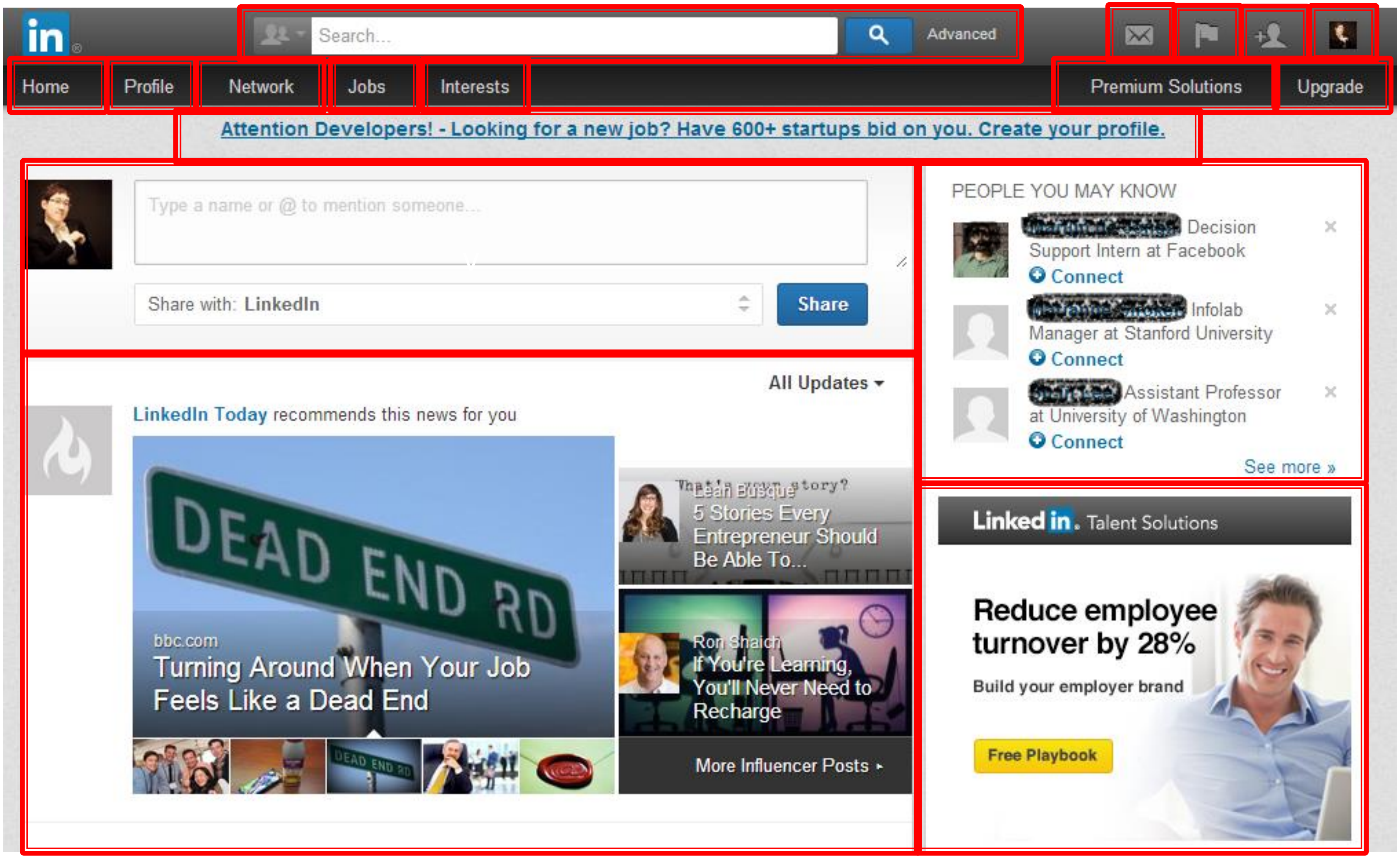
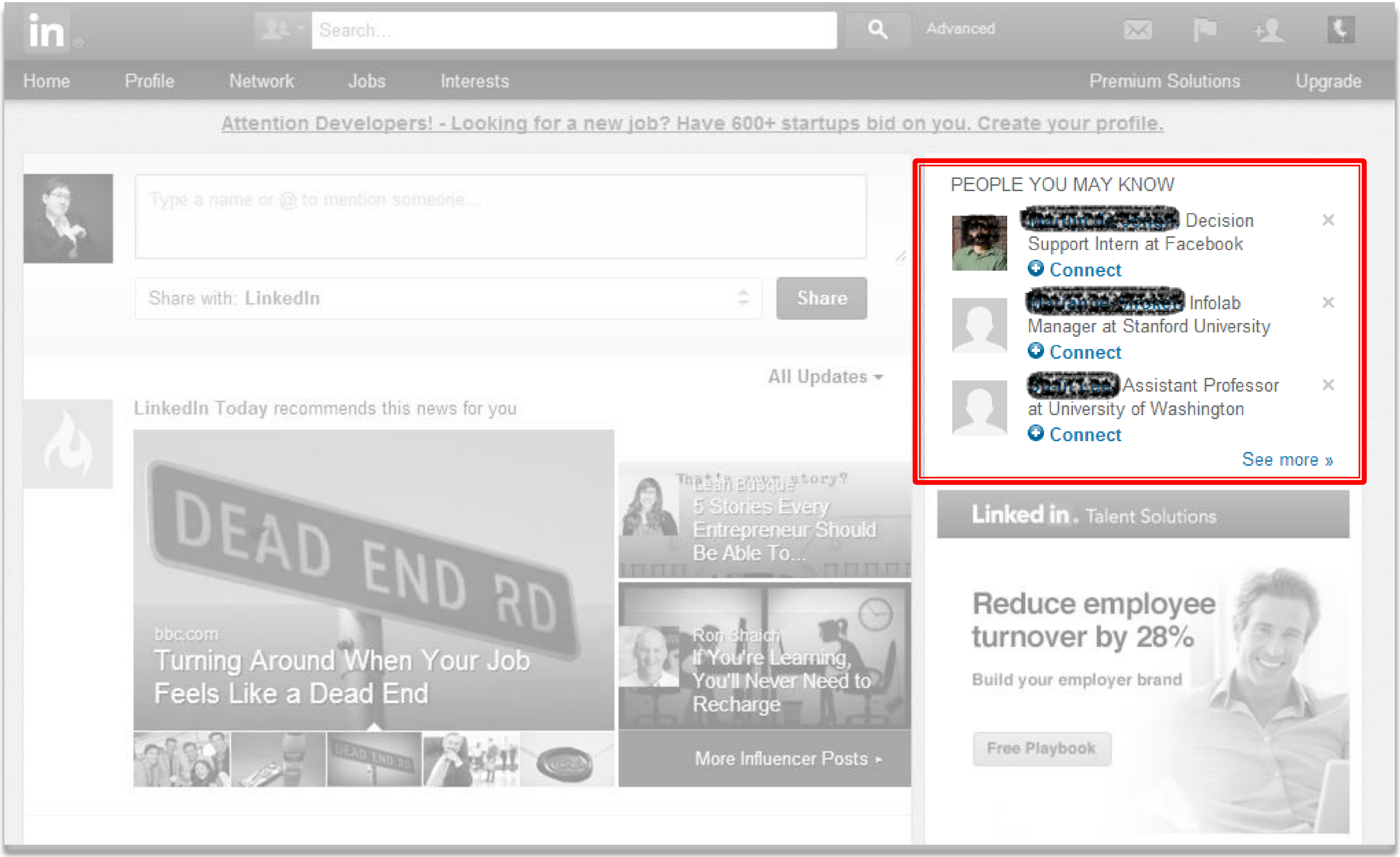
Root Cause Detection in a Service-Oriented Architecture

Myunghwan Kim (Stanford University)
Roshan Sumbaly (LinkedIn Corp.)
Sam Shah (LinkedIn Corp.)

