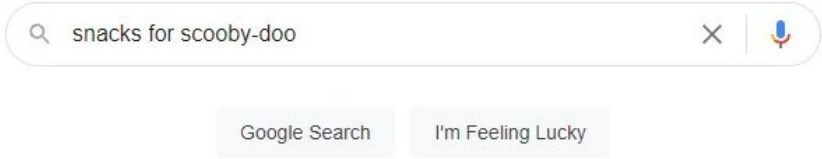
Enhancing distributed tracing to order events

Tanmay Gupta and Anshul Rastogi
Mentor: Dr. Raja Sambasivan,

DISTRIBUTED SYSTEMS TODAY



Distributed systems surround us.

Some examples:

- Google
- Facebook
- Cellular networks*

Distributed systems are

Networks of devices/machines ("nodes"), such as computers or servers that communicate with one another to complete tasks

What is DISTRIBUTED TRACING?

Distributed tracing is when a software tracks the flow of service requests in a distributed system

This is useful for:

- Debugging
- Regulation
- Analyzing performance

https://www.scaleyourapp.com/difference-between-centralized-decentralized-distributed-systems-explained/

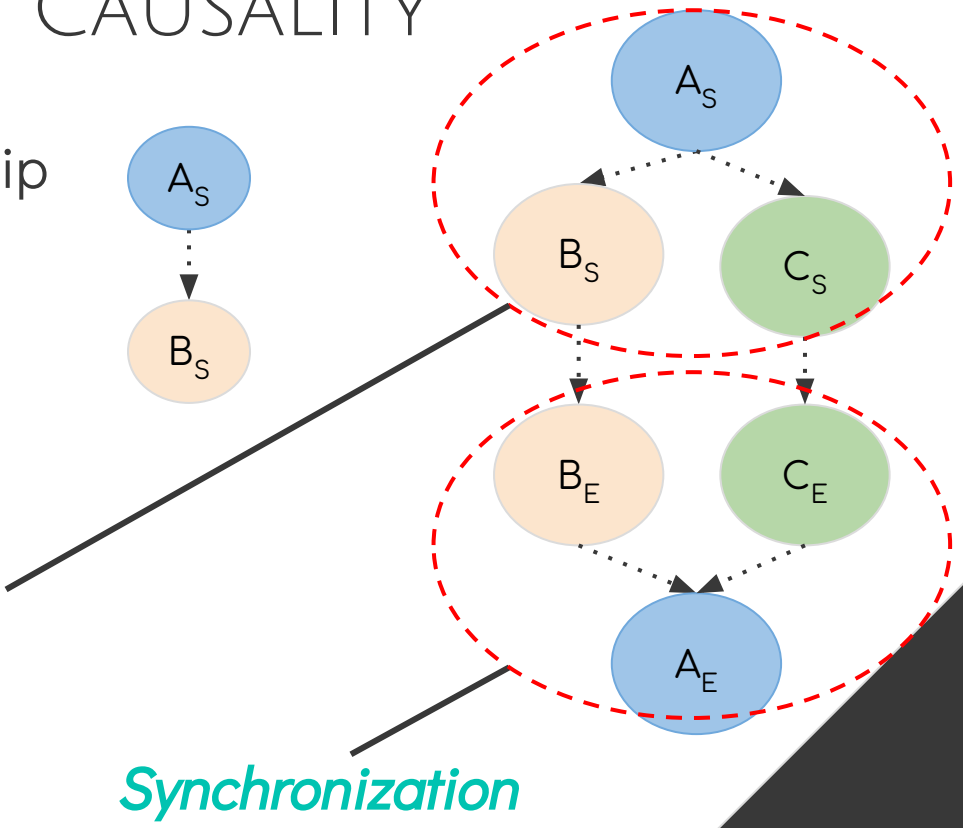
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36356.pdf

https://www.uplers.com/wp-content/uploads/2020/04/Top-5-Debugging-Tools-Your-Front-end-Developers-Should-Have-Hands-On.jpg

