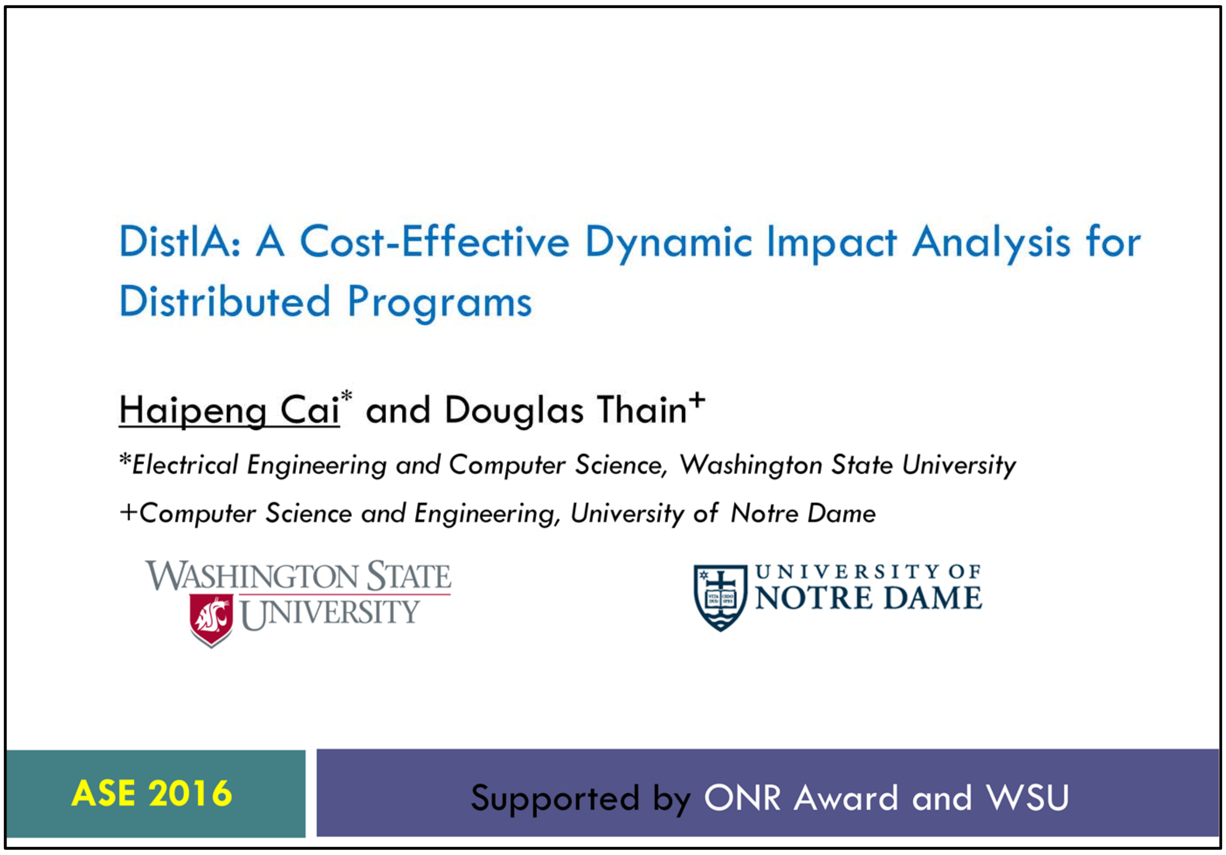


DistIA: A Cost-Effective Dynamic Impact Analysis for Distributed Programs

Haipeng Cai* and Douglas Thain+

*Electrical Engineering and Computer Science, Washington State University

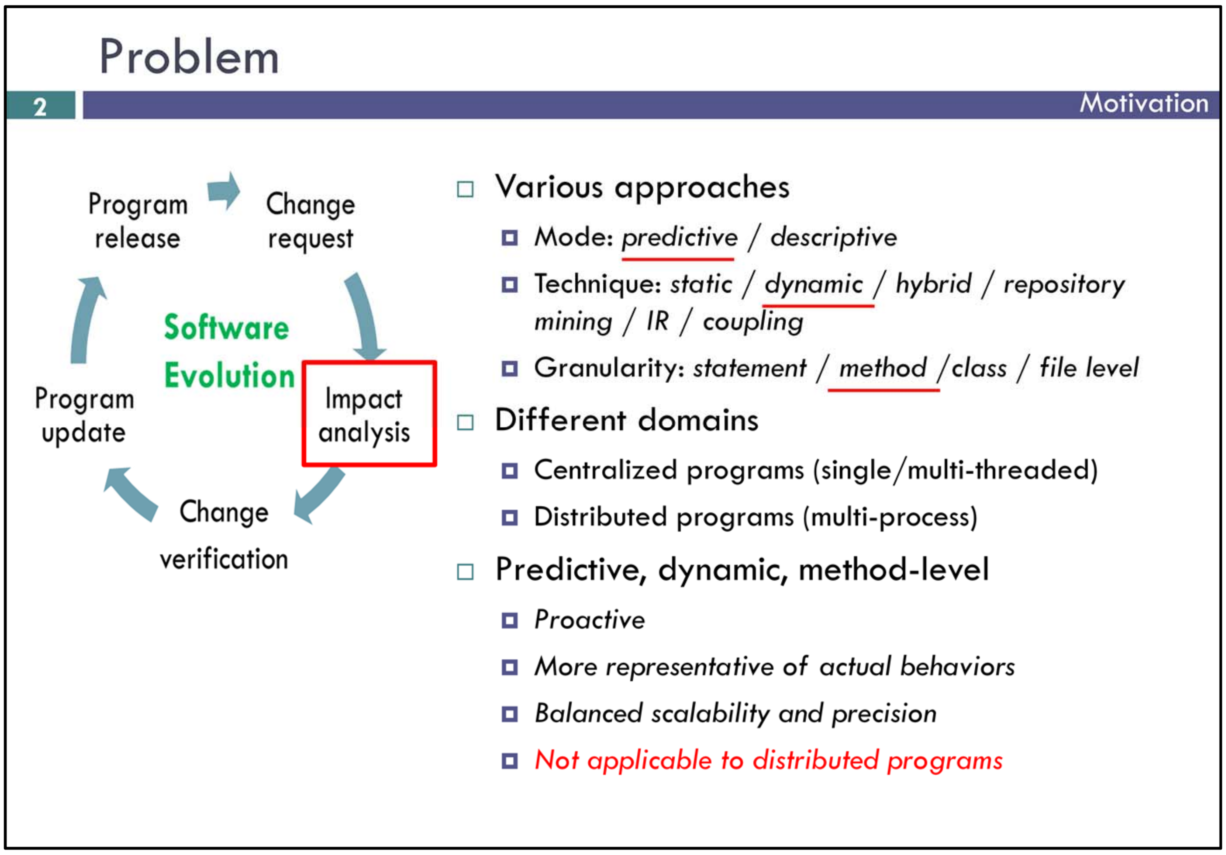
+Computer Science and Engineering, University of Notre Dame

ASE 2016

Supported by ONR Award and WSU

Thanks, {whoever introduces}.
And thanks all for being here to attend this talk.
I am haipeng cai from washington state university, this is part of my thesis work done at notre dame.

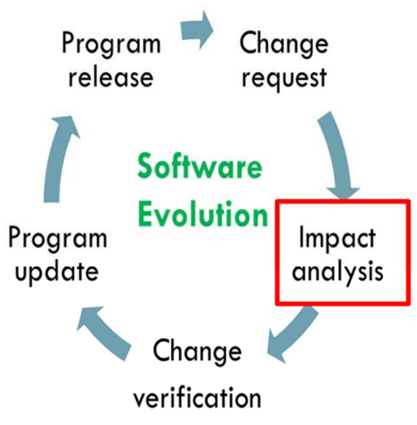
The topic of our paper is about one important activity during software evolution, impact analysis. More specifically, we are interested in the dynamic approach which address potential impacts of candidate changes for concrete program executions. And importantly, we target distributed programs that have not been well addressed in impact analysis.



Problem

2

Motivation



- 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 applicable to distributed programs*

Change impact analysis, or simply impact analysis, is an integral and critical step in software evolution.
Different approaches to impact analysis have been developed over the years.

In terms of the working mode, impact analysis can be predictive, applied before changes are made, or descriptive, applied with concrete changes available already.