Service Oriented Architecture

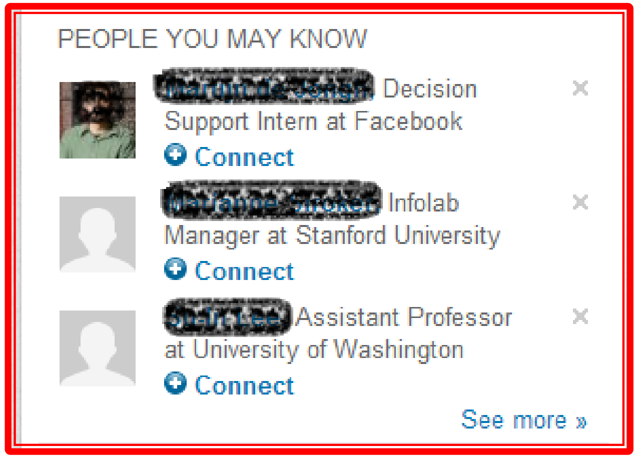


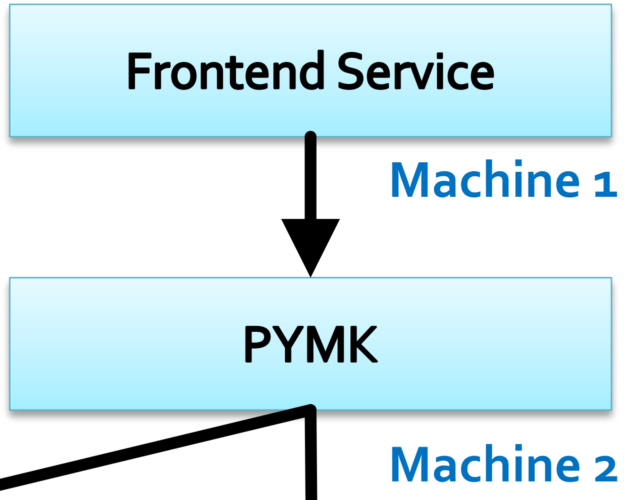
Service Oriented Architecture

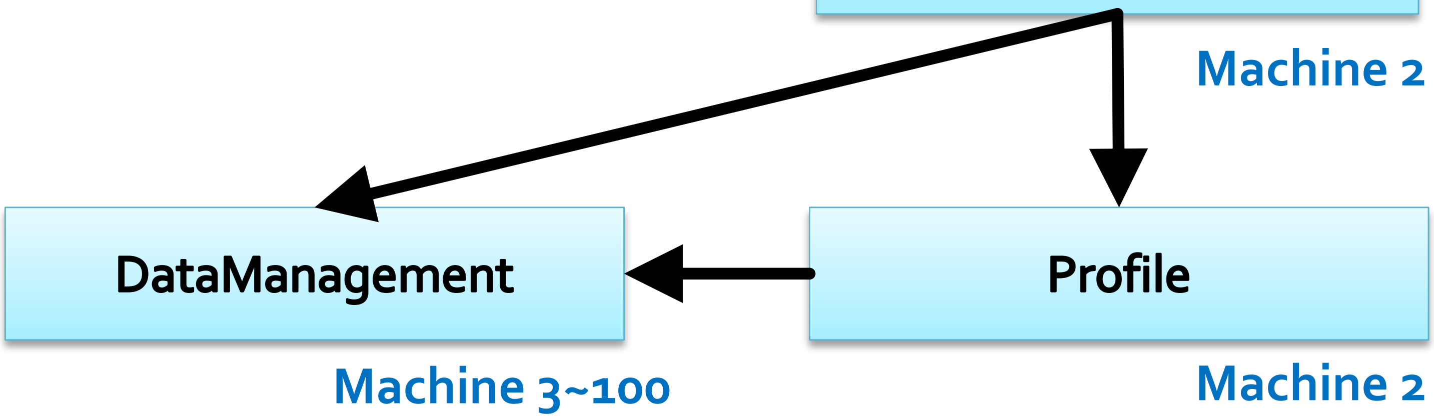


Service Oriented Architecture

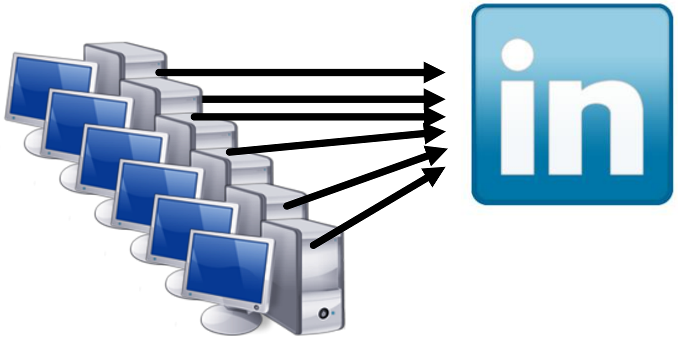
In this web architecture, a service is the atomic unit of functionality. Such an architecture allows easy abstraction and modularity for implementation and reuse, as well as independent scaling of components.



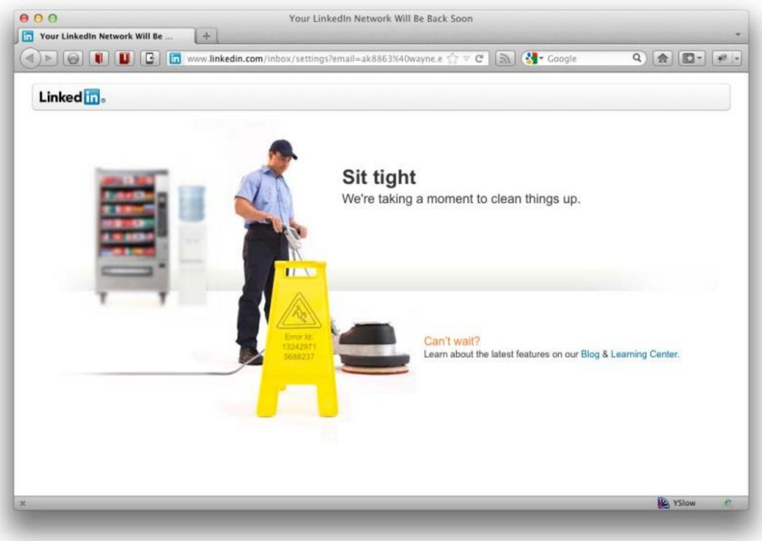




Anomalies Occur !!

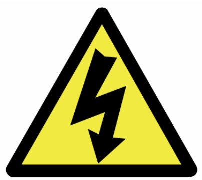


- Service Request Increase



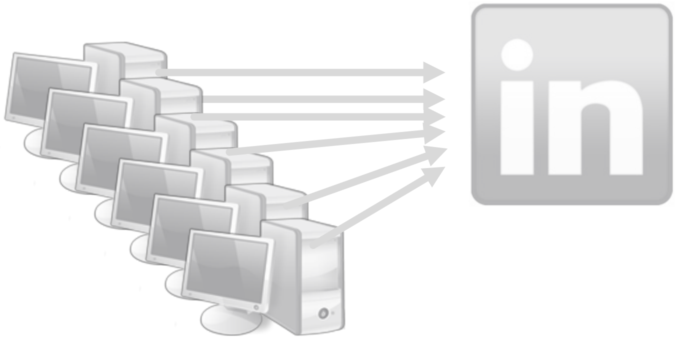
- Software Bug / Error

- Overload: High latency

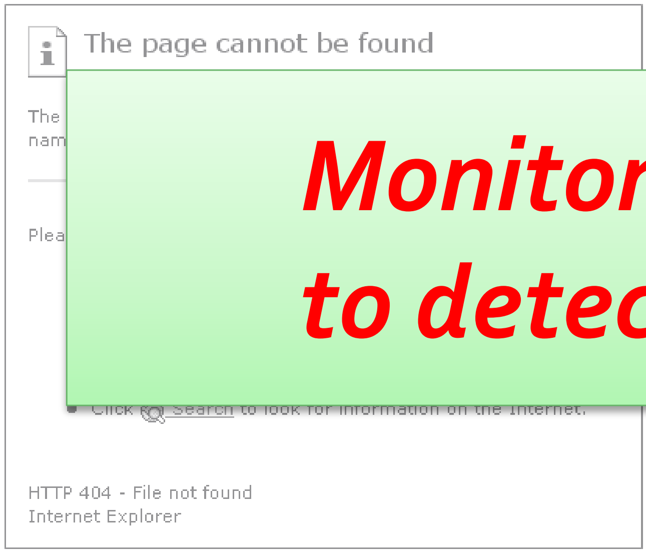


- Server Outage

Anomalies Occur !!



- Service Request Increase



- Software Bug / Error

Monitoring teams are dedicated to detect any anomalies (24 hrs)

- Server Outage

Root Cause Detection

- Anomalies can be detected by
  - Monitoring team
  - Alarms set up by some rules
  - Smart anomaly detection algorithms, etc.
- What is the **next step** once an anomaly is detected?
  - **Root Cause Detection**: Find the root cause of the anomaly
  - Then, fix it properly
- *Root Cause Detection* problem
  - Site availability is revenue impacting, so **short recovery time** is critical
  - Finding the root cause may consume **engineers' efforts**

Goal: ***Minimize resource for finding root causes***

Challenges in Root Cause Detection

- Conventional approach
  - Check metrics (e.g., throughput, latency) of each service week over week
  - Analyze error logs delivered by API
  - May use domain knowledge to narrow down search space

- However,
  - Hard to maintain perfect logging system
  - Hard to keep up-to-date domain knowledge of fast evolving system
  - Time consuming to check metrics of many services large number of <service, API>
  - Time consuming to analyze the logs of many services

Challenges in Root Cause Detection

- Conventional approach
  - Check metrics (e.g., throughput, latency) of each service week over week
  - Analyze error logs delivered by API
  - May use domain knowledge to narrow down search space

- However,
  - Hard to maintain perfect logging system
  - Hard to keep up-to-date domain knowledge of fast evolving system
  - **Time consuming to check metrics of many services**
  - **Time consuming to analyze the logs of many services**

Can we provide candidate services to look at?

Problem Formulation

Input

Anomaly Information
- Time
- Anomalous service
- Anomalous metric

Metrics
- Anomalous metric for services

Call-graph
- Caller-callee between services
- Directed & unweighted

Output

Ranked List of Services
- The order of services to investigate

Example

1. DataManagement
2. PYMK
3. Profile

...

