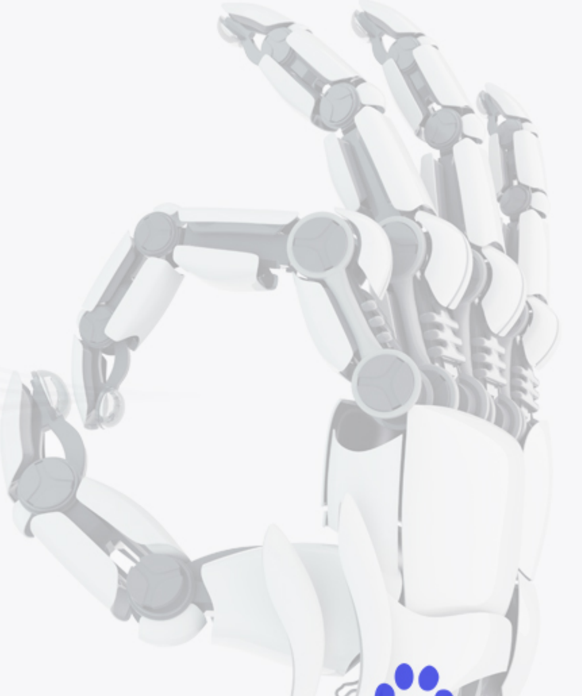
Automatic Metric Screening for Service Diagnosis

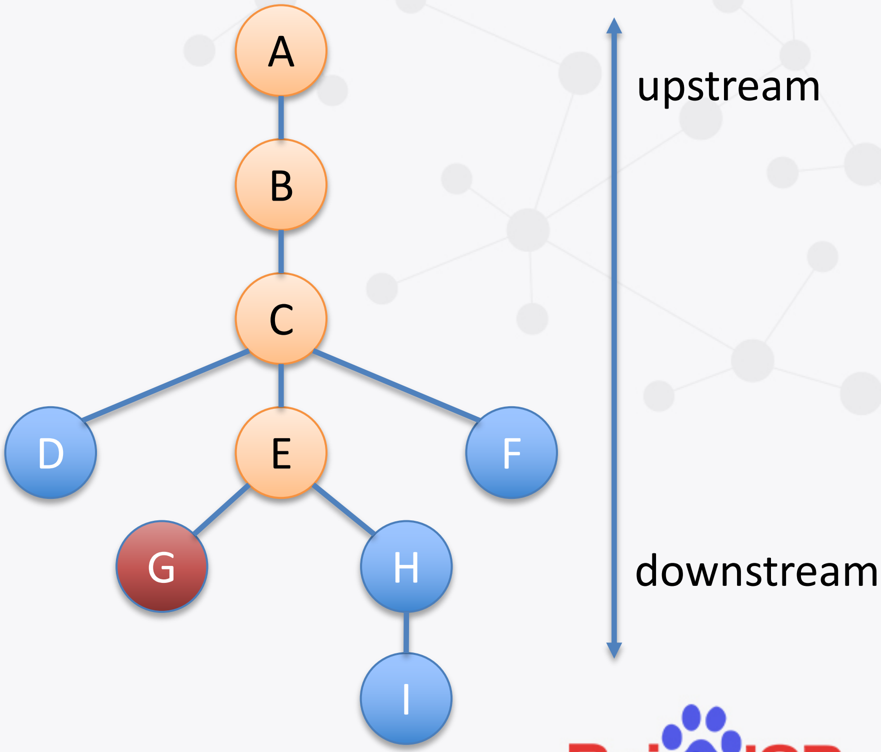
Service Diagnosis

- Quick diagnosis upon service faults
- Narrow down the checking scope
  - Perform stop-loss actions
    - Traffic switching, release rollback
  - Understand how the fault emerges
- Automatic metric screening
  - Reduce diagnosis time by 80%
  - Minimum manual configuration