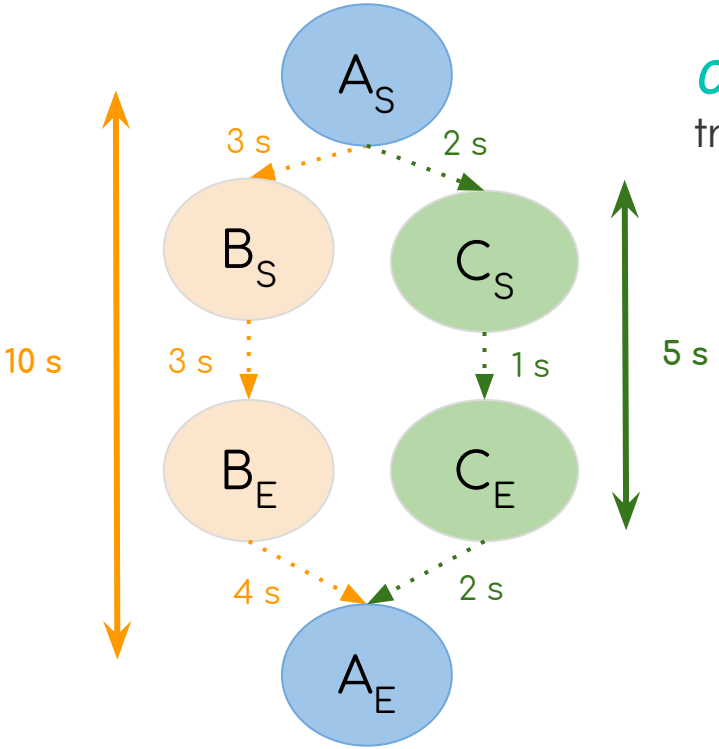
The Ideal Trace: Causality

- *Happens-before* relationship
  - A_s must occur for B_s to occur

- *Concurrent* relationship
  - Events could happen in any order or simultaneously



The Ideal Trace: Applications

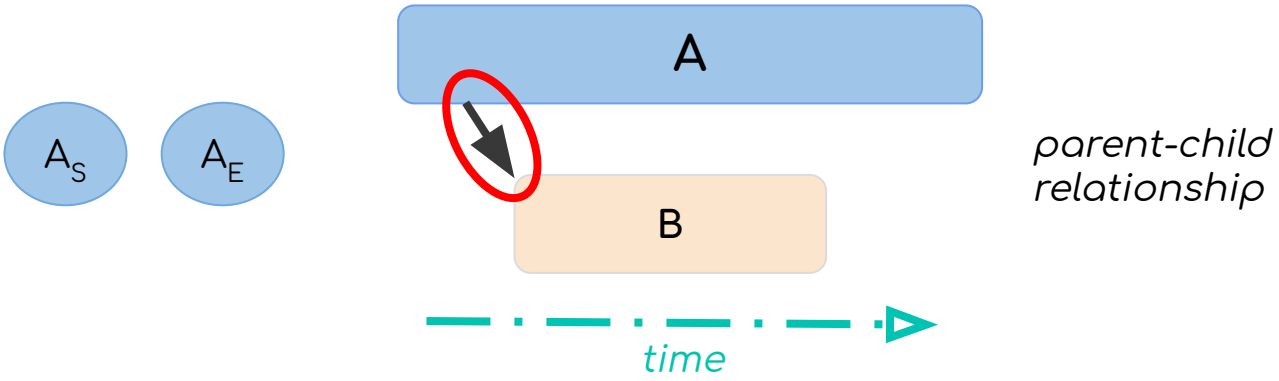


Critical Path: longest path in trace

Tracing: Spans

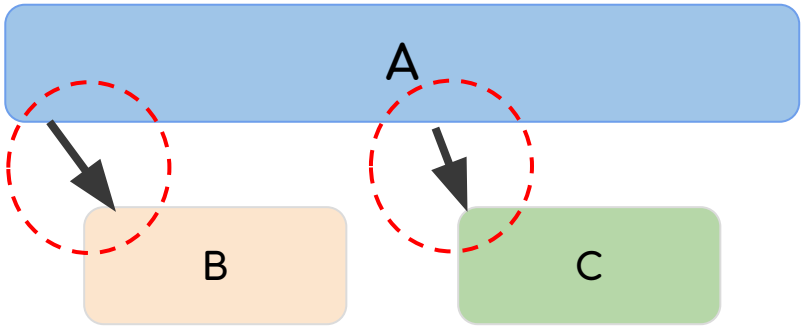
Span (definition):