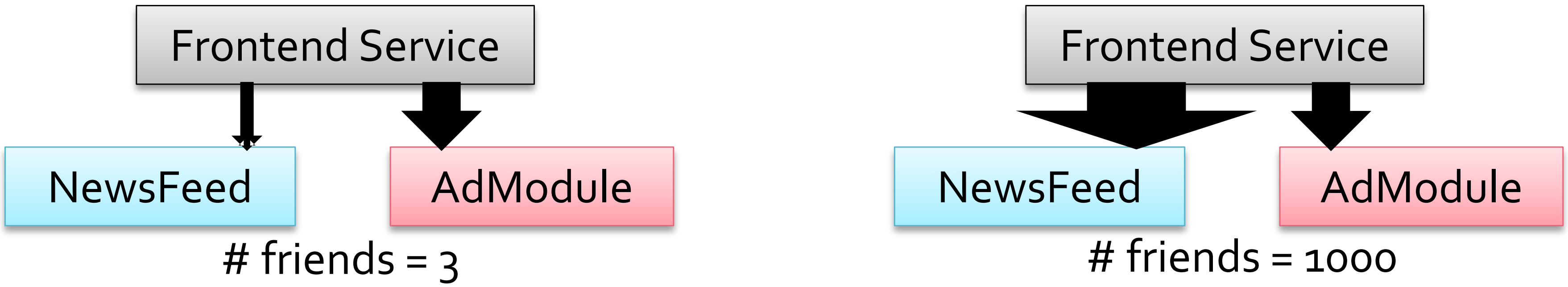
Considerations

- **Unsupervised problem**
  - Labeled data is hard to collect
  - Labeled data in the past may not be valid now
    - Software bugs have been fixed
    - Performance of bottleneck service has been improved
    - New deployed service has caused high latency
- **Fast algorithm**
  - Provide the output (ordered list) online
  - Do not join API metadata online to obtain call trees
- **Call graph ≠ 100% dependency graph**
  - Call graph does not necessarily indicate dependency, the path of anomaly propagation

Call Graph: Partial Information

The user with 1000 connections may have a bottleneck on the update generation sensor, while for a user with just 3 connections all updates might be cached and the bottleneck would instead be the profile information fetching sensor.

- Different task size
  - For example, when showing status update of my friends, the task depends on # of my friends & their update frequency.



- External factor
  - Colocation: Colocated services might show similar anomalous behavior due to hardware failure, ***regardless of the call graph***
  - Malicious user requests

An edge X -> Y in the call graph does not mean that Y propagates an anomaly to X

Our Approach

- Three Key Components
  - **Pattern Similarity**
    - Root cause service would show similar anomalous behavior with the service where an anomaly is detected with respect to a certain metric
  - **External factor finding**
    - Assumption: If some external factor exists, all the affected services would show similar anomalous behaviors
  - **Randomized algorithm on the call graph**
    - Deterministic algorithm may be hard to apply for the call graph that does not necessarily represent dependency

- **We propose MonitorRank**
- **MonitorRank improves PR@5 by about 30% on average**

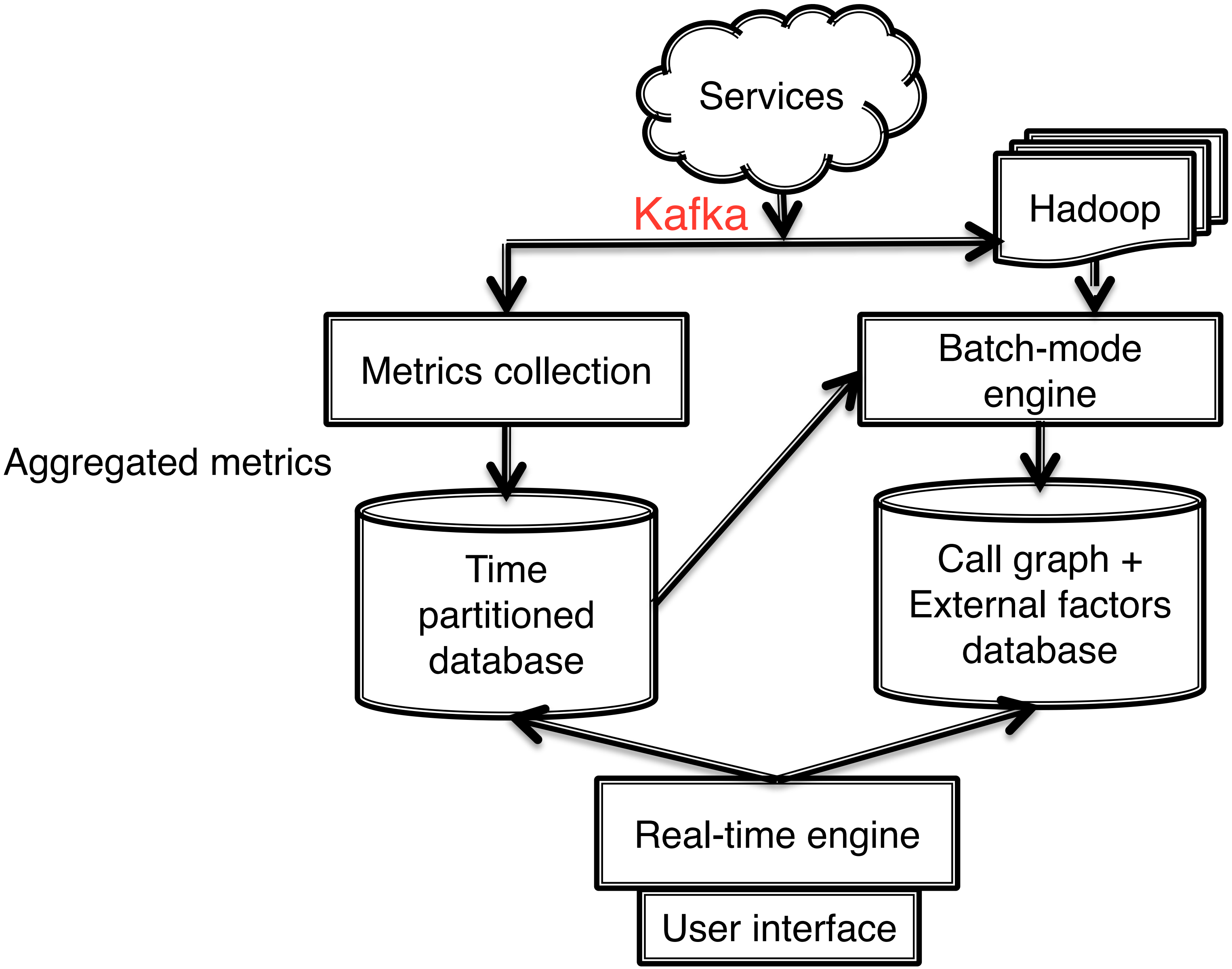
Precision at top K indicates the probability that top K sensors given by each algorithm actually are the root causes of each anomaly case.

Our Approach

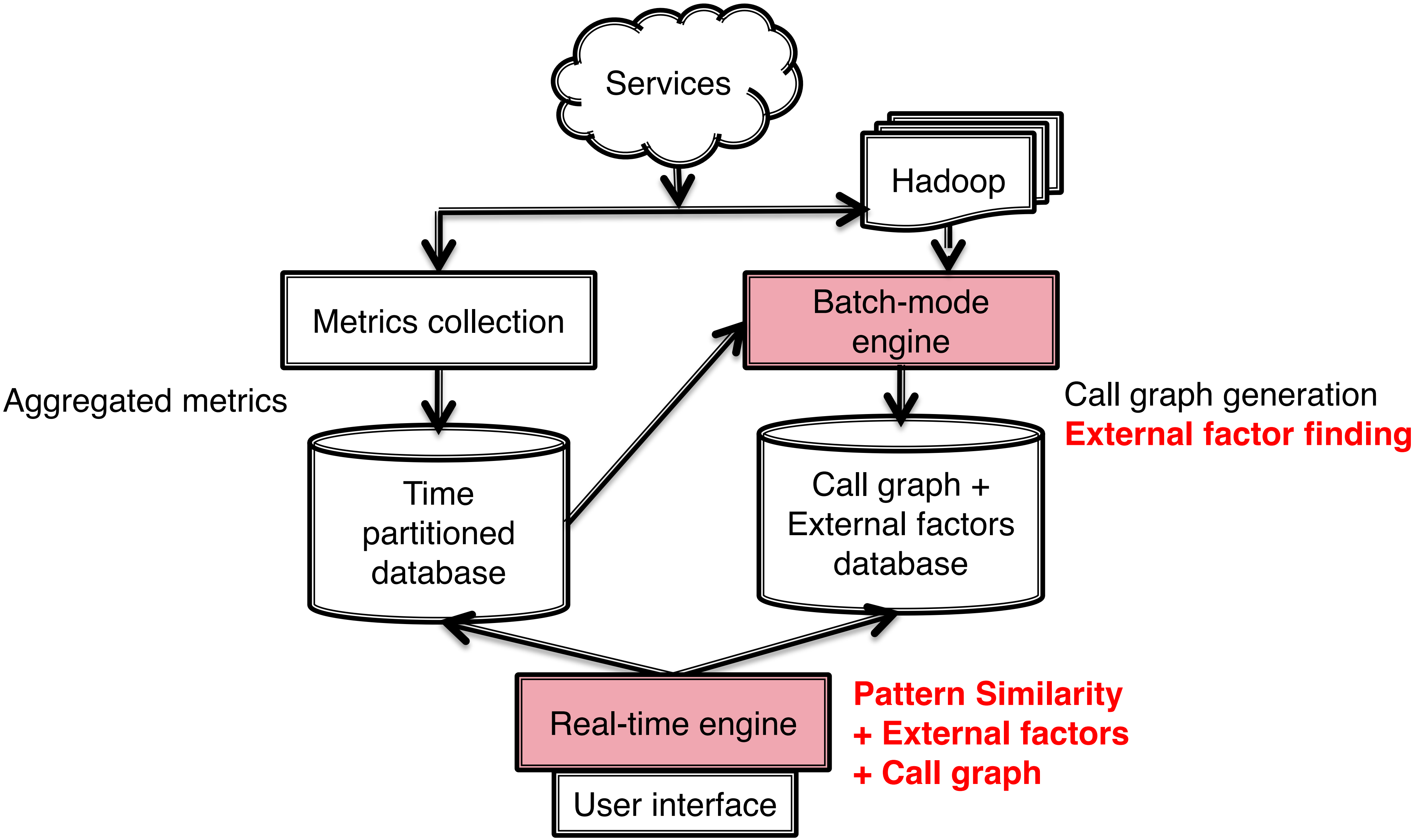
- Three Key Components
  - **Pattern Similarity**
    - Root cause service would show similar anomalous behavior with the service where an anomaly is detected with respect to a certain metric
  - **External factor finding**
    - Assumption: If some external factor exists, all the affected services would show similar anomalous behaviors
  - **Randomized algorithm on the call graph**
    - Deterministic algorithm may be hard to apply for the call graph that does not necessarily represent dependency