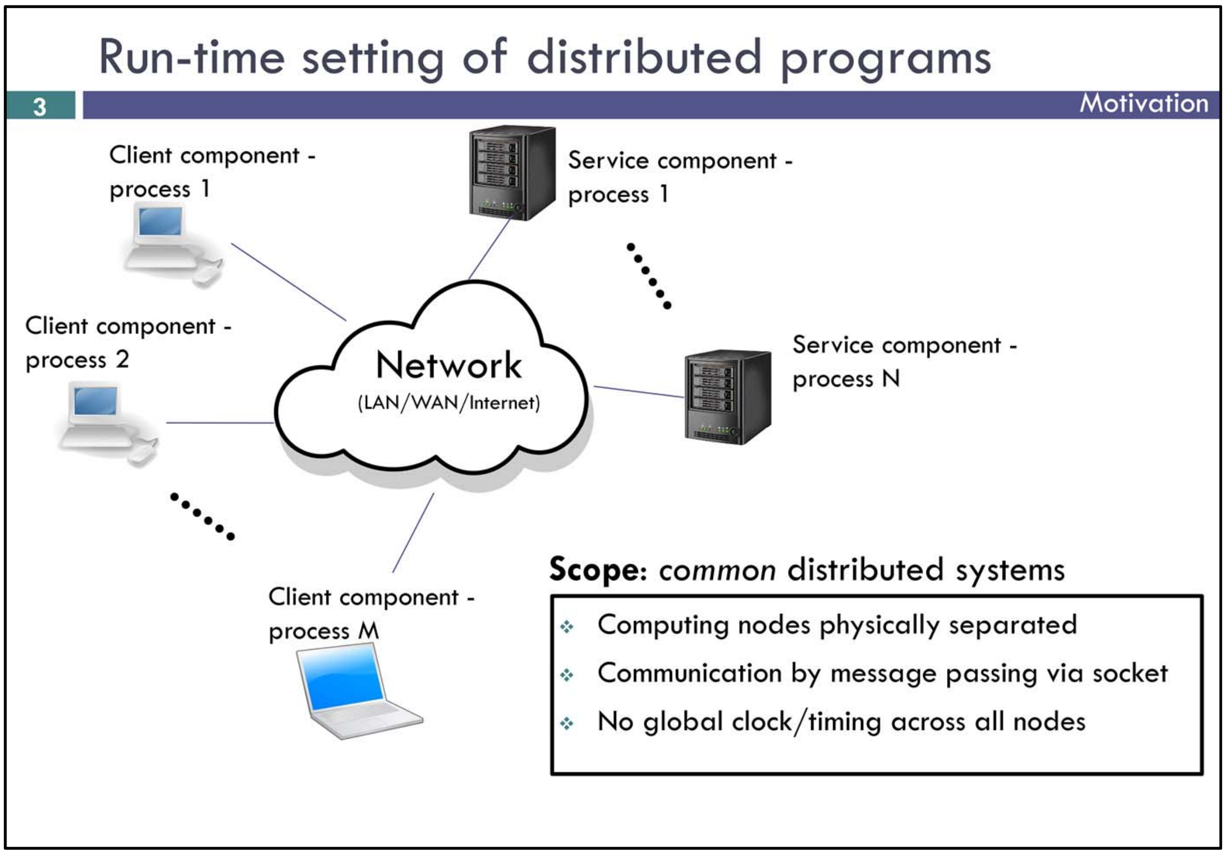
In terms of technique, static, dynamic and hybrid analyses have been proposed, researchers also have exploited techniques beyond code-based analysis, such as mining software repositories, computing coupling measures, and leveraging information retrieval methods.

And impact analysis has been addressed at different levels of granularity, ranging from fine-grained statement level to coarse level of file.

Impact analysis is not only needed for evolving centralized programs (single or multi-threaded), but also needed to evolve distributed programs.

We focus on a predictive dynamic analysis at method level, as the predictive analysis helps developers identify change effects earlier thus stay proactive against change risks; the dynamic approach produces results more representative of actual behaviors of the program; and working at method level enables a good balance between scalability and precision of the analysis.

However, existing such analysis techniques are not applicable to distributed programs.



Run-time setting of distributed programs

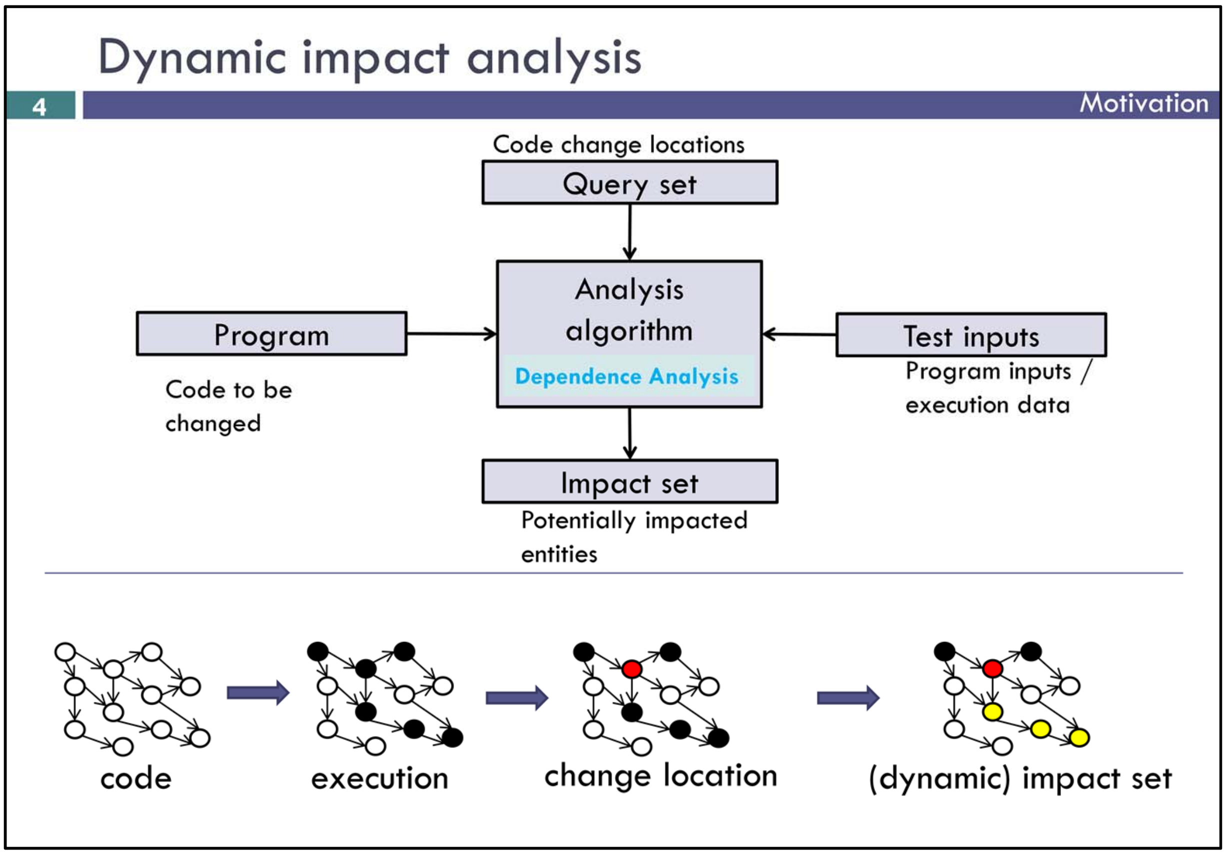
Client component - process 1
Service component - process 1
Client component - process 2
Network
(LAN/WAN/Internet)
Service component - process N
Client component - process M

Scope: *common* distributed systems

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

In a typical run-time setting, the components of a distributed program execute concurrently over multiple networked computers, each called a computing node. Some of these node run the service components while others act as clients, but each runs in a separate process. That is, the computing nodes are distributed across physically separated locations; commonly, they communicate through message passing based on socket. Importantly, there isn't a global clock or timing mechanism within the entire distributed system. These three characteristics define the scope of our work: common distributed systems. (versus event-based, RMI, etc.)

Define the distributed programs we are targeting at: socket-based message passing, without global clock, etc.