> *"The 'span' is the primary building block of a distributed trace, representing an individual unit of work done in a distributed system."*
>
> *– OpenTracing.io*



Tracing: The Challenges

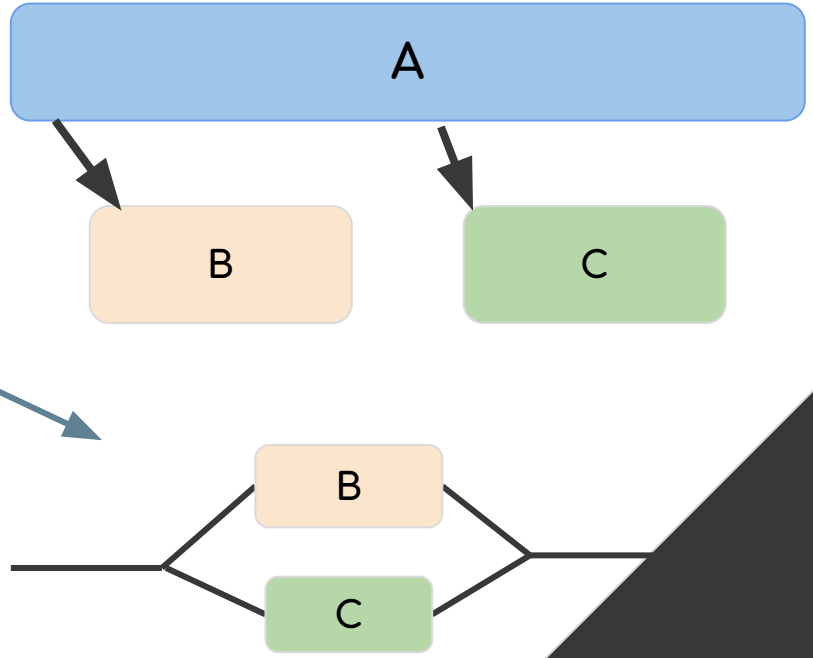
- Caller-callee relationships
  - B dependent on A
  - C dependent on A



- **Is C dependent on B?**

Tracing: The Challenges

- Is C dependent on B?
  - Instrumentation could help
    - Fork in threads => concurrent
    - Hard to do with heterogeneous systems



CAN WE USE BIG DATA TO GET CLOSER TO THE IDEAL TRACE?

Overview

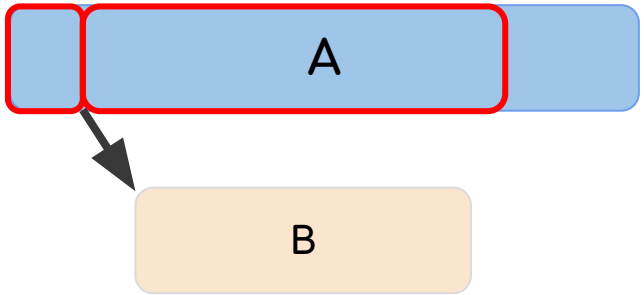
01 Distributed Tracing

02 *The Mystery Machine*

03 *Scooby Systems*

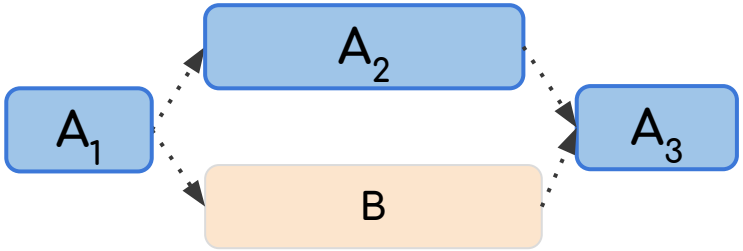
04 Future Work

Spans, segments, & Events



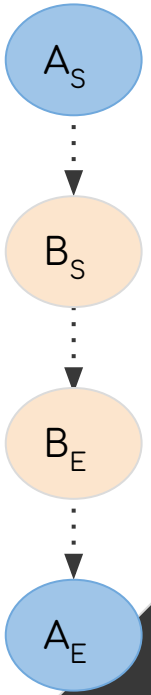
Spans

- Many relationships between parts of A and B
  - E.g. asynchronous concurrency missed