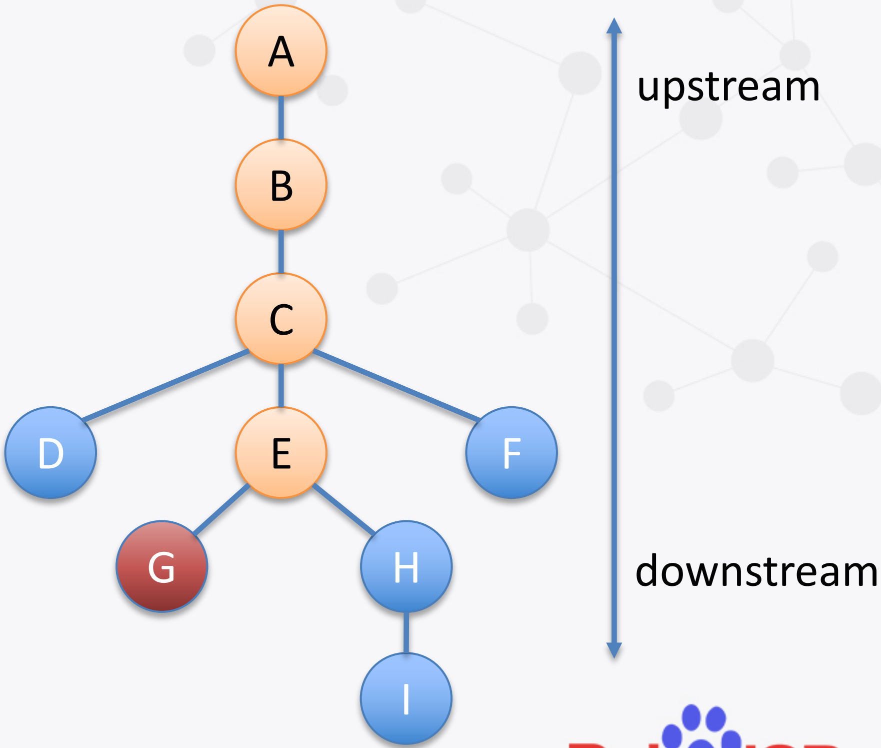
Diagnosing a Fault

- PV lost perceived at front end
- Get the call graph
  - Modules as nodes
  - Calls as edges
- Examine the modules
  - upstream to downstream
  - Look at certain metrics
  - Decide the root cause module