Dynamic impact analysis

4

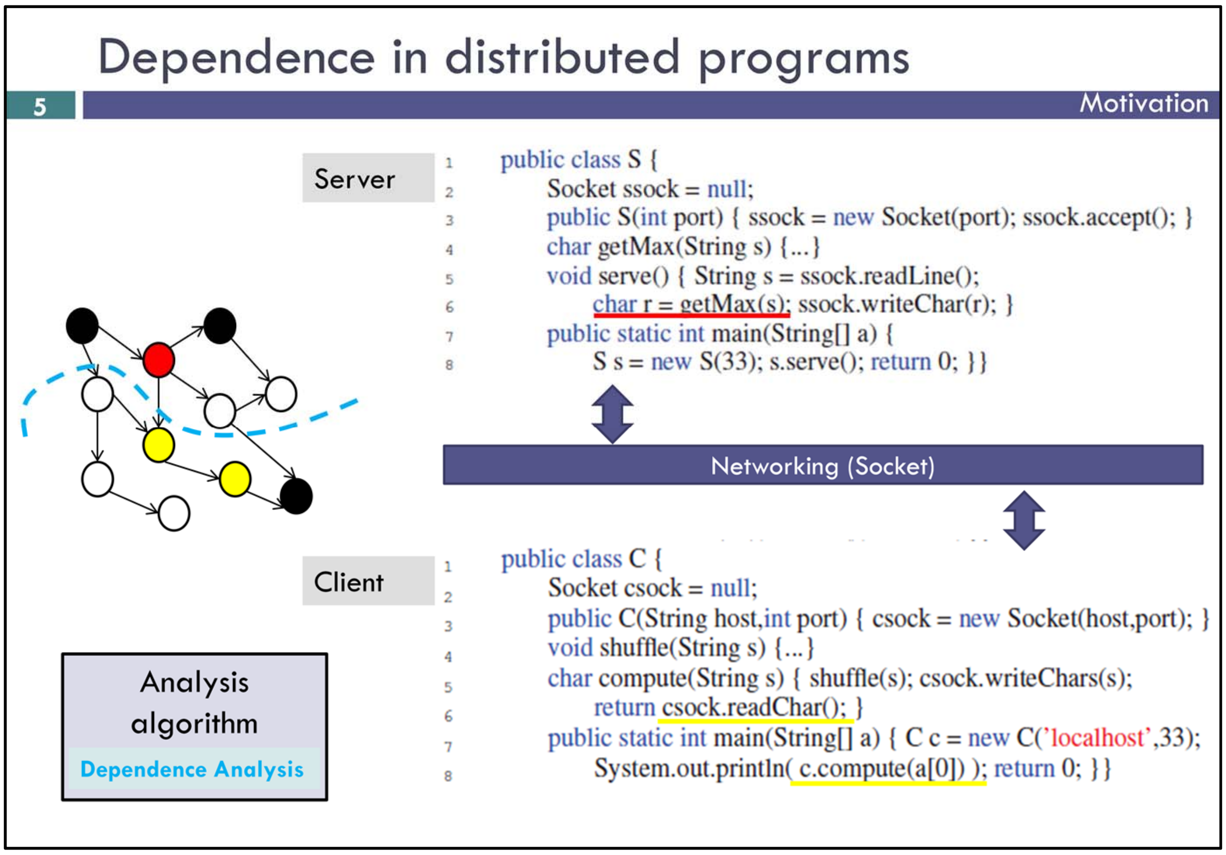
Motivation

Now, let us look at how a dynamic impact analysis in general.
At the core is analysis algorithm, and a major technique used in the algorithm is dependence analysis.

It takes a program to be changed, illustrated by a dependence graph here as dependence analysis works underneath; each node represents a program entity and each edge the dependence between two nodes.

It also takes a set of test inputs, from which execution data can be obtained; the black nodes are covered by the inputs.

