# DistIA: A Cost-Effective Dynamic Impact Analysis for Distributed Programs

Haipeng Cai* and Douglas Thain+

*Electrical Engineering and Computer Science, Washington State University

+Computer Science and Engineering, University of Notre Dame

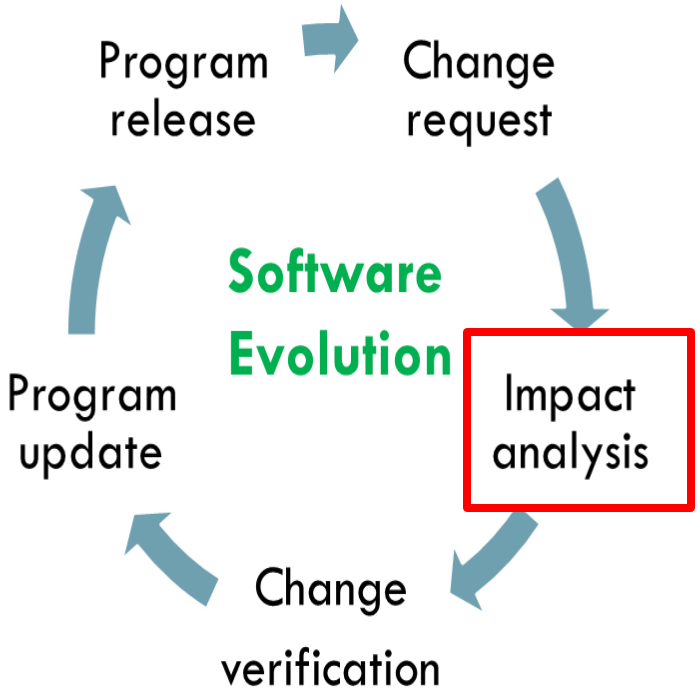


ASE 2016

Supported by ONR Award and WSU

# Problem

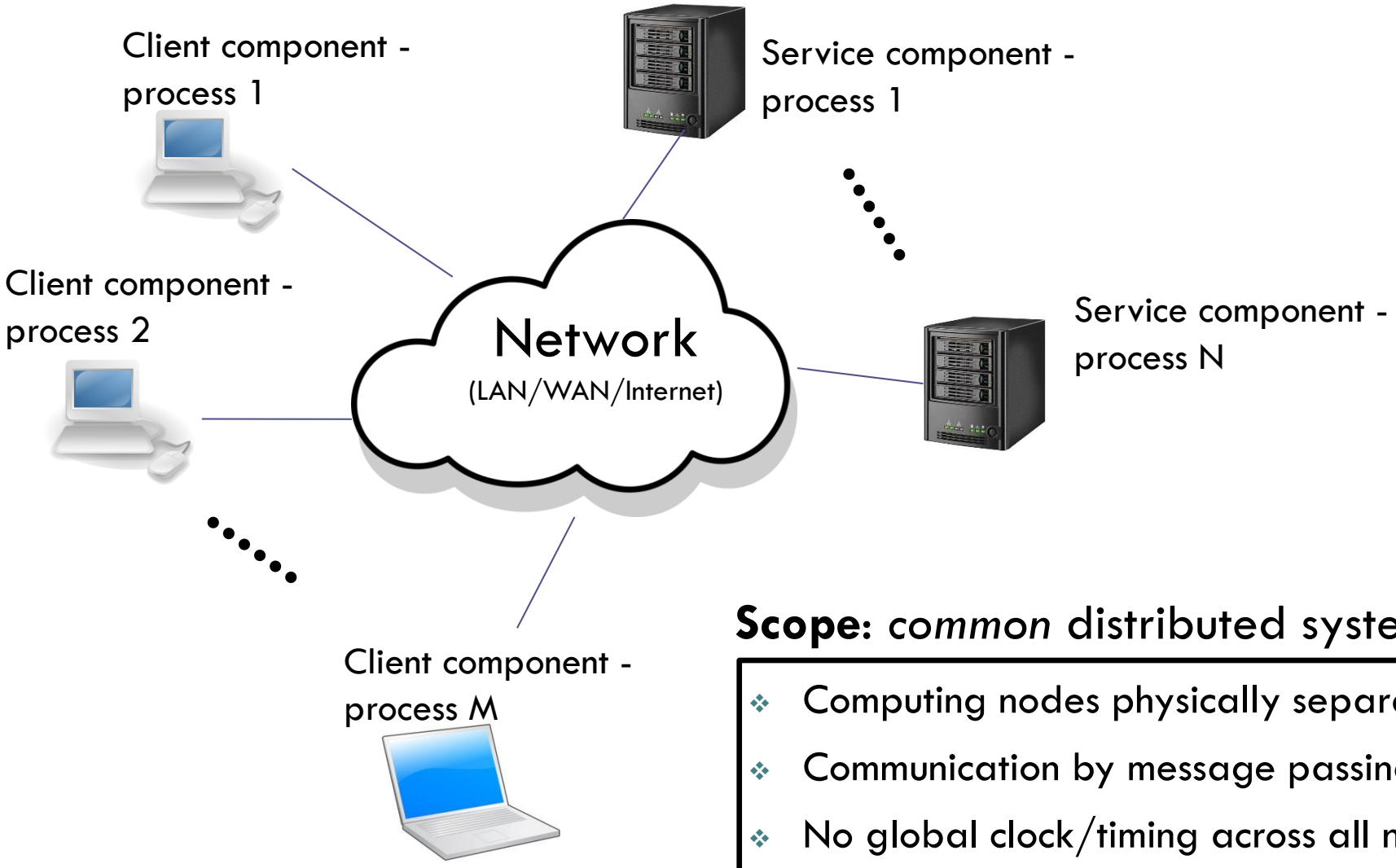





- Various approaches
  - Mode: *predictive* / *descriptive*
  - Technique: *static* / *dynamic* / *hybrid* / *repository mining* / *IR* / *coupling*
  - Granularity: *statement* / *method* /*class* / *file level*