- **We propose MonitorRank**
- **MonitorRank improves PR@5 by about 30% on average**

Our Framework



Our Framework



External Factor Finding

- Runs periodically (for example, bi-weekly)
- Goal: Find services that usually show **similar anomalous behaviors** in some metric even though they are not connected in the call graph —> group those metrics that often become abnormal at the same time even though they are not connected in the call graph
- Q: ***What is "anomalous behavior" in the unlabeled data?***
  - ***Pseudo-anomaly***: Moment showing a sudden rise / drop in a given metric
  - ***High recall*** is favorable, and low precision is OK

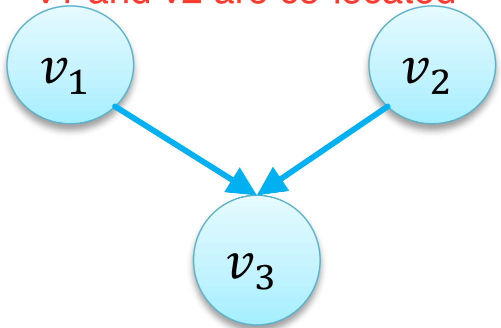
recall=tp/(tp+fn)
precision=tp/(tp+fp)

- Q: ***What is "similar behavior"?***
  - ***Pattern similarity function*** between two services given a time window
- **Our solution: Pseudo-anomaly Clustering Algorithm**

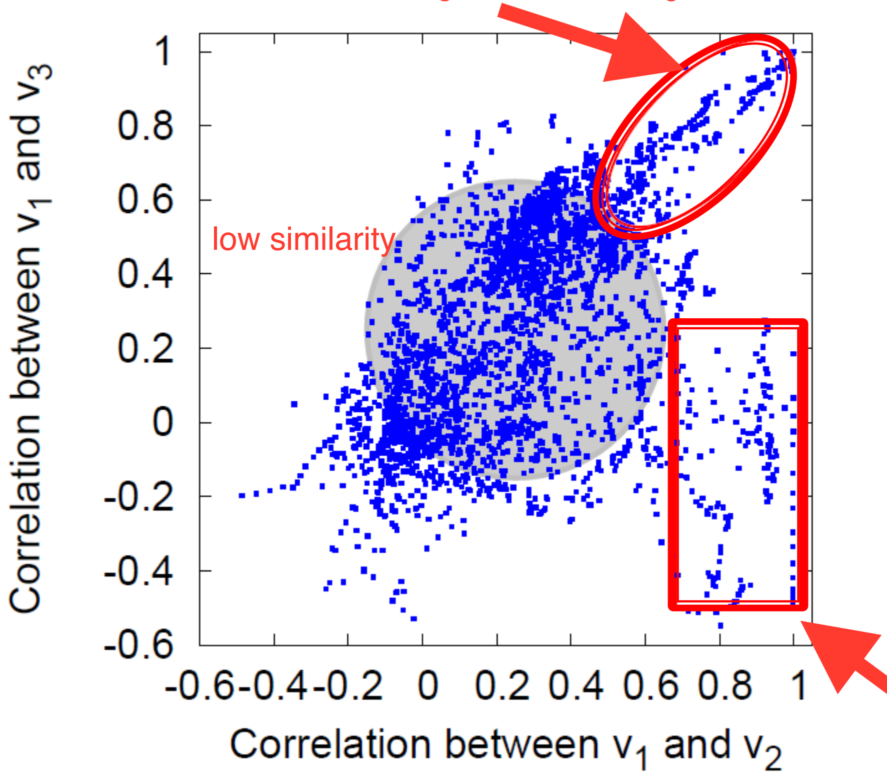
Pseudo-anomaly Clustering

This is for finding the external factors of a given service's given metric

v1 and v2 are co-located



because v1 and v2 were correlated through a common neighbor sensor v3



Due to the co-location of v1 & v2,
the correlation between v1 and v2 can be high even when the correlation between v1 and v3 is low

A separate clustering problem for each frontend sensor

- Find **pseudo-anomalies** on every user-facing service and metric for a certain time period
- For given service and metric, find **a set of services** representing **high pattern similarity with the given service** at each pseudo-anomaly

If sensors are affected by the same external factor, their pattern similarity scores with regard to the seed sensor will be close and high.

- Select such sets with **a certain support**
- **Remove sets** if the services end up with **a common leaf node** in the call graph
- Mitigates false positives by threshold-based pseudo-anomaly detection

Final Solution: MonitorRank

- By taking user input, returns the ordered list of services online
  - Input: anomalous service, time, and metric Va
  - Ingredients: input, metrics, pseudo-anomaly clusters, and call graph
  - Our solution: **MonitorRank**

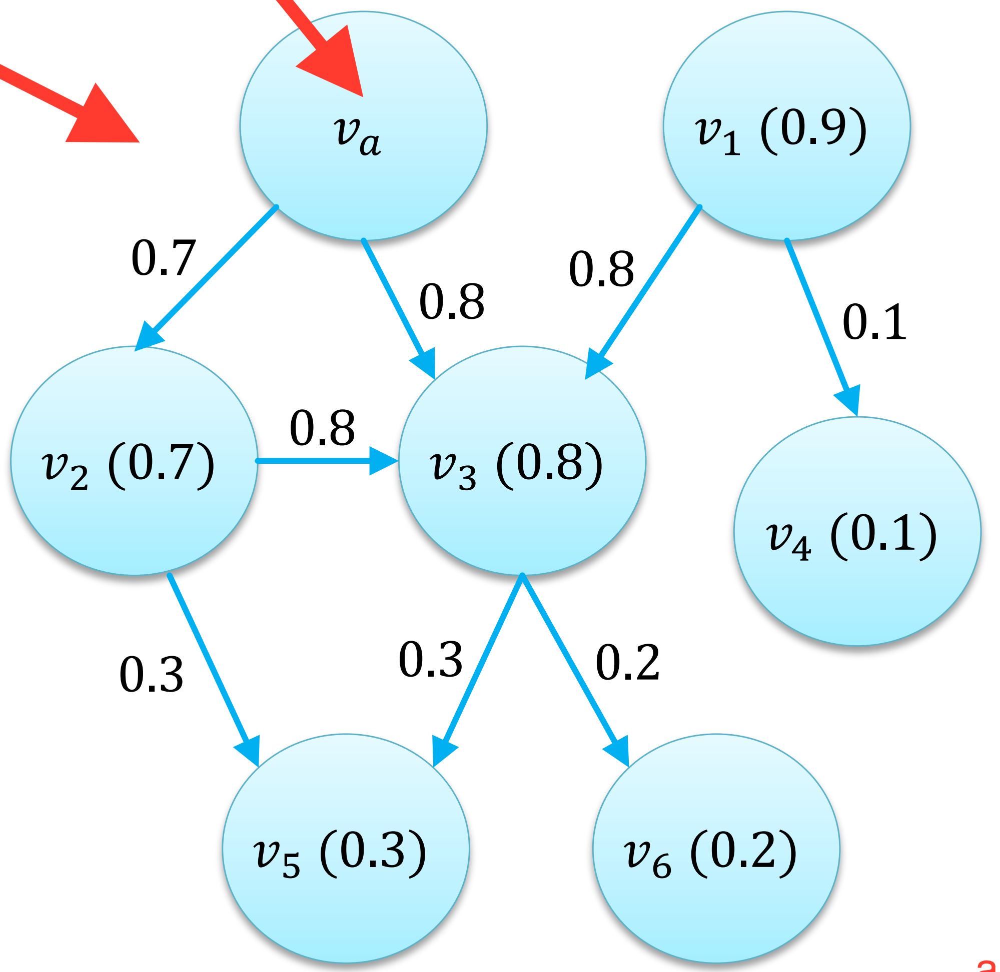
This is the time series similarity between the anomalous service Va with other time series at a particular time window when the anomaly happens.