Then, it takes a query set which is a set of potential change locations, illustrated by the red nodes here; finally dynamic impact set is computed, marked by the yellow nodes, as the eventual output of the 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; }}
```

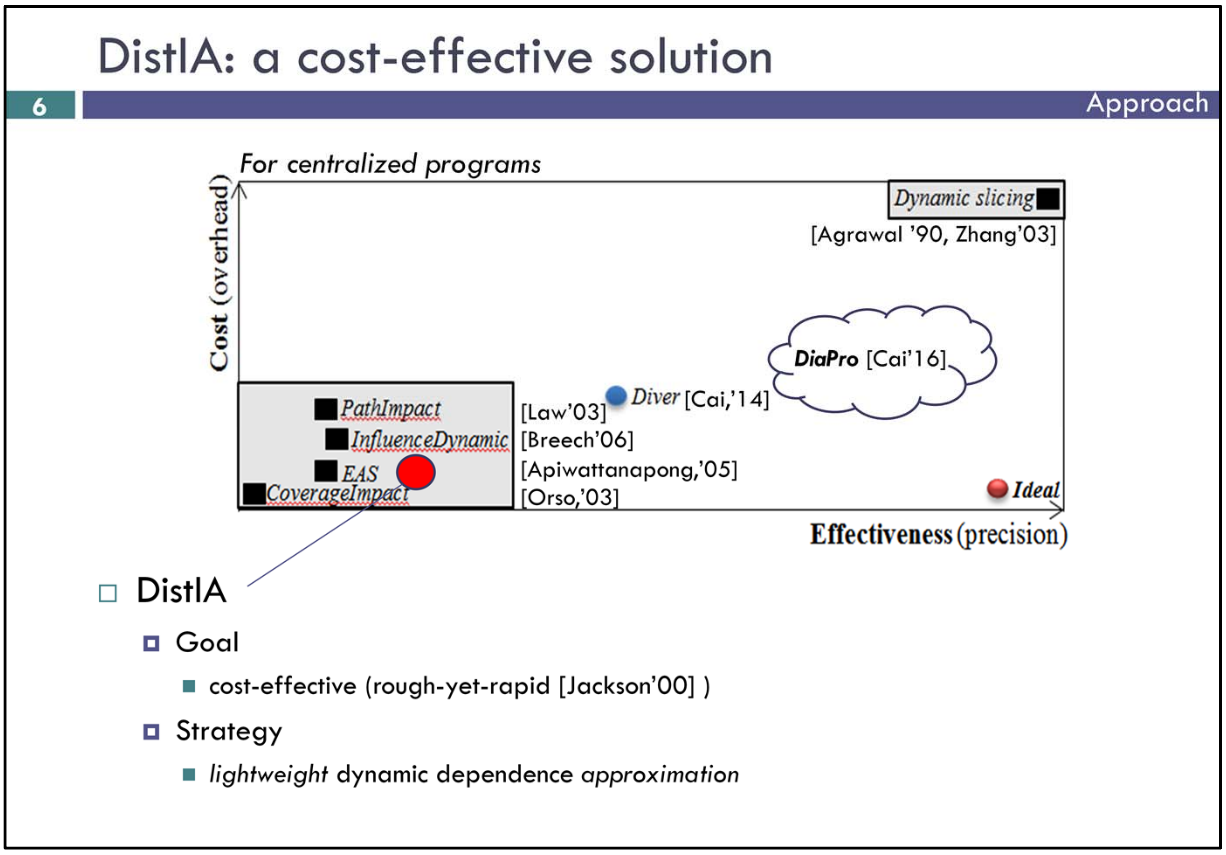
Analysis algorithm

Dependence Analysis

As we have seen, the core of the dependence-based impact analysis is to compute the impacts by navigating dependencies between change locations (red node) and potentially impacted entities (yellow nodes) among all executed ones.

However, in distributed programs computing the dependencies is challenging, as the queries and impacts can be loosely coupled or entirely decoupled, thus this is no explicit dependencies that existing approaches rely on.

For example, the program consists of a server component and a client that communicate through networking facilities, commonly via network socket. The server reads a line from a client and finds the maximal character to send back, while the client simply takes user inputs and relays such a task to the server. The change at line 6 in the server can affect lines 6 and 8 in the client, yet the dependencies between them are difficult to analyze because of their being implicit!



DistIA: a cost-effective solution

For centralized programs
Cost (overhead)
Effectiveness (precision)
Dynamic slicing
[Agrawal '90, Zhang'03]
DiaPro [Cai'16]
Diver [Cai,'14]
PathImpact [Law'03]
InfluenceDynamic [Breech'06]
EAS [Apiwattanapong,'05]
CoverageImpact [Orso,'03]
Ideal

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

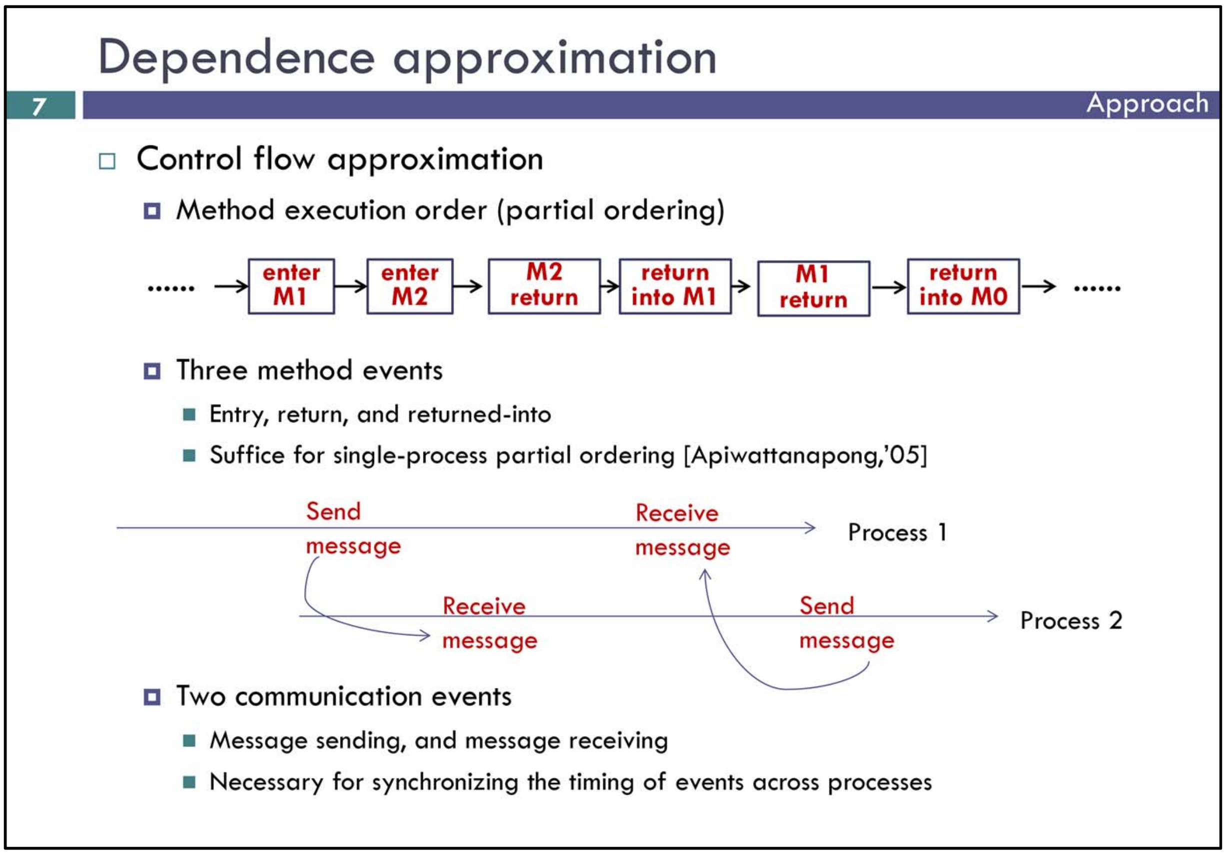
For this problem, our approach, called DistIA, short for distributed program impact analysis, aims at a cost-effective solution.

For centralized programs, dynamic impact analysis has been studied extensively. Previous approaches generally lie at two extremes in this two-dimensional cost-effectiveness design space, where the X axis represents the effectiveness (for instance, precision) and the Y axis represents the cost. The ideal case is right here, the closer to it the better.
We recently developed Diver and DiaPro to fill the gap between the two extremes.

Note that the techniques at the bottom-left here are not precise but highly efficient, thus still provides attractive cost-effective options, or called rough-rapid solutions.

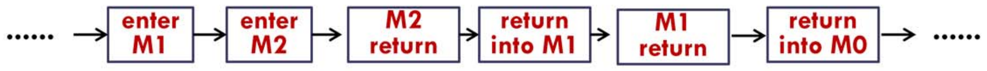
As a first step, we would like to take a position about here at this red spot, with DistIA, our goal is to provide such a cost-effective option for distributed programs.

To reach this goal, our strategy is to approximate dynamic dependencies in a very lightweight manner.

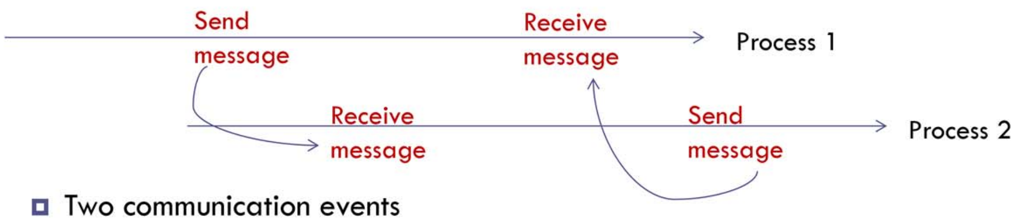


Dependence approximation