- Different domains
  - Centralized programs (single/multi-threaded)
  - Distributed programs (multi-process)
- Predictive, dynamic, method-level
  - *Proactive*
  - *More representative of actual behaviors*
  - *Balanced scalability and precision*
  - *Not available yet to distributed programs*

# Run-time setting of distributed programs



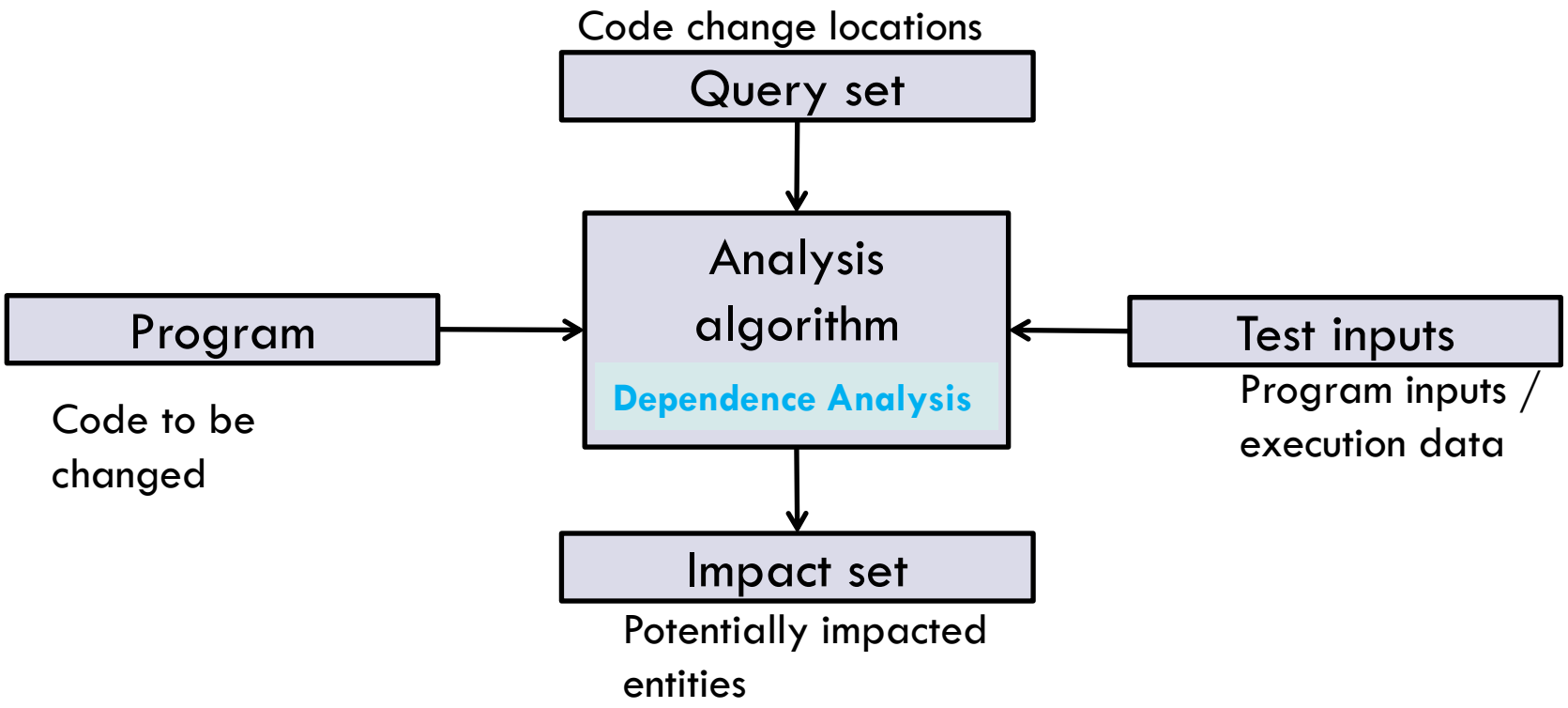


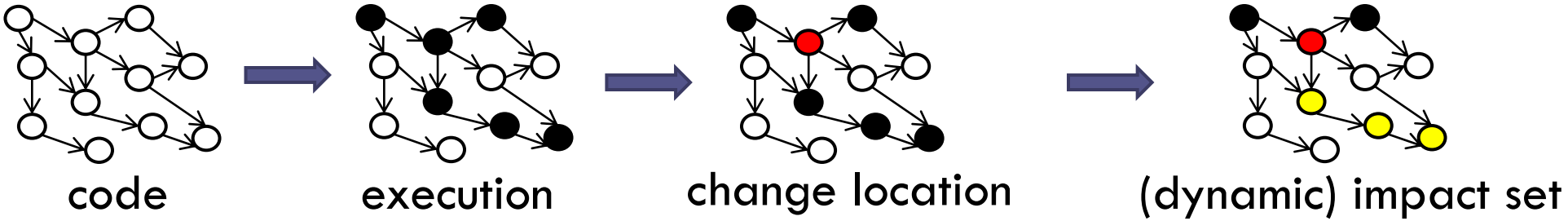

**Scope:** *common* distributed systems

- Computing nodes physically separated
- Communication by message passing via socket
- No global clock/timing across all nodes

