Towards Scalable Critical Alert Mining

Bo Zong

with Yinghui Wu, Jie Song, Ambuj K. Singh, Hasan Cam, Jiawei Han, and Xifeng Yan

1UCSB, 2LogicMonitor, 3Army Research Lab, 4UIUC
Big Data Analytics in Automated System Management

- Complex systems are ubiquitous
  - Nuclear power plant
  - Computer network
  - Social media
  - Chemical production
  - Software system
  - Aircraft system

- Tons of monitoring data generated from complex systems

- Big data analytics are desired to extract knowledge from massive data and automate complex system management
Massive Monitoring Data in Complex Systems

Example: monitoring data in computer networks

Data center

Monitoring data

@Server-A

# MongoDB backup jobs:

Apache response lag:

Mysql-Innodb buffer pool:

SDA write-time:

120-server data center can generate monitoring data 40GB/day
System Malfunction Detection via Alerts

Example: alerts in computer networks

Alert @server-A
01:20am: #MongoDB backup jobs ≥ 30
01:30am: Memory usage ≥ 90%
01:31am: Apache response lag ≥ 2 seconds
01:43am: SDA write-time ≥ 10 times slower than average performance
...
09:32pm: #MySQL full join ≥ 10
09:47pm: CPU usage ≥ 85%
09:48pm: HTTP-80 no response
10:04pm: Storage used ≥ 90%
...

Complex systems could have many issues

For the 40GB/day data generated from the 120-server data center, we will collect 20k+ alerts/day
Mining Critical Alerts

Example: critical alerts in computer networks

Critical!

Disk Read Latency @Server-A

- #MongoDB backup jobs @Server-B
- CPU cores busy @Server-B
- CPU cores busy @Server-B
- MongoDB busy @Server-B
- Mcollective reg status @Server-C

How to efficiently mine critical alerts from massive monitoring data?
Pipeline

- Offline dependency rule mining
- Online alert graph maintenance
- On-demand critical alert mining

Our focus

[0, 1, ..., 1, 1]
[1, 1, ..., 1, 0]
[0, 0, ..., 1, 1]

History alert log

Dependency rules

Incoming alerts

Alert graph

user

On-demand critical alert mining
Alert Graph

- Alert graphs are directed acyclic (DAG)
- Nodes: alerts derived from monitoring data
- Edges:
  - Indicate the probabilistic dependency between two alerts
  - Direction: from one older alert to another younger alert
  - Weight: the probability that the dependency holds
- Example

$\Pr(C|A) = 0.9$ means $A$ has probability 0.9 to be the cause of $C$
Gain of Addressing Alerts

- If alert $u$ is addressed, alerts caused by $u$ will disappear.
- Given a subset of alerts $S$ are addressed, $p(u|S)$ is the probability that alert $u$ will disappear.
  $$p(u|S) = 1 - \prod_{v \in \text{parent}(u)} (1 - p(v|S) \cdot p(u|v))$$
- Given a subset of alerts $S$ are addressed, $\text{Gain}(S)$ quantifies the benefit of addressing $S$.
  $$\text{Gain}(S) = \sum_{u \in V} F(S, u)$$
  
  - $F(S, u)$ quantifies the impact from $S$ to alert $u$.
  - If $F(S, u) = p(u|S)$, $\text{Gain}(S)$ is the expected number of alerts will disappear given alerts in $S$ are addressed.

The cause of $u$ disappears given $S$ is addressed.
Critical Alert Mining

- **Input**
  - An alert graph $G = (V, E)$
  - $k$, #wanted alerts

- **Output**: $S \subseteq V$ such that
  - $|S| = k$
  - $\text{Gain}(S)$ is maximized

**Related problems**

- Critical Alert Mining is not #P hard as Influence Maximization, since alert graphs are DAGs
- Bayesian network inference enables fast conditional probability computation, but cannot efficiently solve top-$k$ queries
Naive Greedy Algorithm

- **Greedy search strategy**
  - Alert graph $G$
  - Find the alert $u$ such that $S \cup \{u\}$ has the largest incremental gain

- **Greedy algorithms have approximation ratio $1 - \frac{1}{e}$** ($\approx 0.63$)

- **Efficiency issue: time complexity $O(k|V||E|)$**

How to speed up greedy algorithms?
Bound and Pruning Algorithm (BnP)

- Pruning unpromising alerts by upper and lower bounds

Alert graph G

- Drawback: pruning might not always work

Can we trade a little approximation quality for better efficiency?
Single-Tree Approximation

- If an alert graph is a tree, a \((1 - \frac{1}{e})\)-approximation algorithm runs in \(O(k|V|)\)

- Intuition: sparsify alert graphs into trees, preserving most information

- Maximum directed spanning trees are trees in an alert graph
  - Span all nodes in an alert graph
  - Sum of edge weights is maximized
Single-Tree Approximation (cont.)

- Linear-time algorithm to search maximum directed spanning tree
  
  ![Graph](image)

- Drawback: accuracy loss in Gain estimation
  
  - Edge of the highest weight is always selected
  - Edges of similar weight never get selected
Multi-Tree Approximation

- Sample multiple trees from an alert graph

\[ G \]

\[ T_1 \]

\[ T_L \]

Tree sampling

Gain estimation

Average Gain

[Gain]

[Gain_{T_1}]

[Gain_{T_L}]

\[ n \]
Experimental Results

- Efficiency comparison on LogicMonitor alert graphs

- BnP is 30 times faster than the baseline
- Multi-tree approximation is 80 times faster with 0.1 quality loss
- Single-tree approximation is 5000 times faster with 0.2 quality loss
Conclusion

- Critical alert mining is an important topic for automated system management in complex systems
- A pipeline is proposed to enable critical alert mining
- Tree approximation practically works well for critical alert mining

Future work

- Critical alert mining with domain knowledge
- Alert pattern mining
  - if two groups of alerts follow the same dependency pattern, they might result from the same problem
- Alert pattern querying
  - if we have a solution to a problem, we apply the same solution when we meet the problem again
Questions?

Thank you!
Experiment Setup

- Real-life data from LogicMonitor
  - 50k performance metrics from 122 servers
  - Spans 53 days

- Offline dependency rule mining
  - Training data: the latest 7 consecutive days
  - Mined 46 set of rules (starting from the 8th day)
  - Learning algorithm: Granger causality

- Alert graphs
  - Constructed 46 alert graphs
  - #nodes: 20k ~ 25k
  - #edges: 162k ~ 270k
Case study

Critical Alert 1
- StorageUsed (Server 1)
  - Apache-BusyWorkers (Server 1)
    - us-west-Ping-avgrtt (Server 2)
  - Apache NoResponse (Server 2)

Critical Alert 2
- StorageUsed (Server 3)
  - shared memoryUsed (Server 3)
  - SDA writetime (Server 3)
  - MongoDB total backup jobs (Server 3)

- DiskReadLatency (Server 4)
  - CPU Cores Busy (Server 5)
  - MongoDB status (Server 6)
  - Mcollective Reg status (Server 5)