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



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

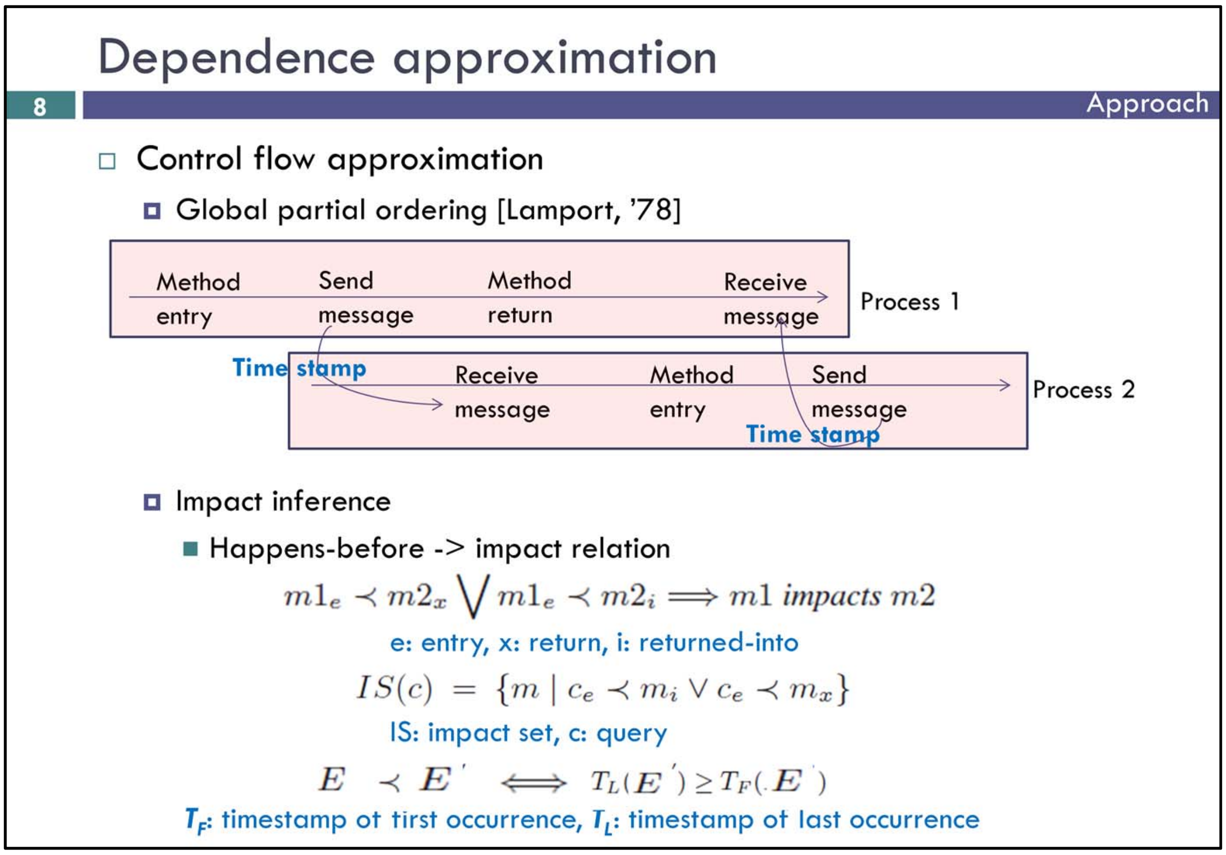


  - Two communication events
    - Message sending, and message receiving
    - Necessary for synchronizing the timing of events across processes

The dependencies between program entities, data or control dependencies, can be safely approximated through control flow as a feasible control flow path from point A to point B must exist for the existence of a dependence between them.

Specifically, we capture execution order of methods in the program by recording three method execution events: entry, return, and returned-into. In fact, for impact analysis, we are mainly concerned about the ordering between the query set and other methods, so we only need a partial ordering.

Now, recording these events is sufficient for deriving a partial ordering of methods within a process, as proved before.
However, as mentioned earlier, different processes are concurrently running on physically separated machines without a global clock. Thus, we also record two types of communication (message passing) events: message sending and message receiving event. We use these events to synchronize the timing of method events across all processes in the distributed system.



Dependence approximation

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

Method entry
Send message
Method return
Receive message
Process 1
Time stamp
Receive message
Method entry
Send message
Time stamp
Process 2

- 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

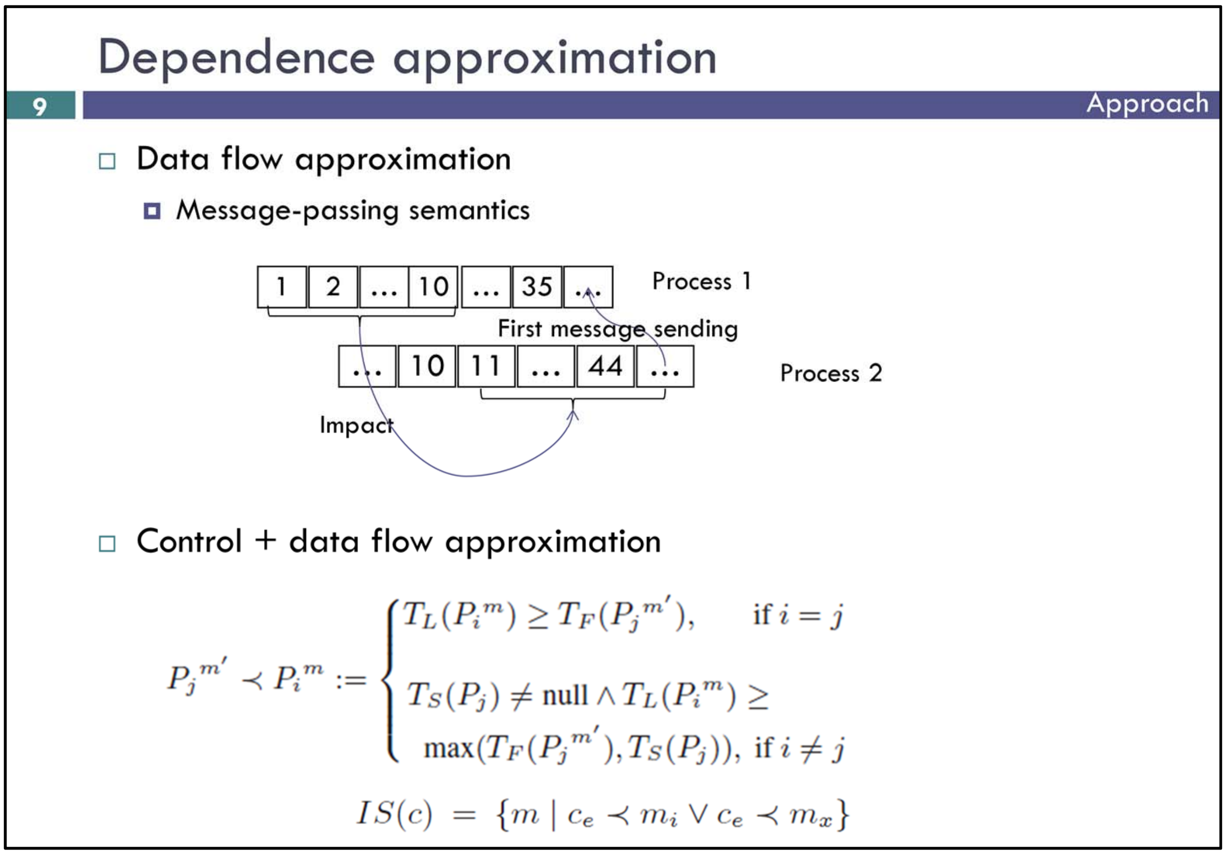
$$E \prec E' \iff T_L(E') \geq T_F(E')$$

T_F: timestamp of first occurrence, T_L: timestamp of last occurrence

Put together, we monitor both method execution events and communication events in each process, and piggyback the current clock value of sending process in the message being sent. When the receiver process receives the message, distIA retrieves the sender's clock and compares the clock with local clock (i.e., the clock of the receiver process) and updates the local clock to the larger and increments it by 1. This process follows the Lamport time-stamping algorithm well-known in distributed systems.

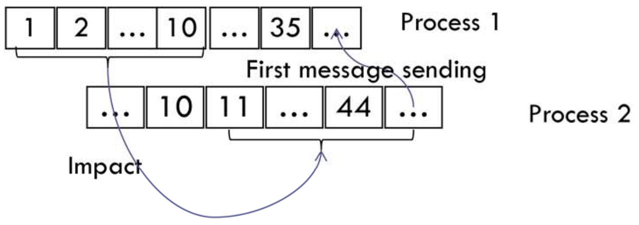
By doing so, we obtain traces of method events that are partially ordered globally within the entire system.
Next, from the this partial ordering, the impact relation between methods can be inferred from the happens-before relation between them based on their partial order. Based on this inference, the impact set of a given query is the set of methods that happens after it.