# Dynamic impact analysis

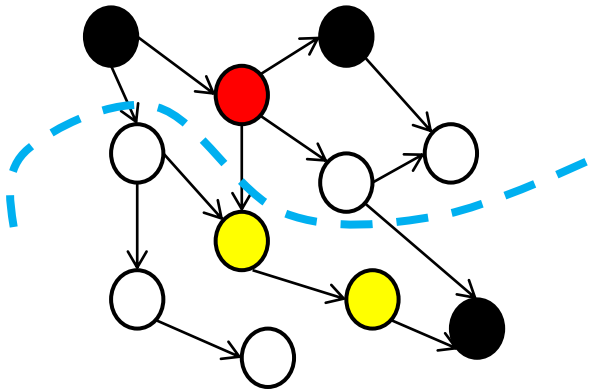




# Dependence in distributed programs





Server

```
1  public class S {
2      Socket ssock = null;
3      public S(int port) { ssock = new Socket(port); ssock.accept(); }
4      char getMax(String s) {...}
5      void serve() { String s = ssock.readLine();
6          char r = getMax(s); ssock.writeChar(r); }
7      public static int main(String[] a) {
8          S s = new S(33); s.serve(); return 0; }}
```

Networking (Socket)

Client

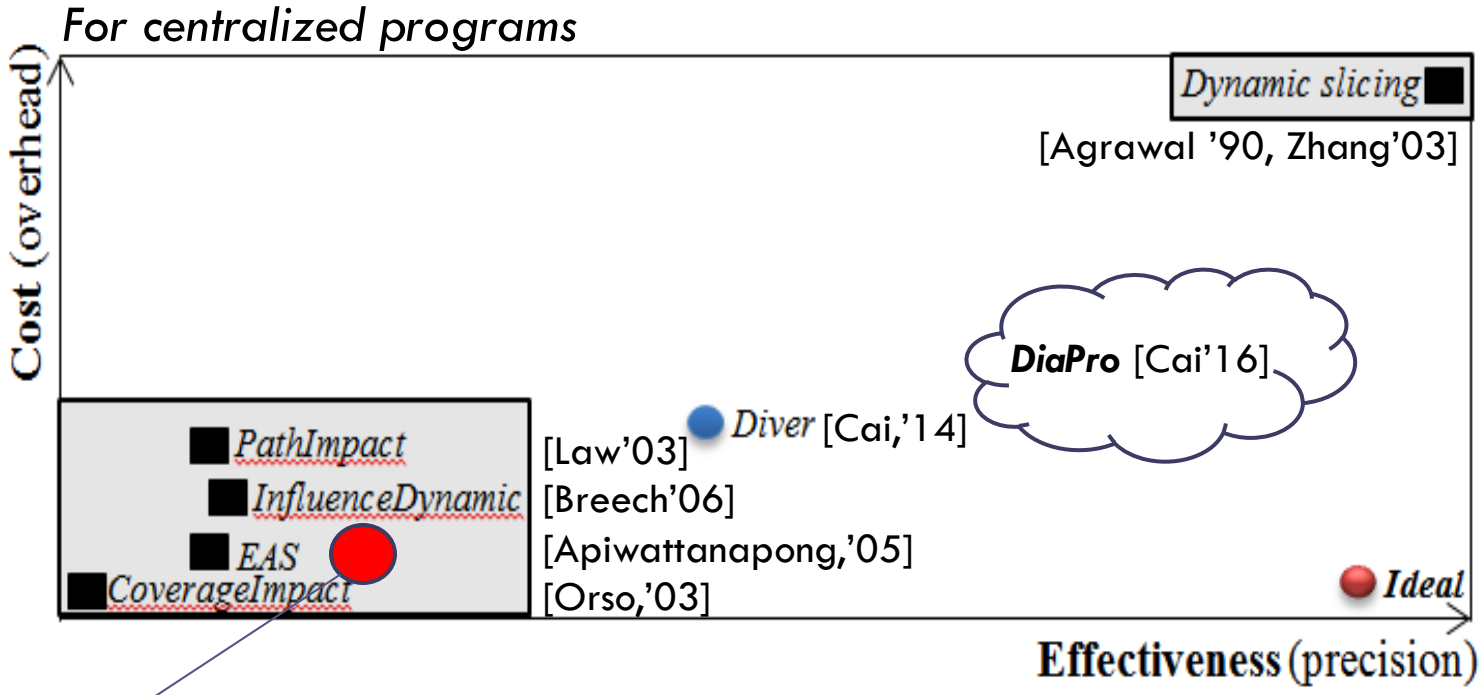
```
1  public class C {
2      Socket csock = null;
3      public C(String host,int port) { csock = new Socket(host,port); }
4      void shuffle(String s) {...}
5      char compute(String s) { shuffle(s); csock.writeChars(s);
6          return csock.readChar(); }
7      public static int main(String[] a) { C c = new C('localhost',33);
8          System.out.println( c.compute(a[0]) ); return 0; }}
```