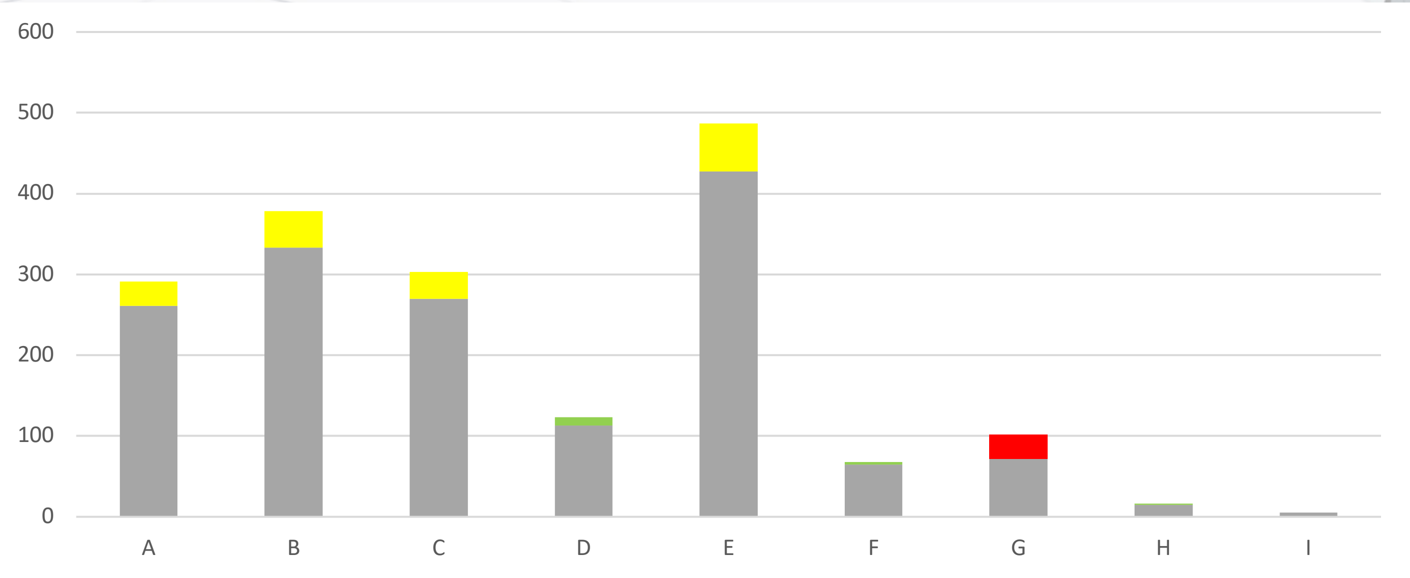
Diagnosis Tool

- Show the call graph
- Mark the abnormal modules
  - How to mark?



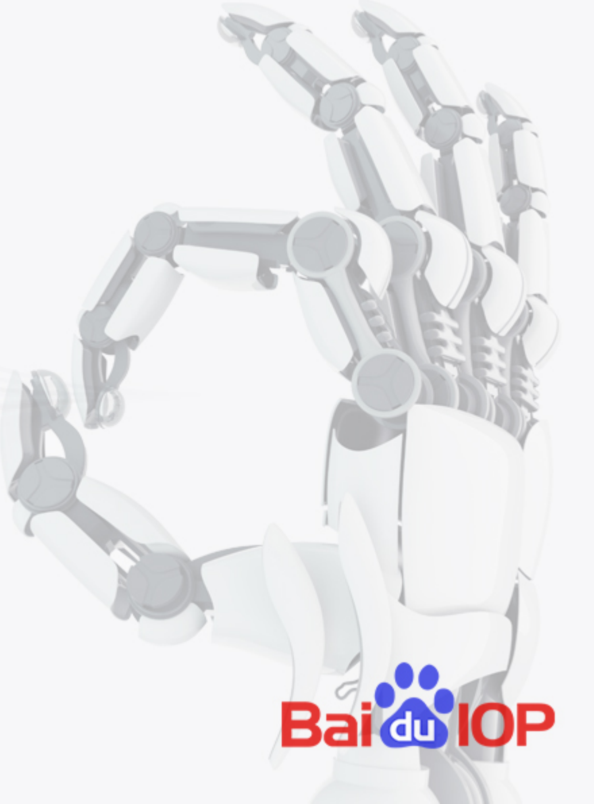
Module Abnormal Detection

- Golden metric, e.g. response time



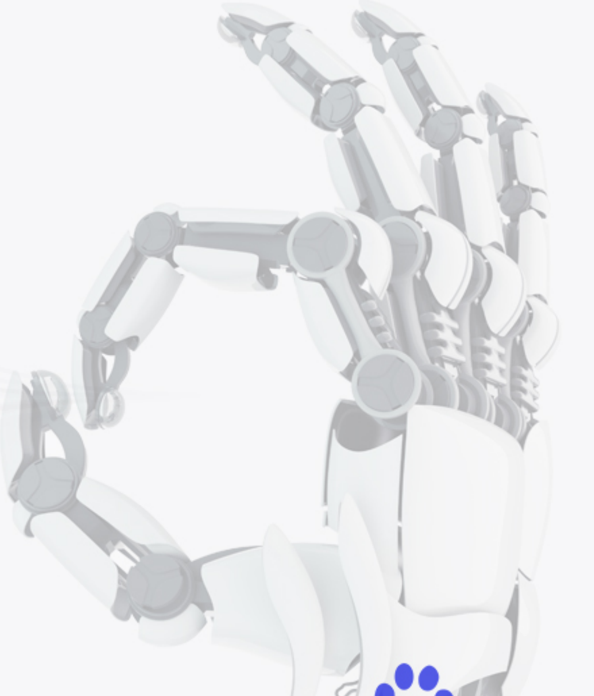
Semi-automatic Approach

- Synchronize call graphs
  - Continuous call graph evolution
  - Standard RPC middleware
- Golden metric configuration
  - New fault type requires new metrics
- Multiple call graphs
  - One call graph per datacenter