To determine the happens-before relation between two events, we just need to compare the timestamps of their first/last occurrences. For example, that event *E* happens before *E'* implies that the timestamp of the first instance of E, the first entry event of a method, is smaller than the timestamp of the last instance of *E'*, the last return or returned-into event of a method.



Dependence approximation

- Data flow approximation
  - Message-passing semantics



- Control + data flow approximation

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

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

The approximation based on control flows only is safe, and we know it is also very rough (imprecise), because apparently being executed after the query does not necessarily imply being dependent on the query or getting impacted by changes in the query.
We could do better in the precision while still remaining rapid. Yet, we may not do heavy stuff, expensive data-flow analysis here. Instead, we do a rough data-flow approximation based on a very simple heuristics.

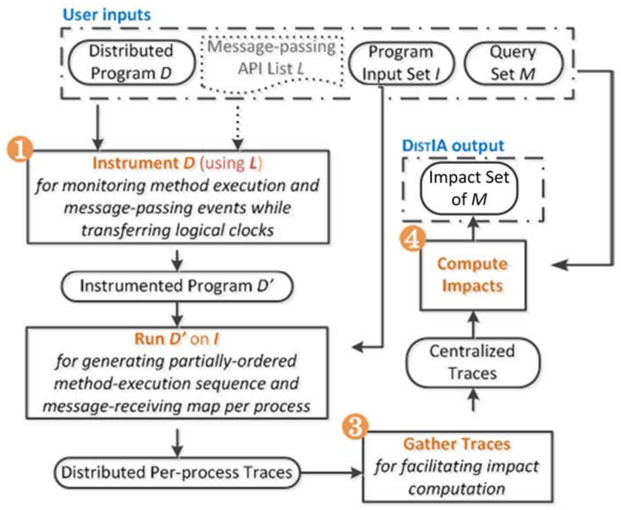
We slightly leverage message-passing semantics. For example, here based on the method-event global partial ordering, methods associated with the events time-stamped with 1, 2, through 10 in process 1 seems to impact methods associated with events in process 2 that are time-stamped with 11 through 44. However, since process 1 never sent any message to process 2, such impacts are false positive apparently.
In another situation, process 2 sends the first message to process 1 after time 44, methods in process 1 whose last execution occurred earlier than that time (such as the method last executed at 35) won't be impacted by methods in process 2 that first executed before time 44, although looking the partial ordering alone would derive such impact relations.

So, put both data and control flow approximation together, this equation provides a unified determination of happens-before relation. Then, my applying this customized happens-before relation to the impact-set computation, we can get more precise results.

DistIA

Approach

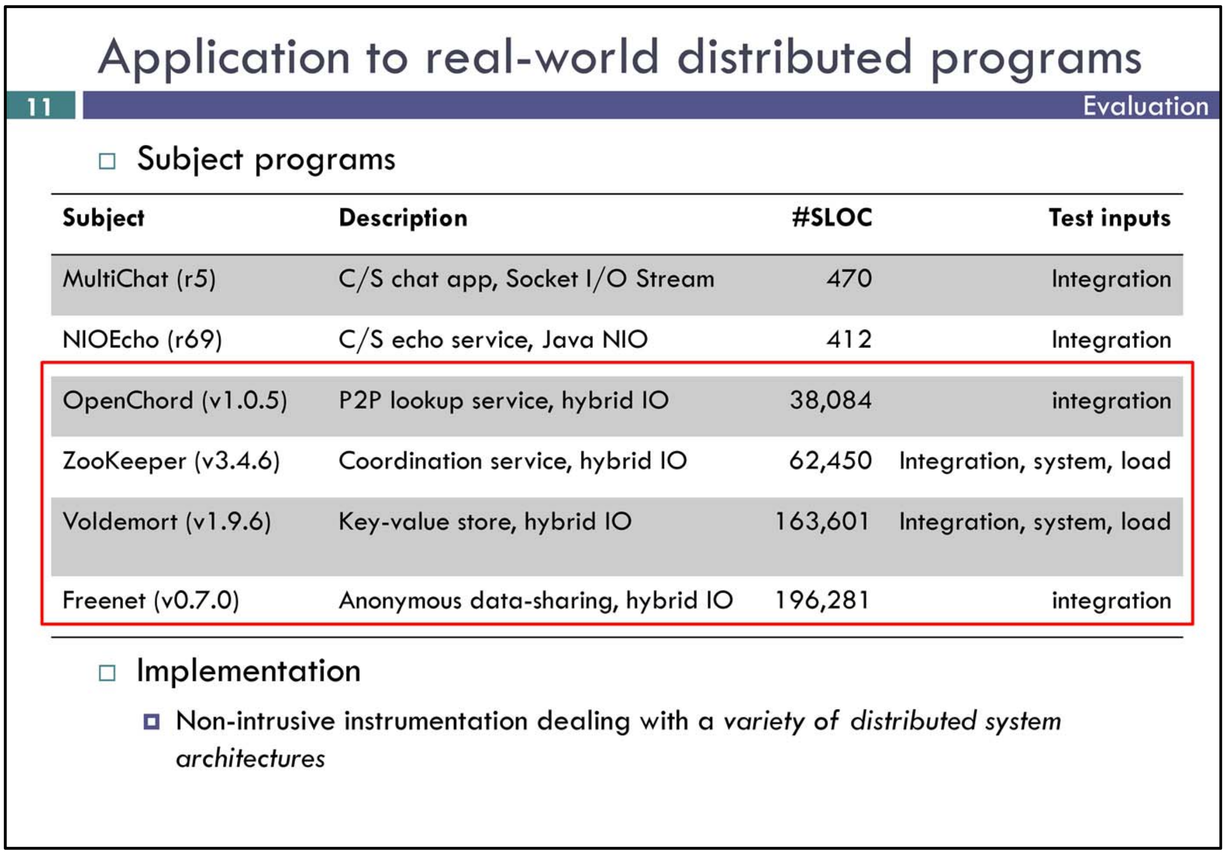
- Workflow


User inputs
Distributed Program D
Message-passing API List L
Program Input Set I
Query Set M
1
Instrument D (using L) for monitoring method execution and message-passing events while transferring logical clocks
Instrumented Program D′
Run D′ on I for generating partially-ordered method-execution sequence and message-receiving map per process
Distributed Per-process Traces
3
Gather Traces for facilitating impact computation
Centralized Traces
4
Compute Impacts
DistIA output
Impact Set of M

- Algorithms
  - Communication event monitoring
  - Impact computation

The following figure shows the overall workflow of our technique. (explain step by step…)