Analysis algorithm

**Dependence Analysis**

# DistIA: a cost-effective solution



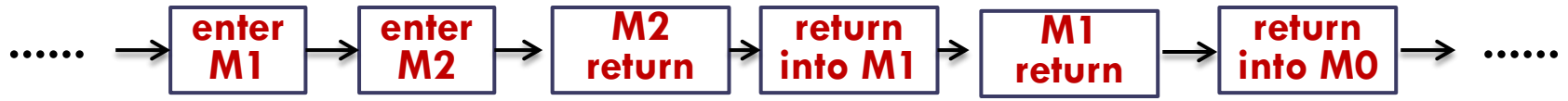



- DistIA
  - Goal
    - cost-effective (rough-yet-rapid [Jackson'00] )
  - Strategy
    - *lightweight* dynamic dependence *approximation*

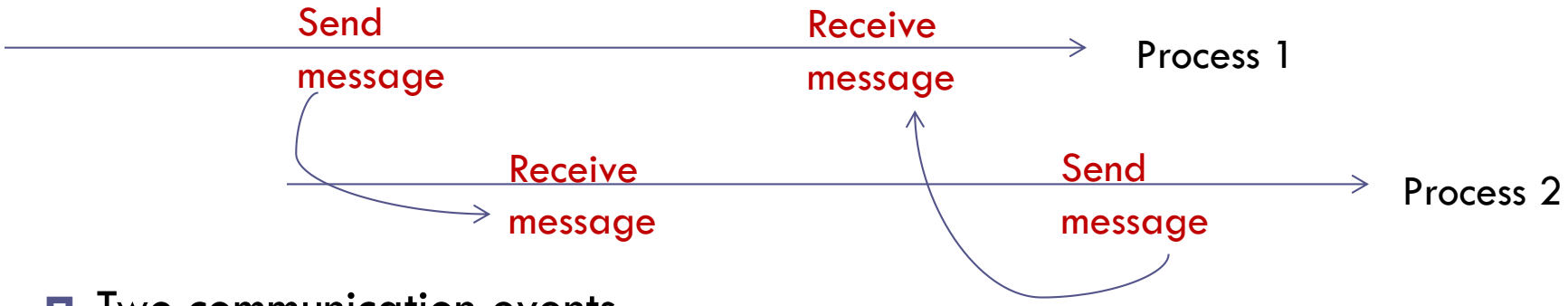
# Dependence approximation



- Control flow approximation
  - Method execution order (partial ordering)




  - Three method events
    - Entry, return, and returned-into
    - Suffice for single-process partial ordering [Apiwattanapong,'05]

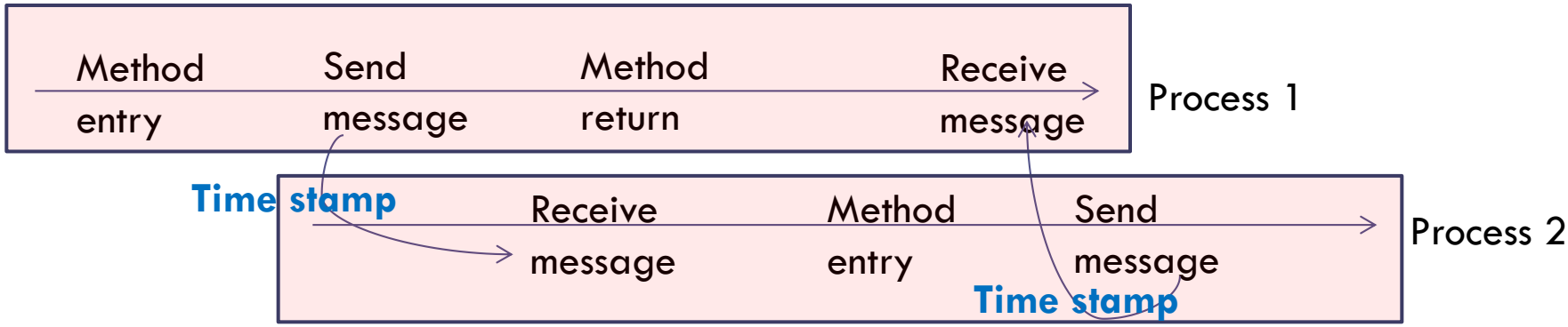



  - Two communication events
    - Message sending, and message receiving
    - Necessary for synchronizing the timing of events across processes

# Dependence approximation



- Control flow approximation
  - Global partial ordering [Lamport, '78]