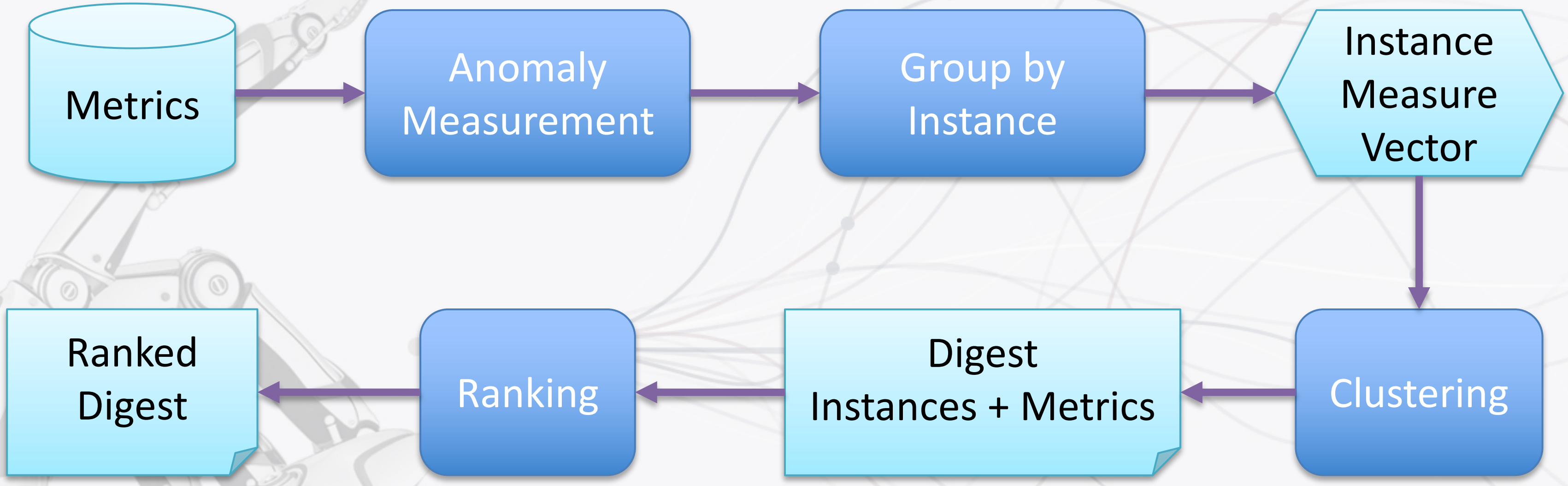
Automatic Metric Screening

- No dependency on golden metrics
  - Check all modules, all metrics
- Find out abnormal metrics
  - Flexible to metric diversity
- Recommendation
  - Quantitative anomaly measurement
  - Legible



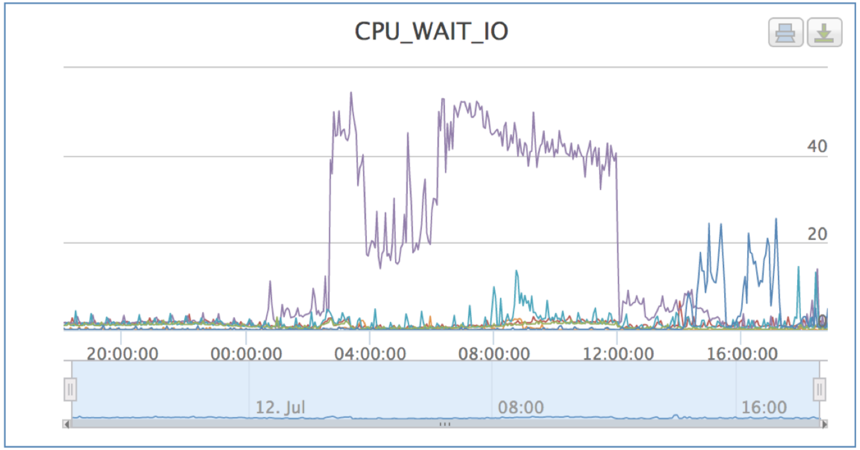
Baidu IOP

Screening Instance Level Metrics



Anomaly Measurement (I)