The message-passing API list is an optional input what provides the signatures of APIs that are invoked for inter-process communication in the distributed program. A default list has been built in DistIA that cover commonly used APIs including blocking and non-blocking network I/O API in Java SDK.

The two core modules of the distIA analysis algorithm are a communication event monitor used for partially ordering method execution events globally, and a post-processor for impact computation. Due to the time limit, I will skip the details of these algorithms. If you are interested in those details, please read our paper.



Application to real-world distributed programs

11

Evaluation

- Subject programs

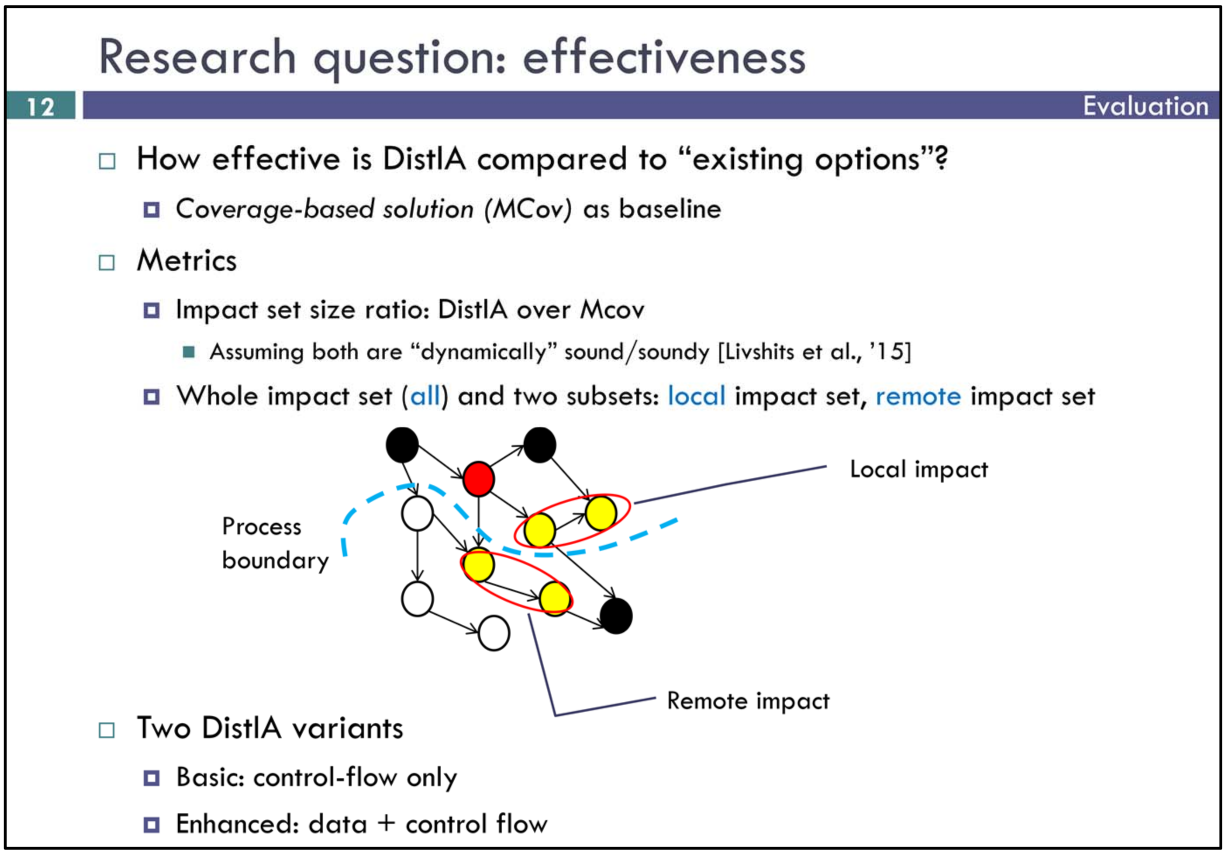
Subject	Description	#SLOC	Test inputs
MultiChat (r5)	C/S chat app, Socket I/O Stream	470	Integration
NIOEcho (r69)	C/S echo service, Java NIO	412	Integration
OpenChord (v1.0.5)	P2P lookup service, hybrid IO	38,084	integration
ZooKeeper (v3.4.6)	Coordination service, hybrid IO	62,450	Integration, system, 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*

We have successfully applied DistIA to real-world distributed programs of various sizes and application domains, including four large distributed software that all adopted hybrid network I/O APIs (non-blocking, through Socket I/O stream, and blocking I/Os, through Java NIO).
For example, Zookeeper is a well-known coordination service, and Voldemort is a distributed data store adopted at LinkedIn for many critical services.

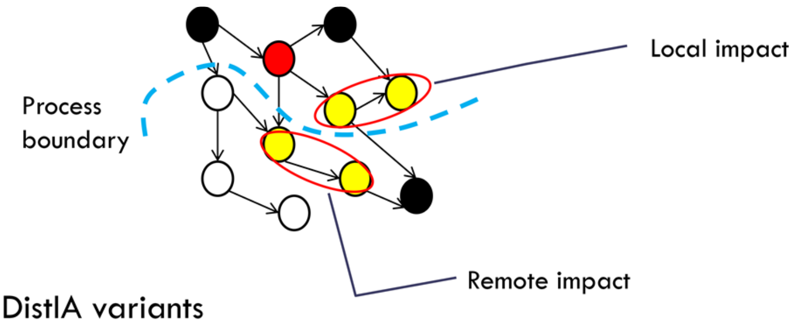
We used system and load tests that come as part of these open-source software packages, and integration tests that we created following official guide provided with these systems. These test inputs help produce execution traces representative of system behaviors.

The implementation of DistIA is non-trivial, detailed discussion on that, esp. the non-intrusive instrumentation that accommodates varied distributed system architectures, can be found in our paper.



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

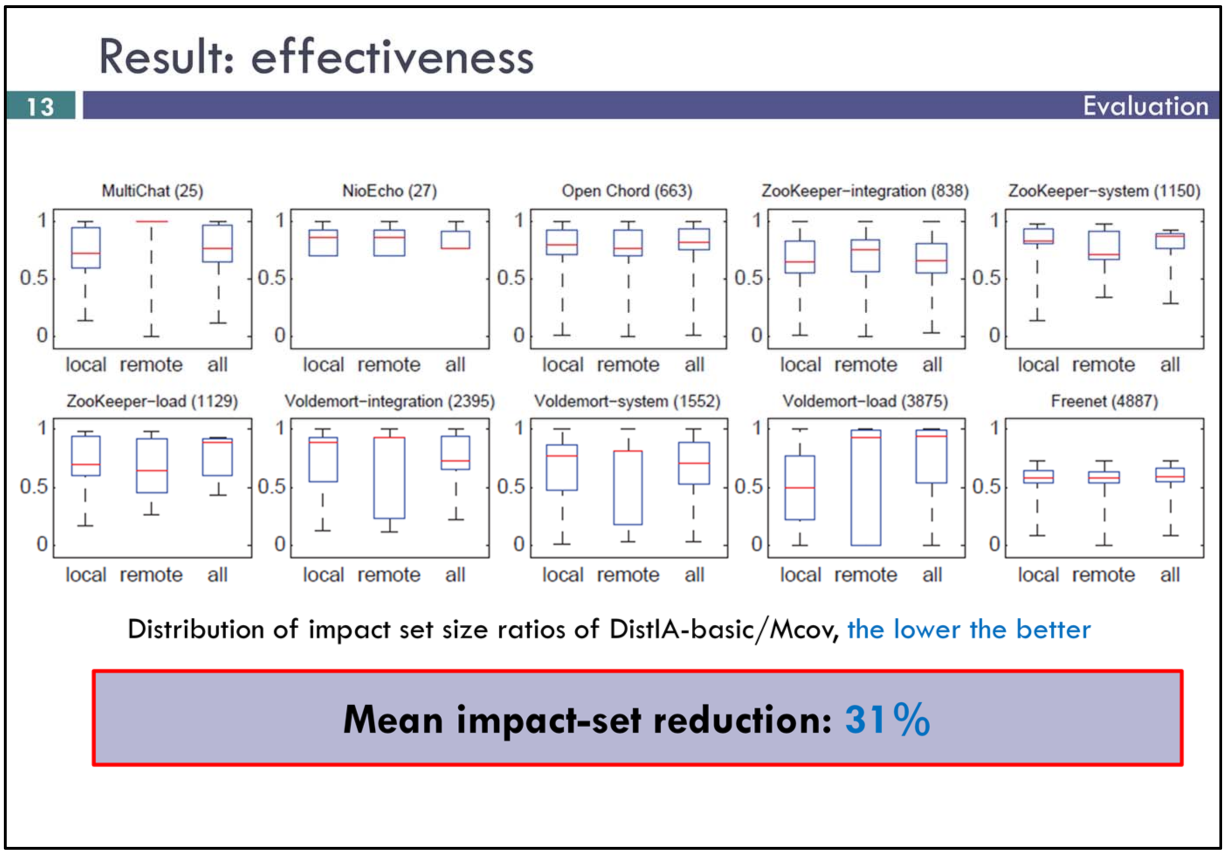


- Two DistIA variants
  - Basic: control-flow only
  - Enhanced: data + control flow

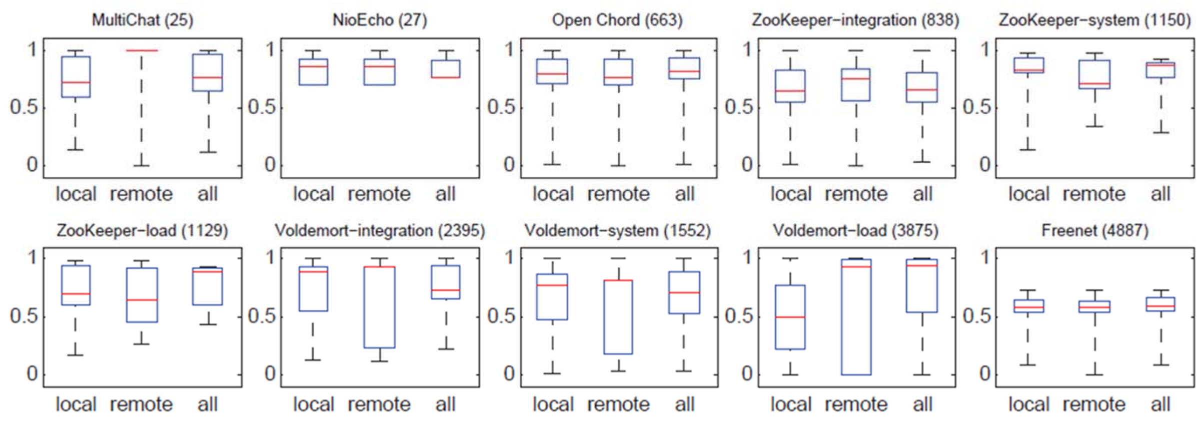
With these subject systems, the first research question is about the effectiveness. Since there is actually no peer solution that is fairly comparable to DistIA, we assume a coverage-based solution, which simply reports all covered methods to be impacted, as the baseline. The goal of course is not to beat the strawman, but to have a reference to help understand the results.