- **MonitorRank**
  - **Random-walk based** algorithm on the call graph
  - Run a walk **based on pattern similarity between the anomalous service** for a given metric and anomaly time
  - **Personalized PageRank**
    - **Random jump** also **based on pattern similarity with the anomalous service**
  - In the end, a single score (**root cause score**) is given to each service
    - Represents the stationary **distribution** that **non-experts are investigating a certain service**

MonitorRank

the anomalous service

Call Graph

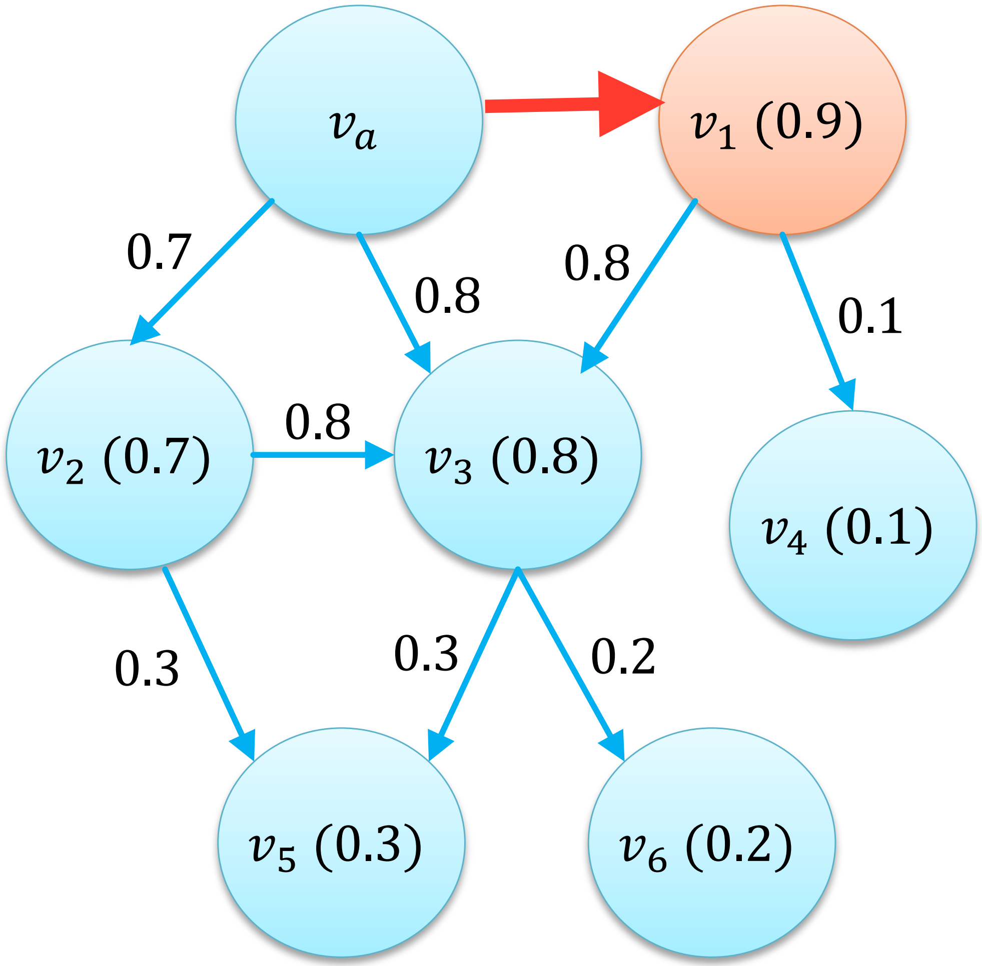


This is the time series similarity between v_i and anomalous service Va's. It signifies the relevance of the service v_i to the given anomaly.

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank

alpha is the probability of random jump, 1-alpha is the probability of random walk

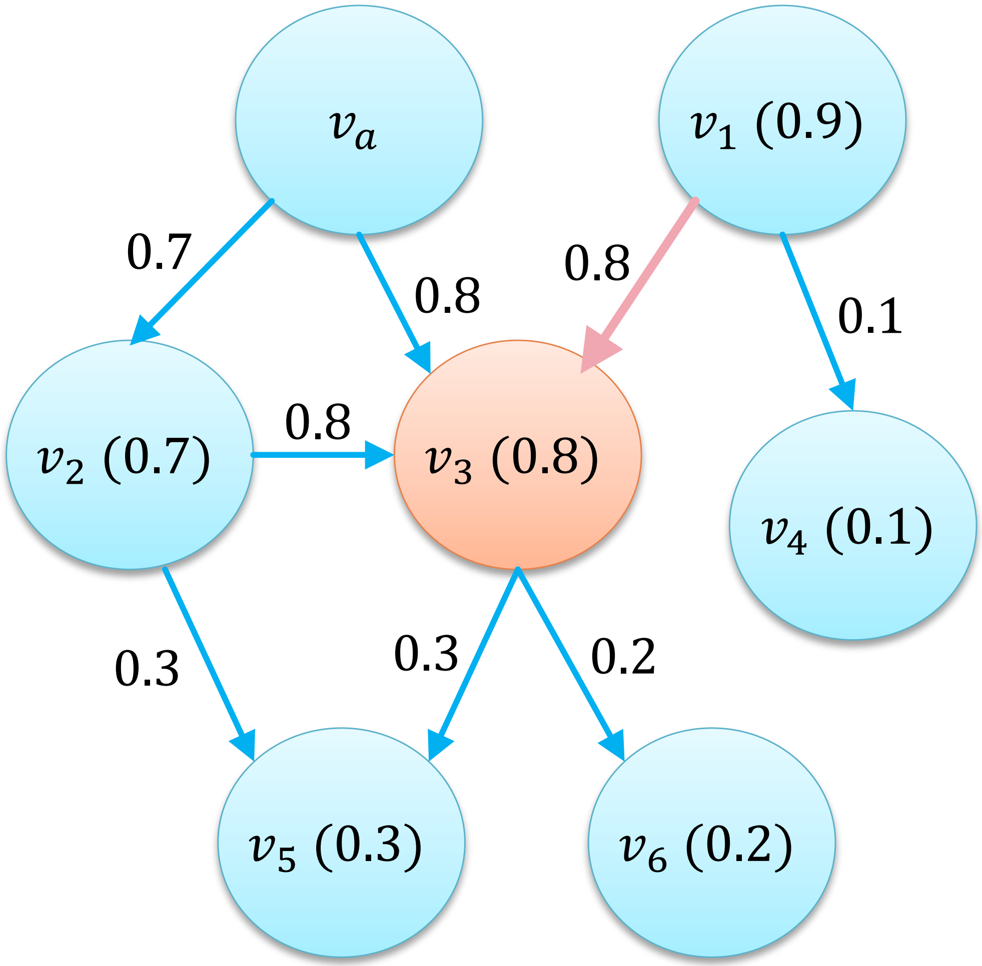


Random jump to a service **proportionally to pattern similarity**

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank

alpha is the probability of random jump, 1-alpha is the probability of random walk



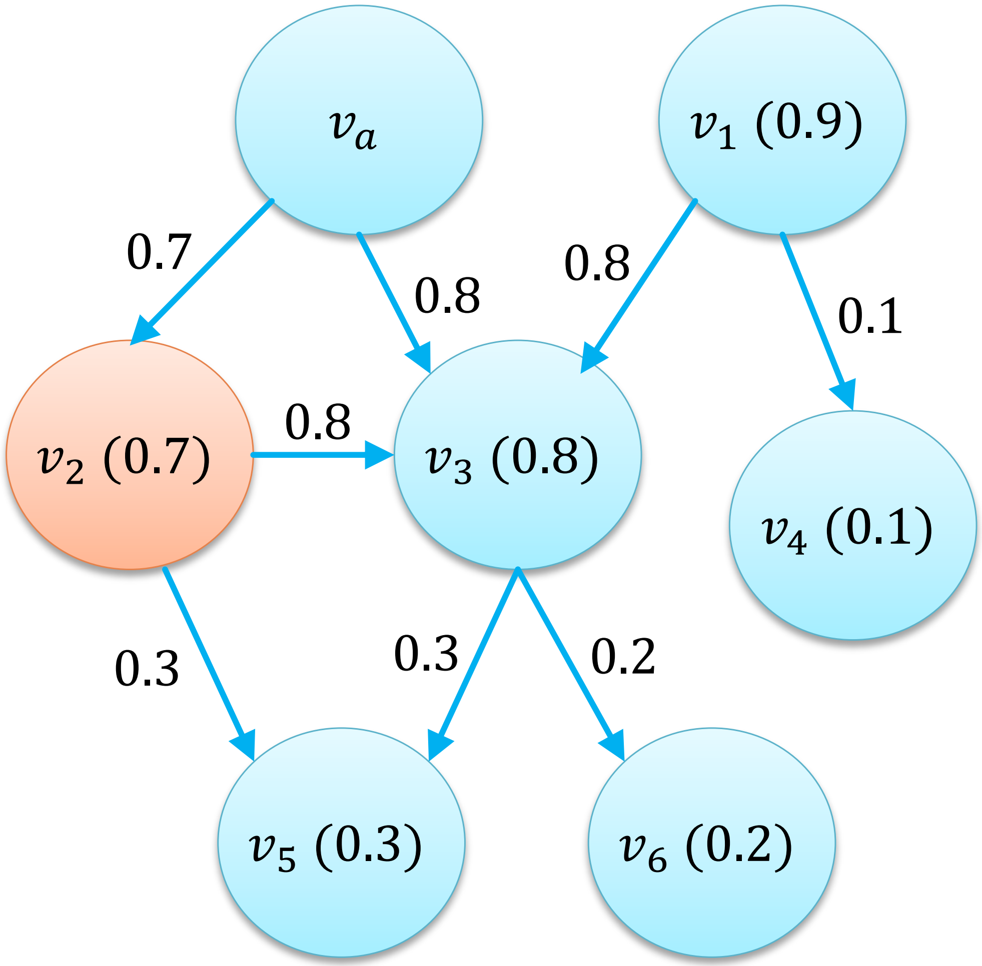
Random jump to a service proportionally to pattern similarity

With probability $1-\alpha$,
walk to a neighbor proportionally to its pattern similarity

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank

alpha is the probability of random jump, 1-alpha is the probability of random walk