Segments

- Traces missing logs



Events

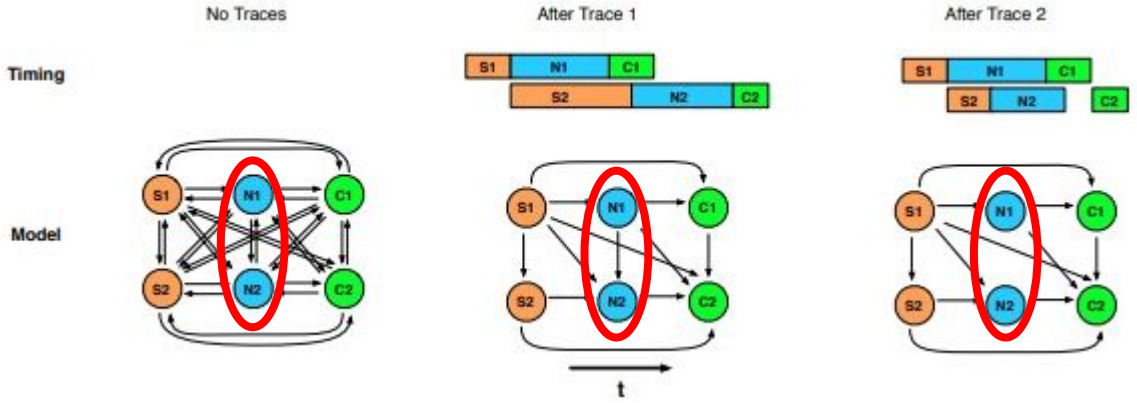
The Mystery Machine: GCM

The Mystery Machine produces a **Global Causal Model** (**GCM**)

- Uses a *segment-based model*.
- shows happens-before dependencies between every segment across the traces such that the dependencies hold for *every trace*.

The Mystery Machine: How it Works

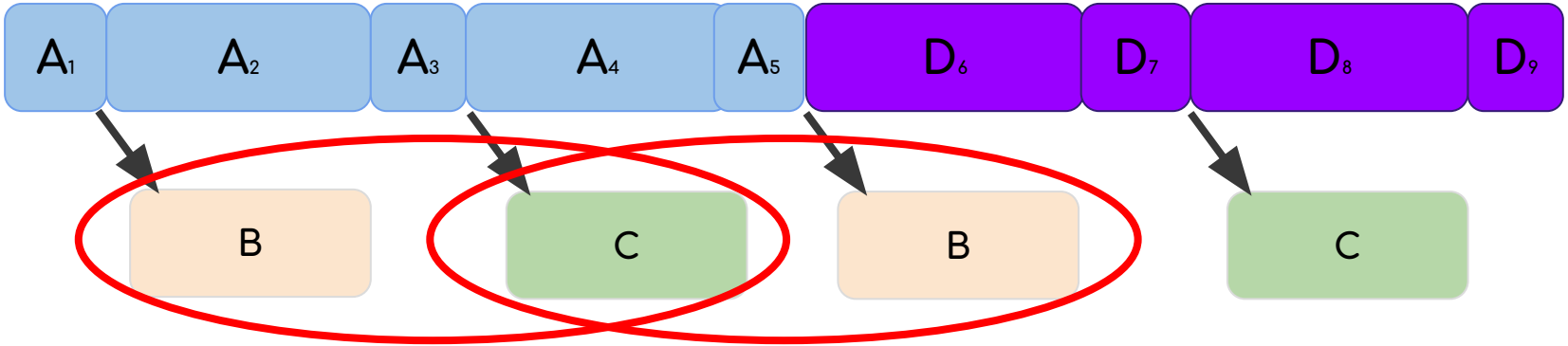
- First creates all hypothetical happens-before edges between all segments to create GCM
- Then takes happens-before relationships across the traces and removes violated edges from GCM
- Reaches final GCM once it has iterated through all traces