- Sudden changes
  - Many faults reflect on sudden changes on certain metrics
  - Easy to understand
  - Easy to detect
    - Compare data after faults against those before faults



Anomaly Measurement (II)

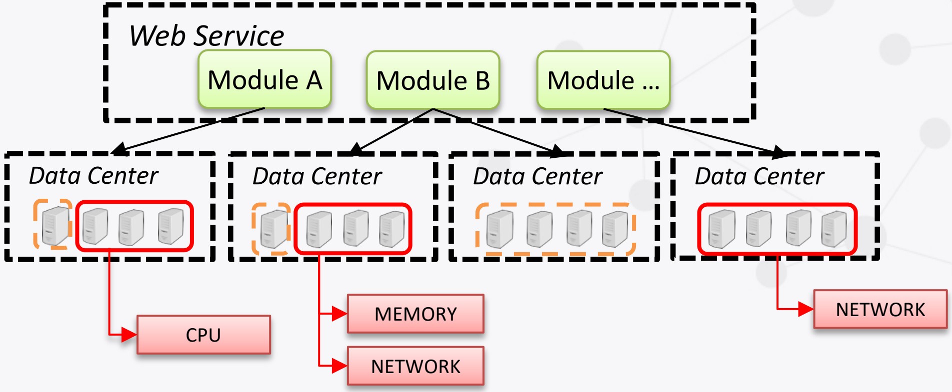
- Data
  - 60 mins before the fault $\{x_i\}$
  - 5 mins after the fault $\{y_j\}$
- Measure
  - Conditional probability $P(\{y_j\}|\{x_i\})$
  - Single point probability
    - Overflow $P(y \geq y_j|\{x_i\})$
    - Underflow $P(y \leq y_j|\{x_i\})$

Anomaly Measurement (III)

- Kernel density estimation
  - $f(y) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(y; x_i, \sigma)$
  - $P\left(y \geq y_j \middle| \{x_i\}\right) = \int_{y_j}^{+\infty} f(y)$
  - $P\left(y \leq y_j \middle| \{x_i\}\right) = \int_{-\infty}^{y_j} f(y)$
  - Kernels: Gaussian, Poisson, Beta
- Use log probability to combine
  - $\log P_{\text{overflow}}\left(\{y_j\} \middle| \{x_i\}\right) = \frac{1}{m}\sum_{j=1}^{m} \log P_{\text{overflow}}\left(y \geq y_j \middle| \{x_i\}\right)$

Clustering

- Anomaly measure vector
  - $D_m = -\log P(\{y_j\}|\{x_i\})$ for metric m
  - $\langle D_1, D_2, \dots \rangle$ for each instance
- Clustering
  - Instances within a module×DC
  - DBSCAN
  - Digest = instance set + abnormal metric set



Bai du IOP

Ranking the Digests

- Proportion of the instances
- Number of abnormal metrics
- Anomaly degree
  - $D_m = -\log P(\{y_j\}|\{x_i\})$
- Train a ranker

Module	Portion	Instances	Metrics
G.DC1	0.3	1.G.DC1 2.G.DC1 3.G.DC1 …	CPU_SERVER_LOADAVG_1 22.801403 NET_TCP_ACTIVE_OPENS 16.90959 CPU_IDLE 15.137489 CPU_HT_IDLE 14.600515 NET_TCP_TIME_WAIT 11.838425 CPU_INTERRUPT 11.682794 NET_TCP_OUT_SEGS 11.613833 CPU_SERVER_LOADAVG_5 11.550683 NET_TCP_IN_SEGS 11.136453 …
E.DC1	0.1	7.E.DC1 9.E.DC1 …	NET_TCP_CURR_ESTAB 15.696724 NET_TOTAL_SOCKETS_USED 14.688062 DISK_TOTAL_READ_REQ 11.241439 …

Bai du IOP

Offline Evaluation Result

- Use Linux standard system performance indicators
- 70 historic cases
  - Root cause can reflect on Linux indicators
- 60 ranked root cause digest as top 1

- Human diagnosis time: 6min~152min, mean 35min
- Algorithm execution time: <=6min

THANK YOU !

BAIDU@IOP出品