Random jump to a service proportionally to pattern similarity

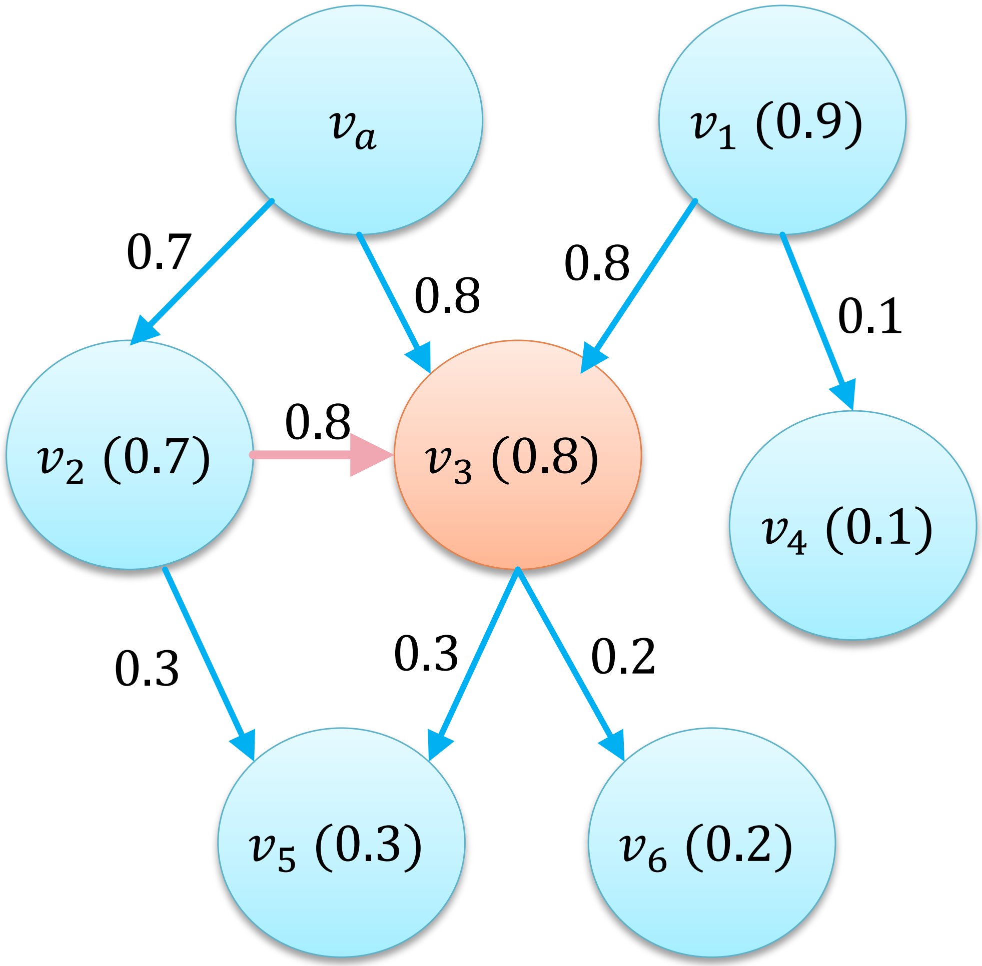
With probability $1-\alpha$, walk to a neighbor proportionally to its pattern similarity

With probability α, **random jump** to a service **proportionally to pattern similarity**

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank

alpha is the probability of random jump, 1-alpha is the probability of random walk



Random jump to a service
proportionally to pattern similarity

With probability $1-\alpha$,
walk to a neighbor
proportionally to its pattern similarity

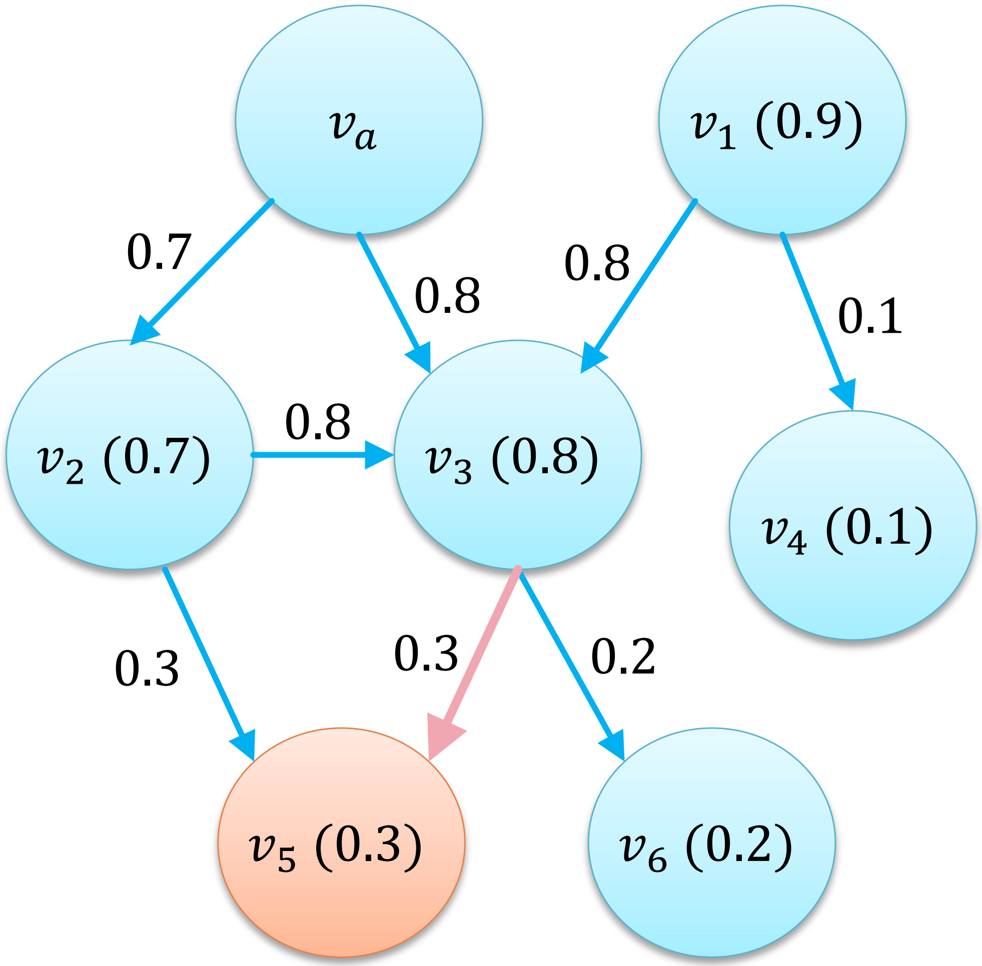
With probability α,
random jump to a service
proportionally to pattern similarity

Repeat this procedure
until convergence

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank

alpha is the probability of random jump, 1-alpha is the probability of random walk



Random jump to a service proportionally to pattern similarity

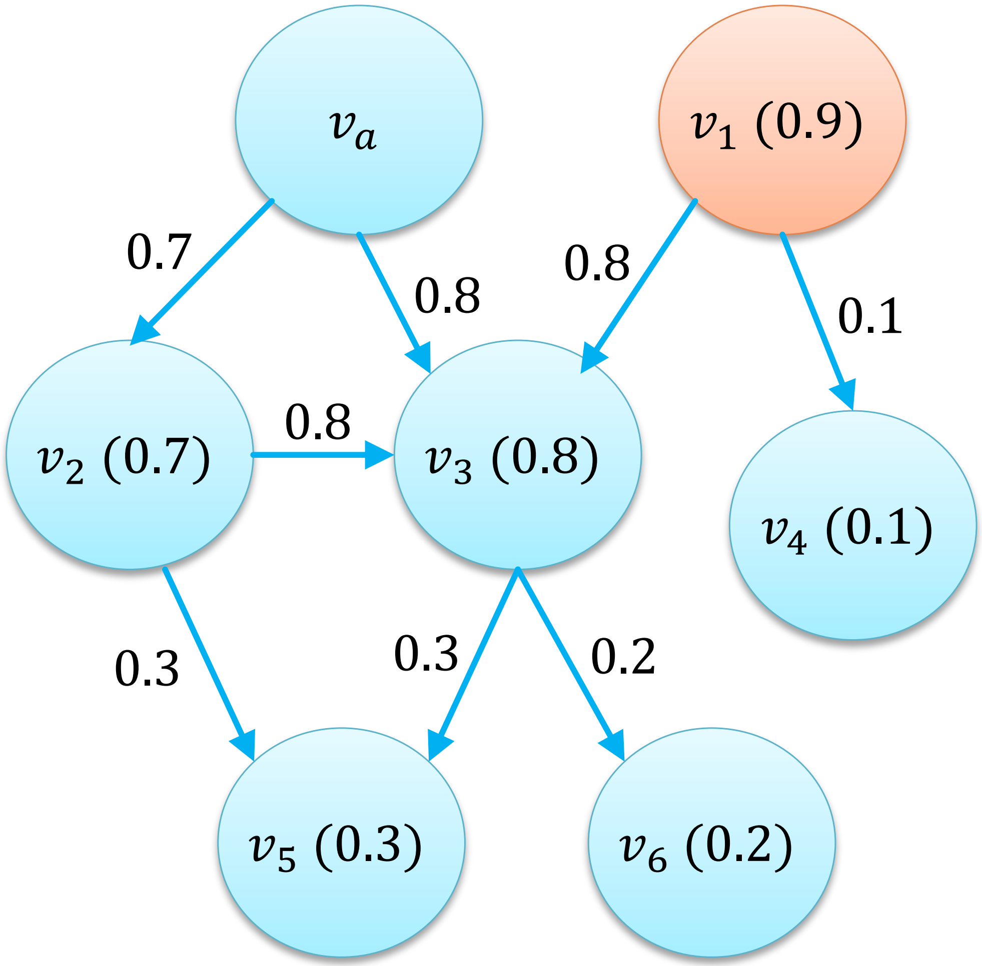
With probability $1-\alpha$, walk to a neighbor proportionally to its pattern similarity

With probability α, random jump to a service proportionally to pattern similarity