The Mystery Machine: Limitations

- **Problem:** Assumes enough natural variation
- *The Mystery Machine* assumes Facebook-specific things
- **Problem:** Assumes that all traces are correct, leaving no room for error such as:
  - Clock skew
  - Anomalies in structure (caused by bugs)

The Mystery Machine: Limitations



- **Problem:** Mystery Machine assumes that there are no repeats of the same segment or event
- GCM starts with interconnected B and C
  - Sees `B -> C`; removes `C -> B` from model
  - Sees `C -> B`; removes `B -> C` from model

But what if `B -> C` is the true structure?

Overview

01 Distributed Tracing

02 *The Mystery Machine*

03 *Scooby Systems*

04 Future Work

User-Defined Threshold

- Address **rigidity** of *The Mystery Machine*
  - Don't eliminate edge after just one violation

$s_{A \to B}$ = # of successes of A→B
$v_{A \to B}$ = # of violations of A→B
$u_{A \to B}$ = A, B do not both occur

- Chose **success-based model** contrary to *The Mystery Machine*
  - Counts successes of each relationship
  - User has freedom to choose threshold in terms of $s_{A \to B}$ and $v_{A \to B}$

$$s_{A \to B} = s_{A \to B}$$
$$v_{A \to B} = s_{B \to A} + q_{A \to B}$$
$$u_{A \to B} = T - s_{A \to B} - v_{A \to B}$$

A and B at same time

Total # of occurrences

Scooby Systems

- Preprocessing
  - Create Causal Model
- Main Algorithm
  - for each trace
    - for each pair of spans
      - update causal model
- Apply Threshold
  - For each relationship in causal model
    - if (passes threshold) add edge to final model

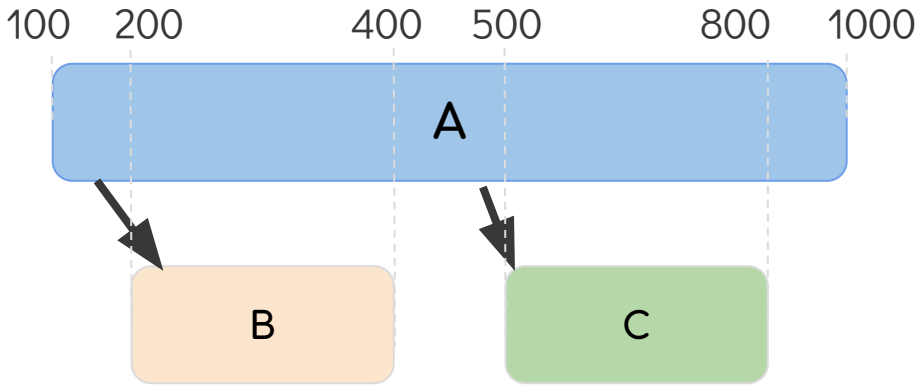
Scooby Systems

Scooby Systems in Action

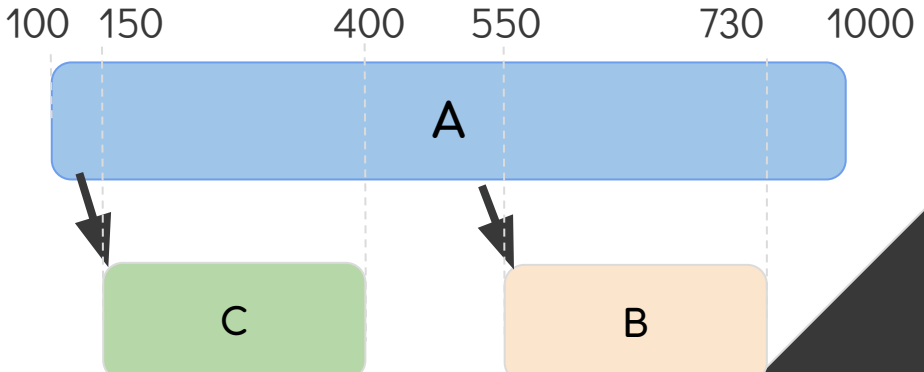
Input

Trace ID: "1"