  - Impact inference
    - Happens-before -> impact relation

$$m1_e \prec m2_x \bigvee m1_e \prec m2_i \Longrightarrow m1 \; impacts \; m2$$

e: entry, x: return, i: returned-into

$$IS(c) = \{m \mid c_e \prec m_i \vee c_e \prec m_x\}$$

IS: impact set, c: query

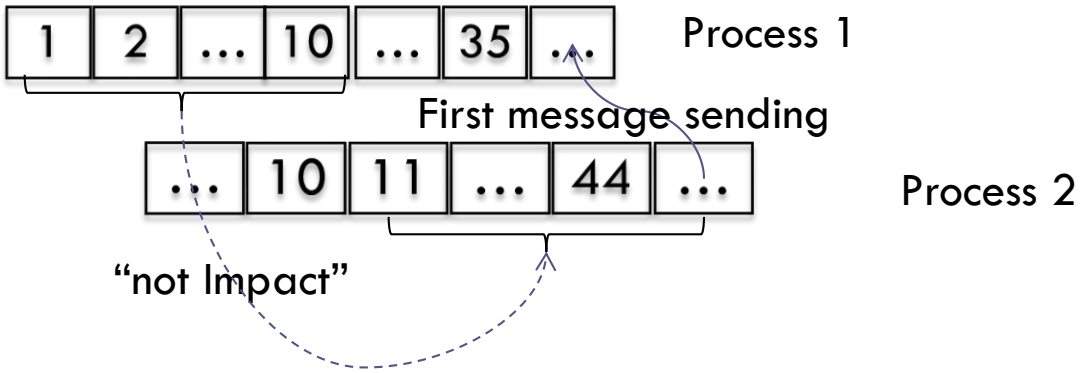
$$E \prec E' \iff T_L(E') \geq T_F(E')$$

$T_F$: timestamp ot tirst occurrence, $T_L$: timestamp ot last occurrence

# Dependence approximation



- Data flow approximation
  - Message-passing semantics




- Control + data flow approximation

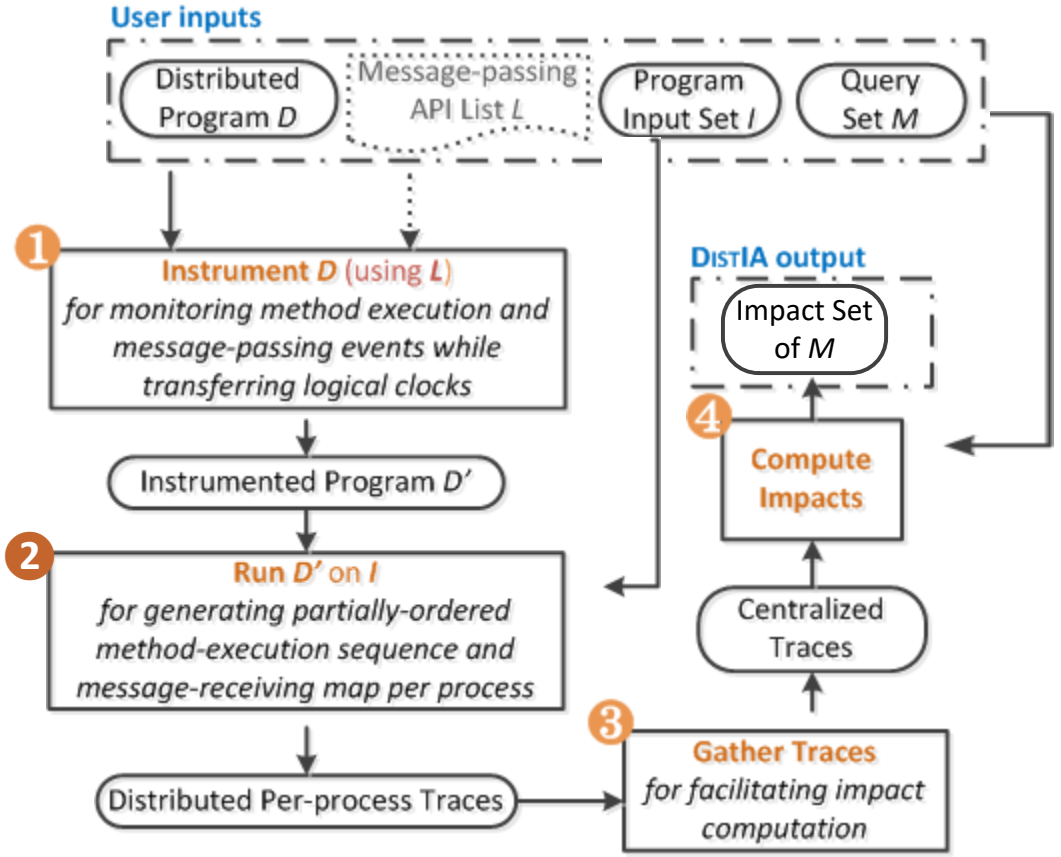
$$P_j^{m'} \prec P_i^{m} := \begin{cases} T_L(P_i^{m}) \geq T_F(P_j^{m'}), & \text{if } i = j \\ T_S(P_j) \neq \text{null} \wedge T_L(P_i^{m}) \geq \\ \quad \max(T_F(P_j^{m'}), T_S(P_j)), & \text{if } i \neq j \end{cases}$$