Repeat this procedure until convergence

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank



Random jump to a service proportionally to pattern similarity

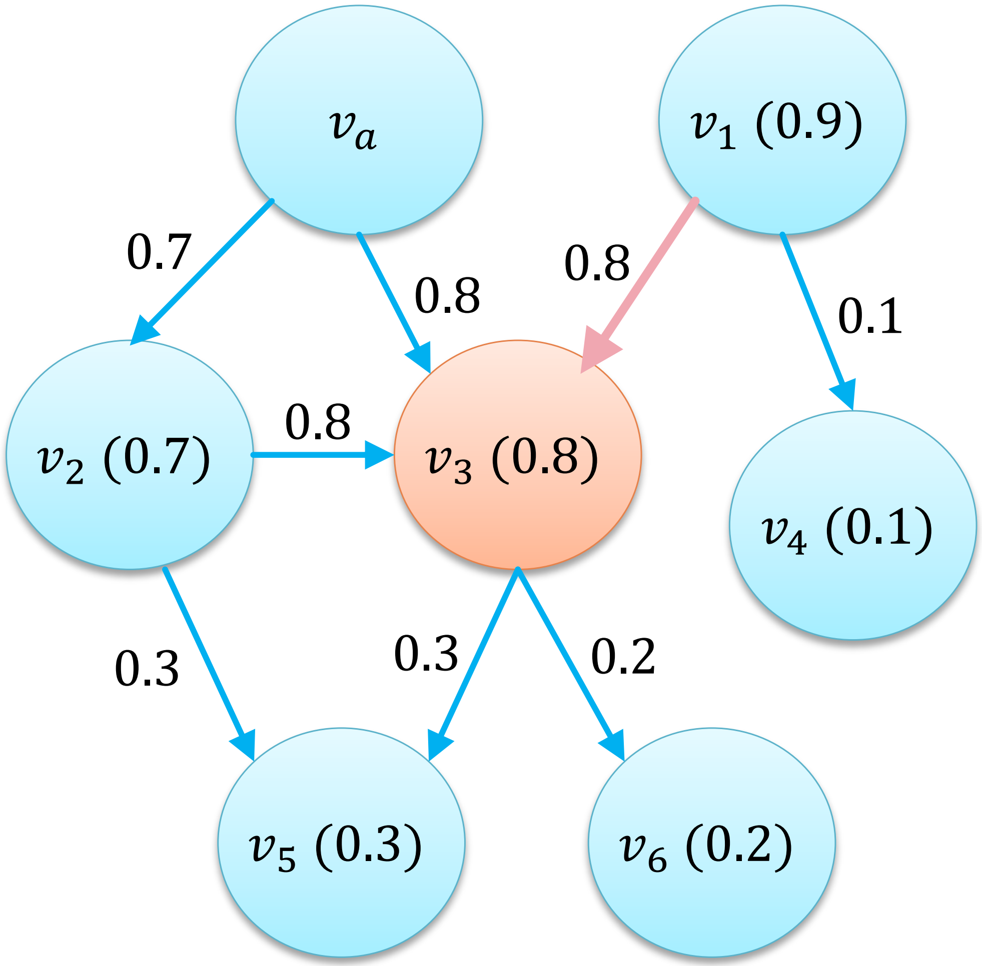
With probability $1 - \alpha$, walk to a neighbor proportionally to its pattern similarity

With probability α, random jump to a service proportionally to pattern similarity

Repeat this procedure until convergence

Numbers in ( ) indicate pattern similarity with v_a

- A node’s root cause score is the probability of visiting each node.
- It represents the stationary distribution that multiple non-experts are investigating a certain service.
- We assume that more visits on a certain node by our random walk implies that the anomaly on that node can best explain the anomalies of all the other nodes.



Random jump to a service proportionally to pattern similarity

With probability $1-\alpha$, walk to a neighbor proportionally to its pattern similarity

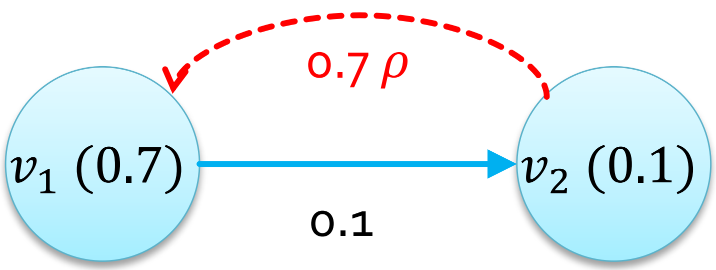
With probability α, random jump to a service proportionally to pattern similarity

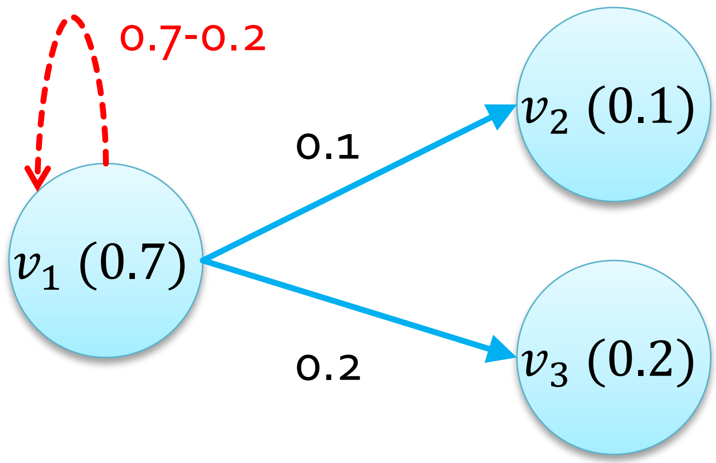
Repeat this procedure until convergence

Numbers in ( ) indicate pattern similarity with v_a

MonitorRank: Details

- Frontend -> Backend direction of call graph
  - Could be trapped into branches with low pattern similarity
  - Solution: Allow **backward edges** with weight multiplied by $\rho < 1$
  - Local exploration (while random jump is global exploration)
- Random-walk enforces moving
  - What if no neighbors represent high pattern similarity?
  - Solution: Add **self edges** with weight subtract by max. pattern similarity of out-going neighbors





MonitorRank: External Factors

MonitorRank blends the pseudo-anomaly clusters with the random walk algorithm by finding the best-match cluster with the current metric data and giving more scores to sensors in the selected cluster.

- How do we combine pseudo-anomaly clusters?
  - Find the best matched clusters given pattern similarity score for each backend service

    If the root cause of the current anomaly is the common external factor of some pseudo-anomaly cluster C, then the pattern similarity scores of sensors in C, would be higher than any sensor not in C

    - Use the criterion $\frac{min.score\ in\ cluster}{max.score\ not\ in\ cluster}$ as each cluster score
    - Find the cluster of maximum cluster score
  - Compute average similarity of services in the selected cluster
  - Add the average similarity to the pattern similarity of each service in the cluster

    v_i on the previous page

  - Do the random-walk as before

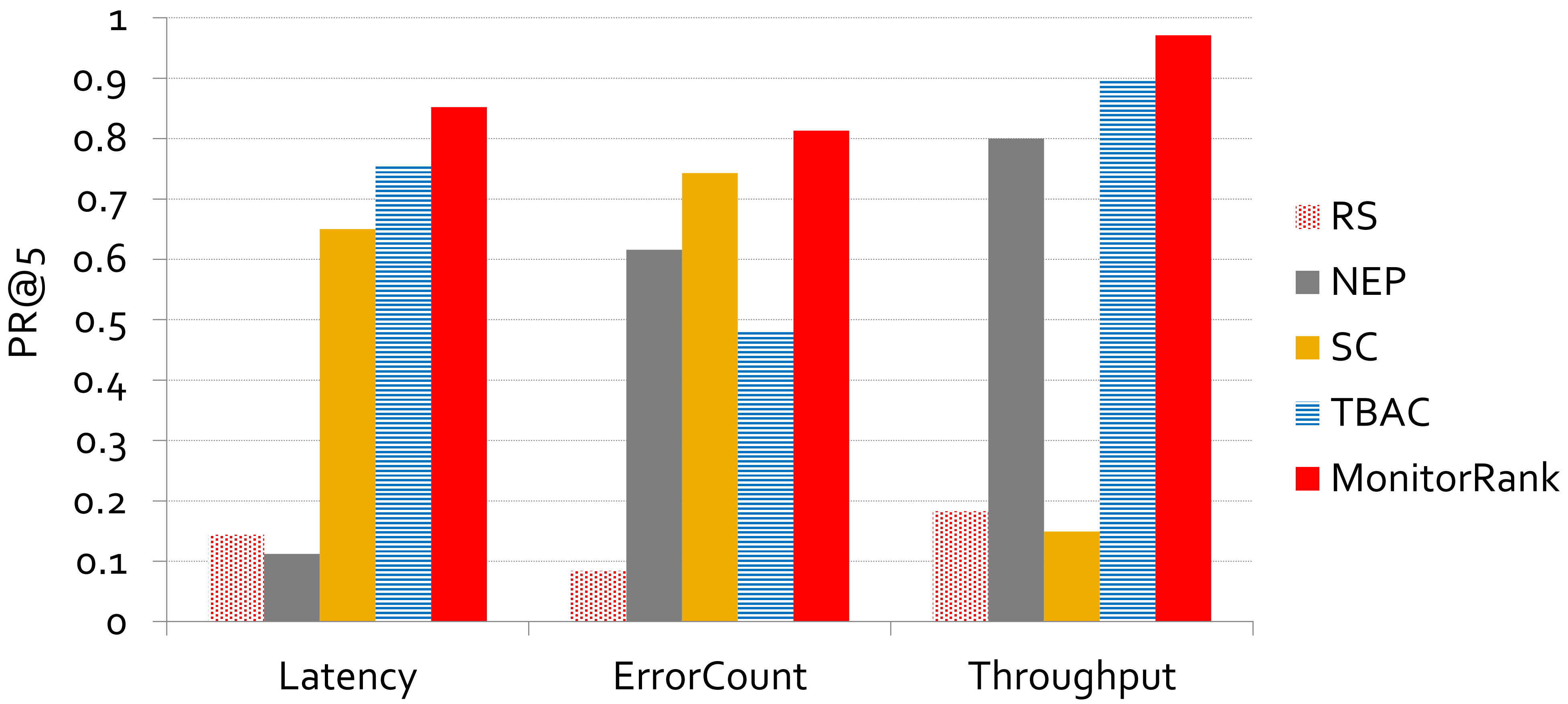
By regarding the average pattern similarity score of sensors in C* as the pattern similarity score of the external factor corresponding to C*, we add this average score to Sc for every sensor c ∈ C∗. In this way, we leverage the fact that engineers in the monitoring team examine the sensors related to the external factor first.