Timestamps (ms):

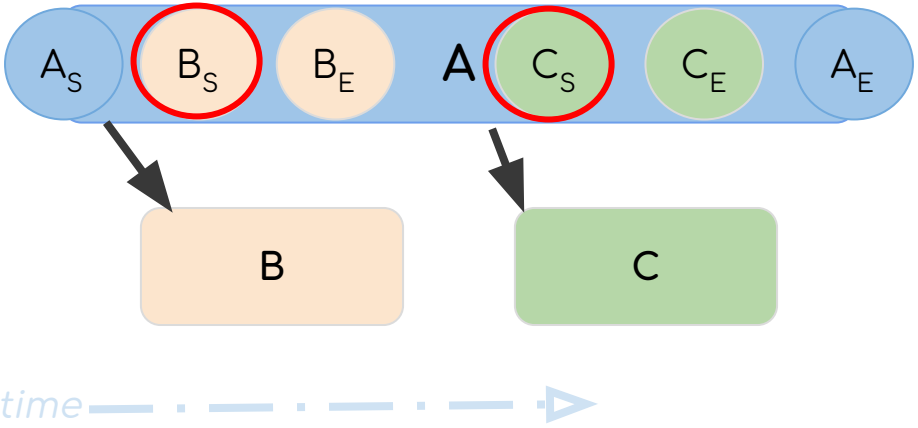


Trace ID: "2"



Scooby Systems in Action

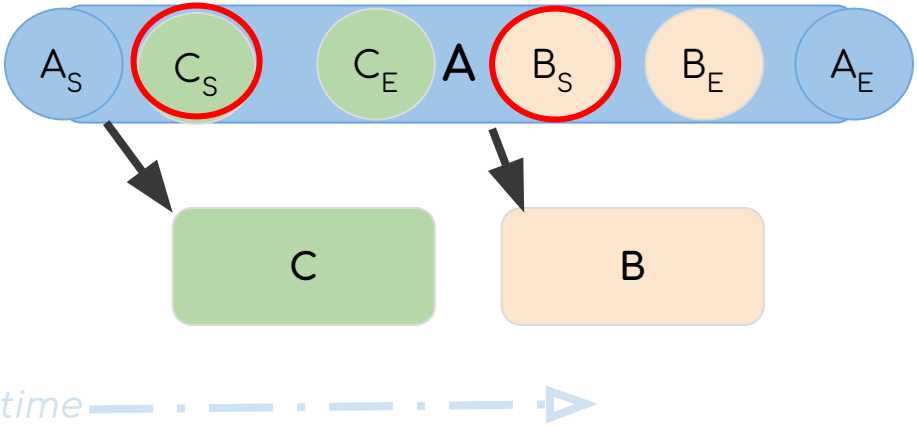
Trace 1



	A_S	A_E	B_S	B_E	C_S	C_E
A_S	0	1	1	1	1	1
A_E	0	0	0	0	0	0
B_S	0	1	0	1	1	1
B_E	0	1	0	0	1	1
C_S	0	1	0	0	0	1
C_E	0	1	0	0	0	0

Scooby Systems in Action

Trace 2



	A_S	A_E	B_S	B_E	C_S	C_E
A_S	0	1	1	1	1	1
A_E	0	0	0	0	0	0
B_S	0	1	0	1	0	0
B_E	0	1	0	0	0	0
C_S	0	1	1	1	0	1
C_E	0	1	1	1	0	0

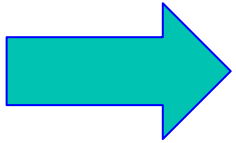
Scooby Systems in Action

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 & 1 & 1 \\ 0 & 2 & 0 & 0 & 1 & 1 \\ 0 & 2 & 1 & 1 & 0 & 2 \\ 0 & 2 & 1 & 1 & 0 & 0 \end{bmatrix}$$