$$IS(c) = \{m \mid c_e \prec m_i \vee c_e \prec m_x\}$$

# DistIA



- Workflow




- Algorithms
  - Communication event monitoring
  - Impact computation

# Application to real-world distributed programs



- Subject programs

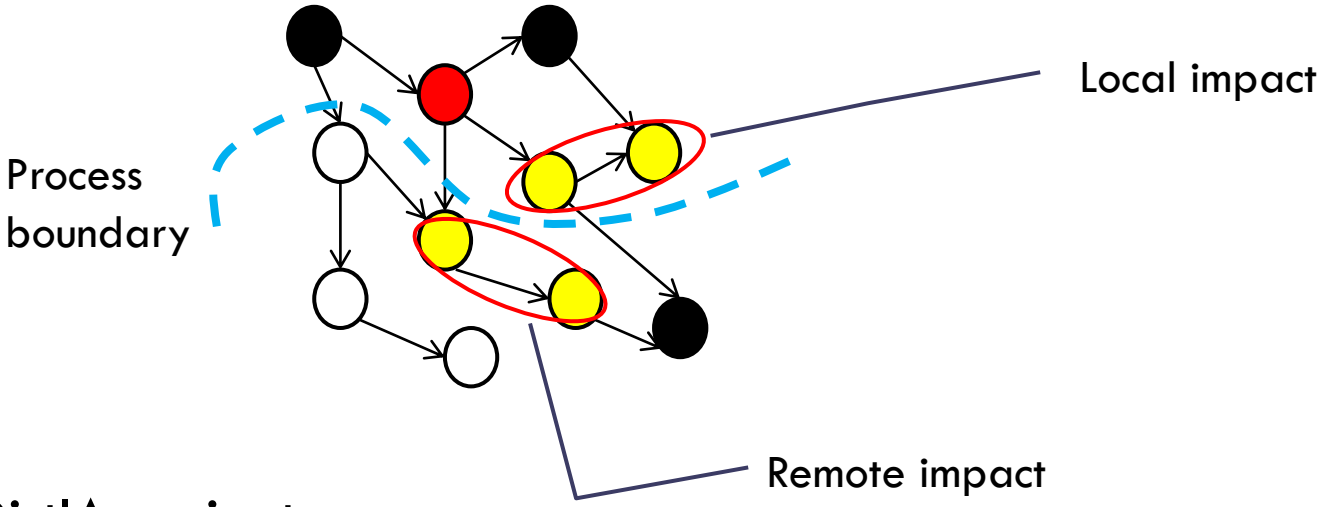
| Subject | Description | #SLOC | Test inputs |
|---|---|---|---|
| MultiChat (r5) | C/S chat app, Socket I/O Stream | 470 | Integration |
| NIOEcho (r69) | C/S echo service, Java NIO | 412 | Integration |
| OpenChord (v1.0.5) | P2P lookup service, hybrid IO | 38,084 | integration |
| ZooKeeper (v3.4.6) | Coordination service, hybrid IO | 62,450 | Integration, system, load |
| Voldemort (v1.9.6) | Key-value store, hybrid IO | 163,601 | Integration, system, load |
| Freenet (v0.7.0) | Anonymous data-sharing, hybrid IO | 196,281 | integration |

- Implementation
  - Non-intrusive instrumentation dealing with a *variety of distributed system architectures*

# Research question: effectiveness



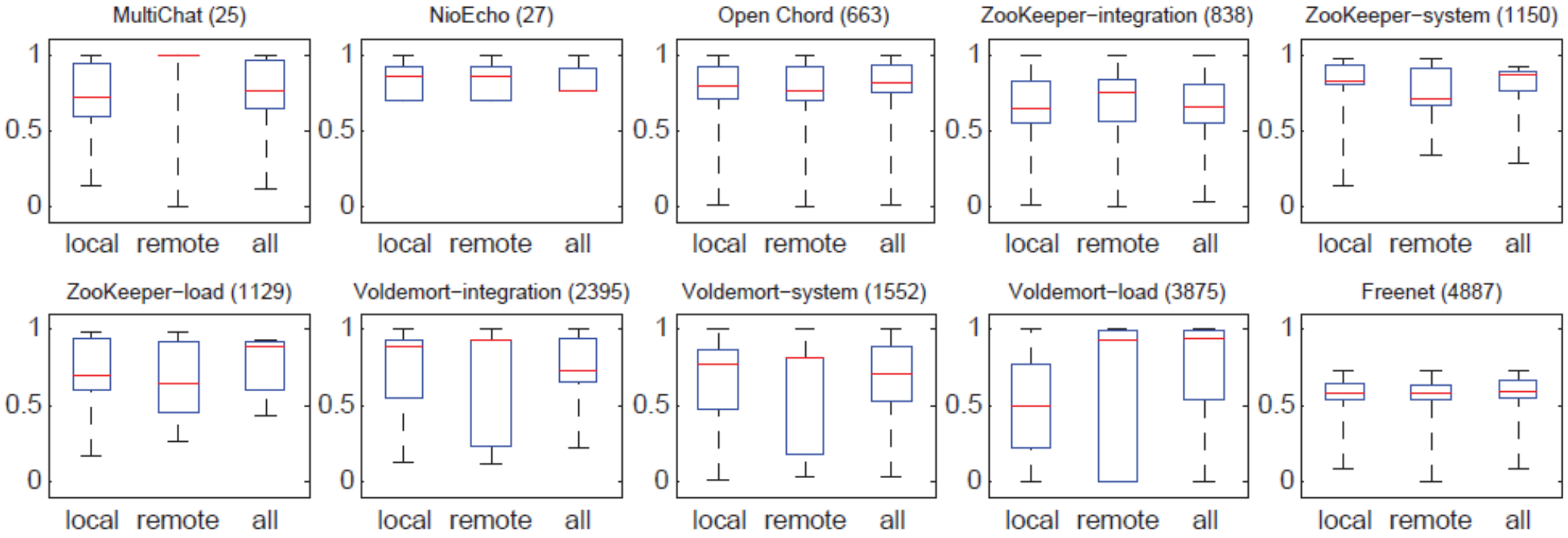
- How effective is DistIA compared to "existing options"?
  - *Coverage-based solution (MCov)* as baseline
- Metrics
  - Impact set size ratio: DistIA over Mcov
    - Assuming both are "dynamically" sound/soundy [Livshits et al., '15]
  - Whole impact set (all) and two subsets: local impact set, remote impact set