Experiments

- Datasets
  - From LinkedIn site issue management system
    - Latency (25 examples), Error count (71 examples), Throughput (35 examples)
- Baseline methods
  - Random Selection (RS): Pick up services in a random order
  - Node Error Propensity (NEP): Pick up services that produce more errors
  - Sudden Change (SC): Pick up services that show the most change compared to previous time window given a metric
  - Timing Behavior Anomaly Correlation (TBAC) [Marwede et. al.]
    - Correlation + Call graph
    - However, considers the call graph as a dependency graph
- We will see PR@K as evaluation metrics in this talk
  - PR@K = (# of detected root causes) / min(# of root causes, K)

Precision at top K indicates the probability that top K sensors given by each algorithm actually are the root causes of each anomaly case.

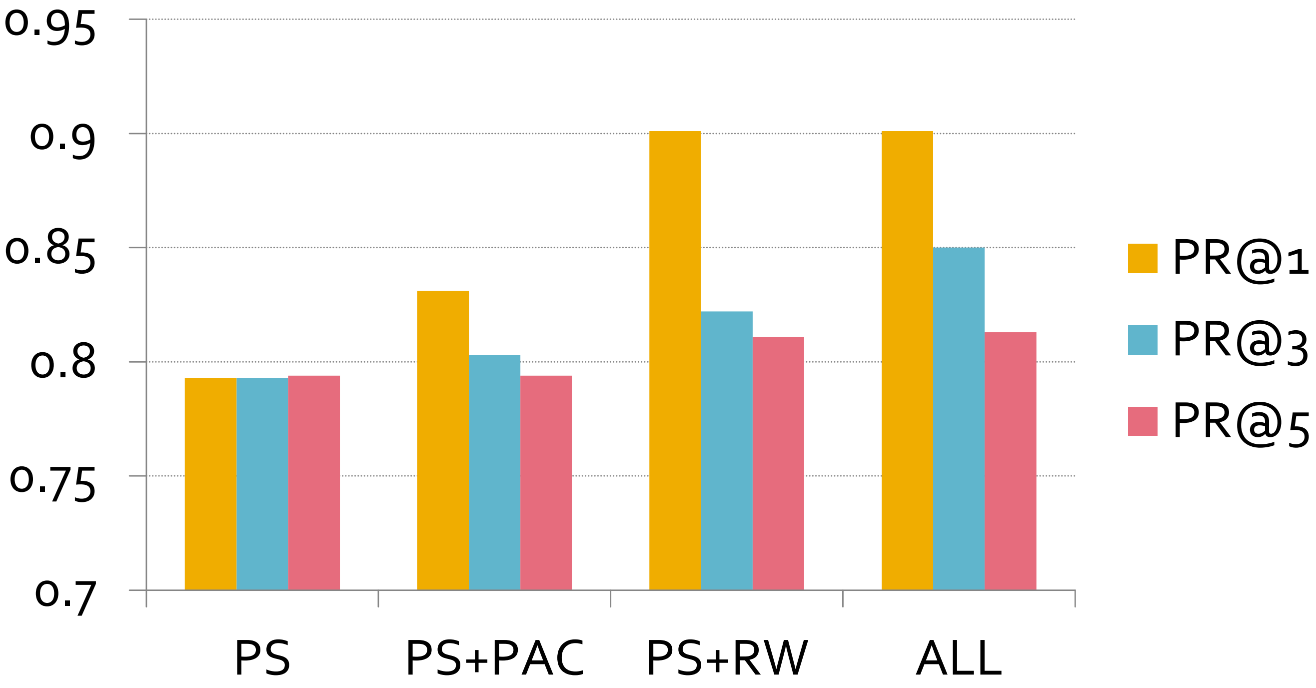
Experiments: Results



MonitorRank outperforms the other methods

Experiments: Subcomponent

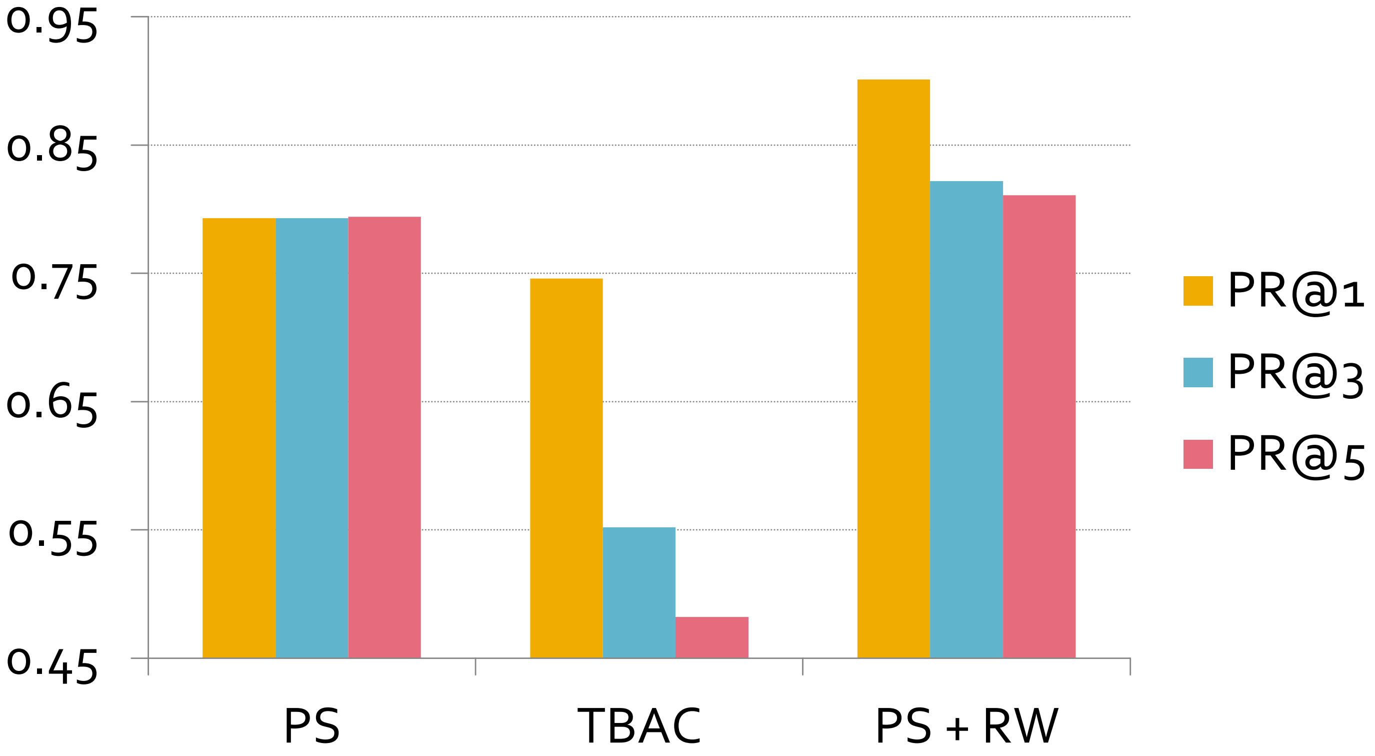
- Use only partial subcomponents
  - Pattern Similarity (PS), Pseudo-anomaly Clustering (PAC), and Random Walk (RW)



Each component plays some role,
while the effect by RW > the effect by PAC

Experiments: Effect by call graph

- Both TBAC and MonitorRank use the call graph
  - Q: Do they use it properly?



Randomized way improves performance
under the circumstance that call graph ≠ dependency

Conclusion

- Build framework to reduce resource for root cause detection of an anomaly in a service-oriented architecture
  - Considers pattern similarity, external factors, and call graph
  - In particular, admits the situation that call graph does not necessarily represent dependency
  - MonitorRank provides improved list of root cause services by about 30% on average in terms of PR@5
  - Randomized algorithm works better than deterministic algorithm on the given call graph situation

- Future work
  - May use weighted call graph given by some metric such as throughput
  - Might be combined with standardized text logs

Related Work

- Troubleshooting
  - Monalytics [Wang et. al. ICAC'11]
  - VScope [Wang et. al. Middleware'12]
- Anomaly detection
  - Subspace method [Mahimkar et. al. CoNext'11]
  - Matrix factorization [Xiong et. al. ICDM'11]
  - Streaming data [Tan et. al. IJCAI'11]
- Root cause detection
  - Supervised algorithm [Ahmed et. al. SysML'07] [Chen et. al. ICAC'04]
  - Anomaly correlation + Graph algorithm
    - [Arefin et. al. ISM'10] [Marwede et. al. CSMR'09] [Jiang et. al. CNSM'10]

Thank you