THRESHOLD

$$\begin{bmatrix} 0 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 2 & 1 & 1 \\ 0 & 2 & 0 & 0 & 1 & 1 \\ 0 & 2 & 1 & 1 & 0 & 2 \\ 0 & 2 & 1 & 1 & 0 & 0 \end{bmatrix}$$

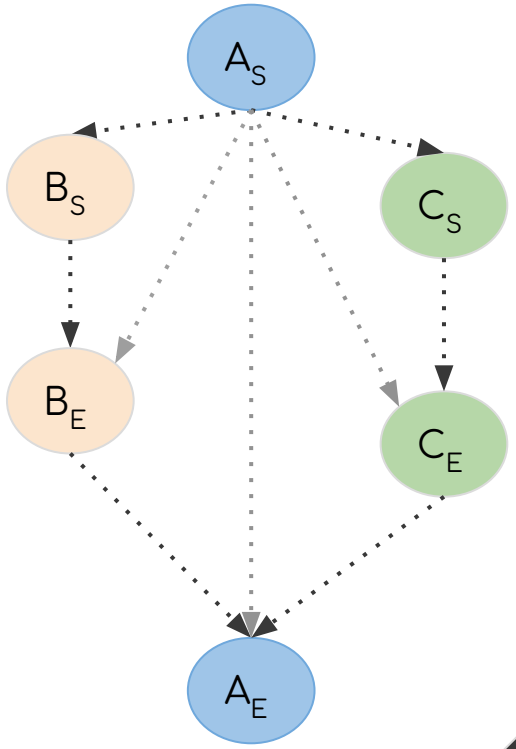
s/T > 90%



$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

GCM Visualized

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Overview

01 Distributed Tracing

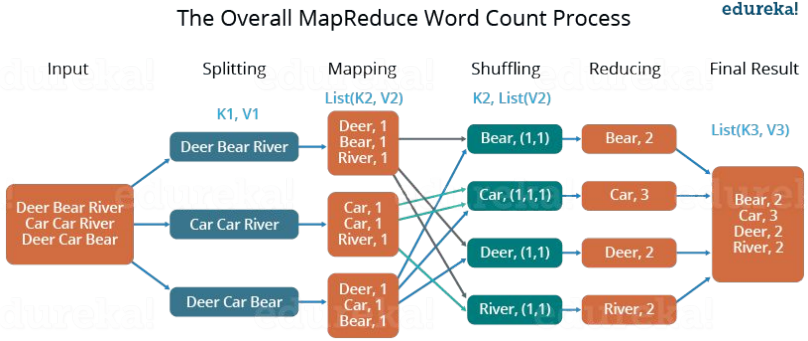
02 *The Mystery Machine*

03 *Scooby Systems*

04 Future Work

Next Steps

- Implement *Scooby Systems* in Hadoop
  - Scalable
- Threshold-based determination of edges
- Proposing solutions / further analysis of *Mystery Machine* limitations
- Evaluation
  - DeathStarBench



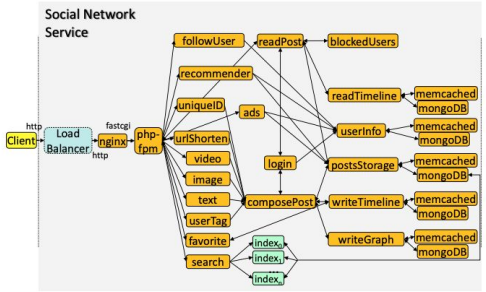


Figure 4. The architecture (microservices dependency graph) of *Social Network*.

ACKNOWLEDGEMENTS

Thanks to **Darby Huye**, **Max Liu**, **Raja Sambasivan**, & D.O.C.C. Lab for their guidance

Thanks to the **PRIMES Program** for providing this opportunity

Thanks to our **families**

Thank **YOU** for listening!

Any Questions?