- Two DistIA variants
  - Basic: control-flow only
  - Enhanced: data + control flow

# Result: effectiveness



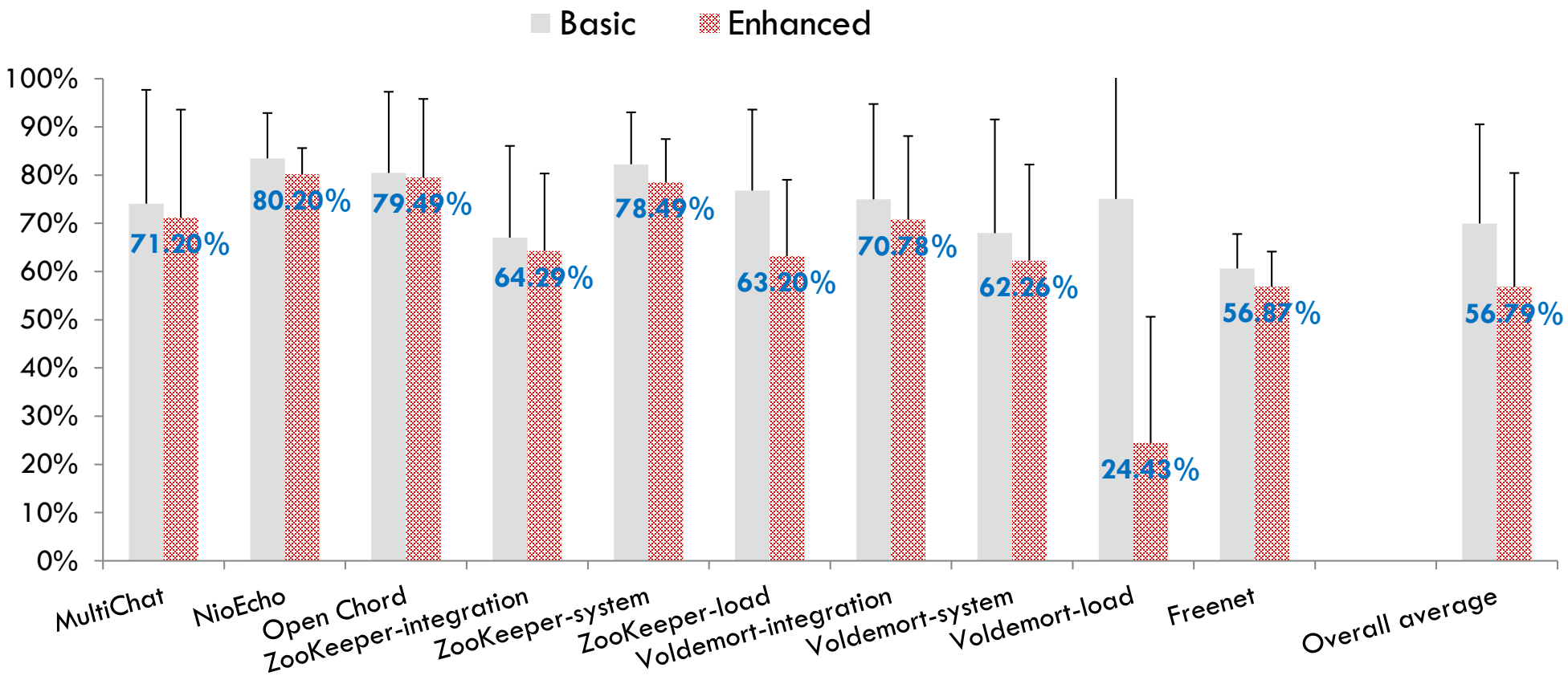



Distribution of impact set size ratios of DistIA-basic/Mcov, the lower the better

**Mean impact-set reduction: 31%**

# Result: effectiveness






Mean impact set size ratios of DistIA-basic versus DistIA-enhanced, the lower the better

**Overall mean impact-set reduction: 43%**

# Research question: costs



- How efficient and scalable would DistIA be?
  - Practicality of using it in terms of overheads

- Metrics
  - Time cost
  - Storage cost
  - Run-time slowdown

# Results: costs



| Phase | Range | Mean |
|---|---|---|
| Instrumentation | 12~165 seconds | 62 seconds |
| Run-time slowdown | 1~21% | 8% |
| Impact-set querying | 4~114 milliseconds | 66 milliseconds |

Time costs of DistIA enhanced; the basic version costs even less

*Storage cost < 1MB

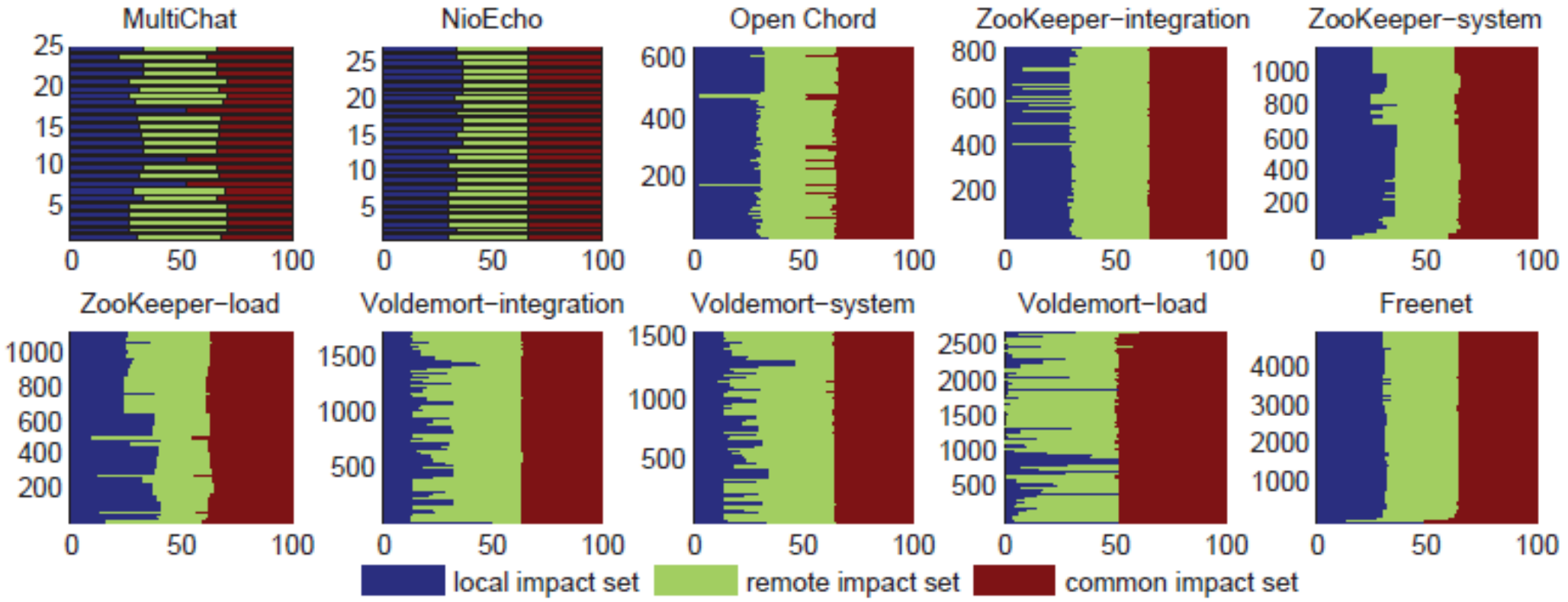
# Research question: impact distribution



- How are the impacts distributed across process boundaries?
  - Component-level structure of distributed programs
- Metrics
  - Impact-set breakdown: local, remote, common impact sets

# Result: impact distribution






Breakdown of each impact set (given by DistIA-basic) into three disjoint subsets

# Summing up



- Contributions
  - The first dynamic impact analysis for common distributed programs with socket-based message passing
  - Open-source implementation of the analysis working on real-world, large distributed systems of various architectures
  - Empirical evidences showing its promising effectiveness and scalability
- Future work
  - Explore other cost-effectiveness options (with better precision)
  - Exploit its use in distributed system testing and understanding

# Acknowledgements




- ONR grant to Notre Dame and WSU faculty startup fund for financial support
- Anonymous reviewers for very valuable comments
- Your attendance and attention


# Q&A



## DistIA: A Cost-Effective Dynamic Impact Analysis for Distributed Programs

*Drawing on partial ordering of lightweight dynamic information (method execution events) and simple message-passing semantics heuristics to offer a cost-effective impact-analysis option for real-world distributed programs.*

Haipeng Cai

http://eecs.wsu.edu/~hcai/

hcai@eecs.wsu.edu

# Case study I



- How precise are the DistIA impact sets relative to actual dynamic impacts?

| Subject & input | DistEA IS (precision) | | Manual true IS | | *MCov* IS | |
|---|---|---|---|---|---|---|
| | local | remote | local | remote | local | remote |
| MultiChat | 1 (100%) | 13 (69.2%) | 1 | 9 | 3 | 21 |
| MultiChat | 13 (76.9%) | 2 (50%) | 10 | 1 | 22 | 3 |
| Voldemort-system | 4 (100%) | 23 (56.5%) | 4 | 13 | 740 | 809 |
| Voldemort-system | 3 (33.3%) | 0 (-) | 1 | 0 | 811 | 440 |
| Voldemort-load | 13 (46.1%) | 41 (41.4%) | 6 | 17 | 288 | 500 |
| **Overall average** | 6.8 (**71.2%**) | 15.8 (**51.7%**) | 4.7 | 8 | 373 | 354.6 |

- Overall mean precision
  - DistEA-basic: ~60%
  - DistEA-enhanced: ~70%

# Case study 2



- How may DistIA results help with understanding inter-process interaction in distributed programs?
  - NioEcho
    - clearly showing the request initiation from client and server's response by echoing the message received, followed by client's steps in receiving the reply and processing it
  - ZooKeeper
    - Helped identify the coordination server relays the client request to a worker threat that interacts with database to carry out the client inquiries
  - In particular: the appearance of communication events in the trace and the timestamp (ordering) of method events are very helpful with sorting out the interactions