Specifically we measure the effectiveness by impact set size ratio of DistIA over Mcov. We took each executed method as a query, computed its impact set, denoted as 'all', and its two subsets, local and remote impact sets. (show the illustration): just for illustration, the process boundary is the network; local impacts are impacted entities executed in the same process as the query; otherwise, the impacted entities are remote impacts.

To examine how much our simple leverage of message-passing semantics help with the effectiveness, we looked at the results of two variants of distIA: the basic version exploits the method-level control flows only, and the enhanced version additionally exploits the simple data-flow heuristics.



Result: effectiveness



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

Mean impact-set reduction: 31%

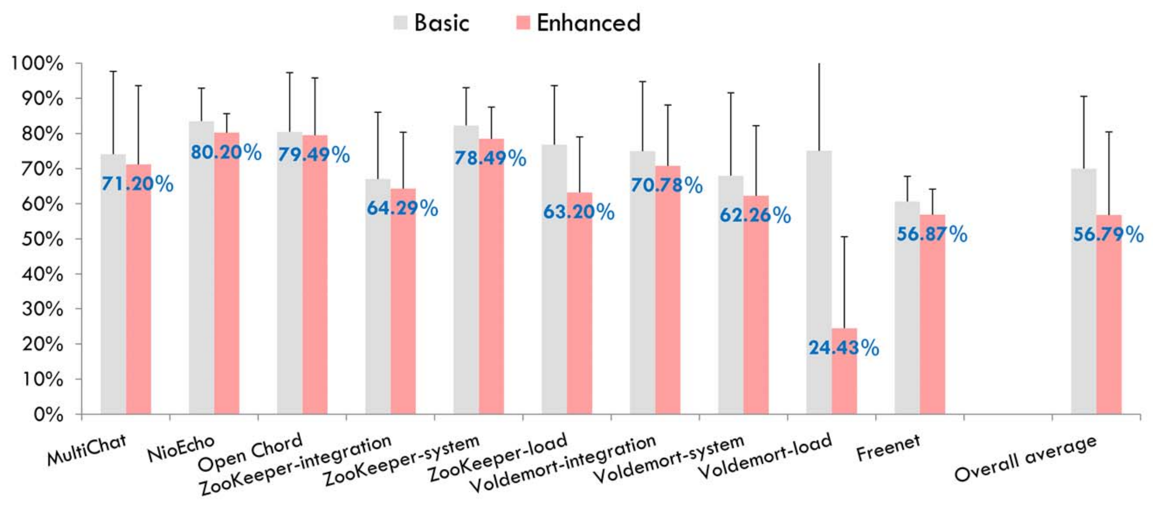
The effectiveness result of the basic version is depicted here by boxplots showing the distribution of the metric values (that is, the impact set size ratios, the lower the better). The X axis lists the three impact sets (all, local and remote), and the Y axis shows the effectiveness of each query.
For subjects having multiple test cases, the figure shows the result per test case, as you can see here from the chart title. The number in the parentheses are the total numbers of queries.

As shown, DistIA performs always noticeably better than the baseline, as expected. Also, in terms of average effectiveness, DistIA works generally better on larger systems. In fact, it worked the best for the largest program Freenet, achieving a steady impact-set reduction of almost 50%.

(Show the banner) Overall, the mean impact set reduction relative to the baseline is 31%.

Result: effectiveness

14 Evaluation



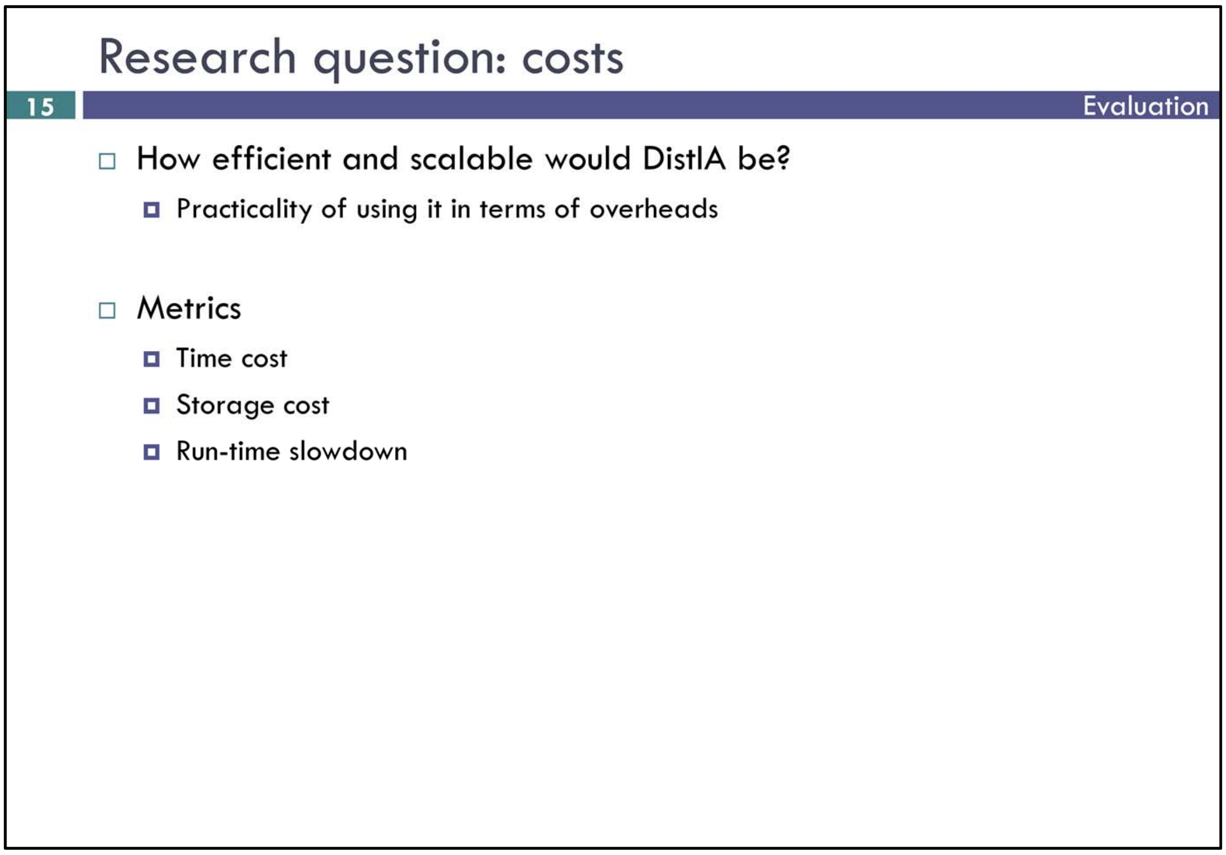
Mean impact set size ratios of DistIA-basic versus DistIA-enhanced, the lower the better

Overall mean impact-set reduction: 43%

To compare the two versions of DistIA, this figure shows the average effectiveness, on the Y axis, of the enhanced version versus the basic one, for each program and test shown on the X axis.
The data labels are the numbers of the enhanced version.

We can see that in some cases the improvement was significant.

(show the banner) Overall, the mean impact-set reduction achieved by utilizing both partial ordering method events and the simple message-passing semantics heuristics is 43%, which is 12% further down when compared to the basic control-flow approximation.



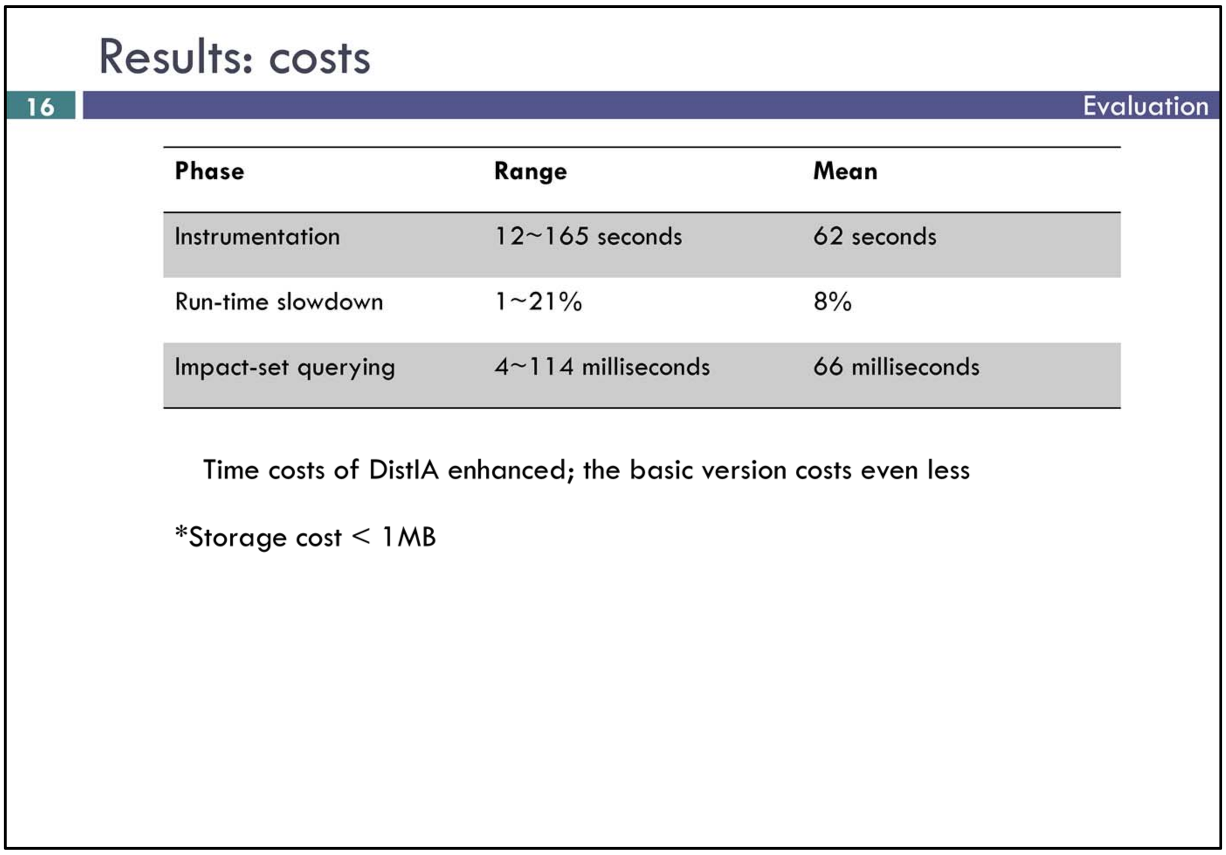
Research question: costs

15

Evaluation

- How efficient and scalable would DistIA be?
  - Practicality of using it in terms of overheads
- Metrics
  - Time cost
  - Storage cost
  - Run-time slowdown

Our second research question concerns the efficiency of our analysis.
(Show the metrics) including the time and storage costs, as well as the run-time slowdown caused by the instrumentation.



Results: costs

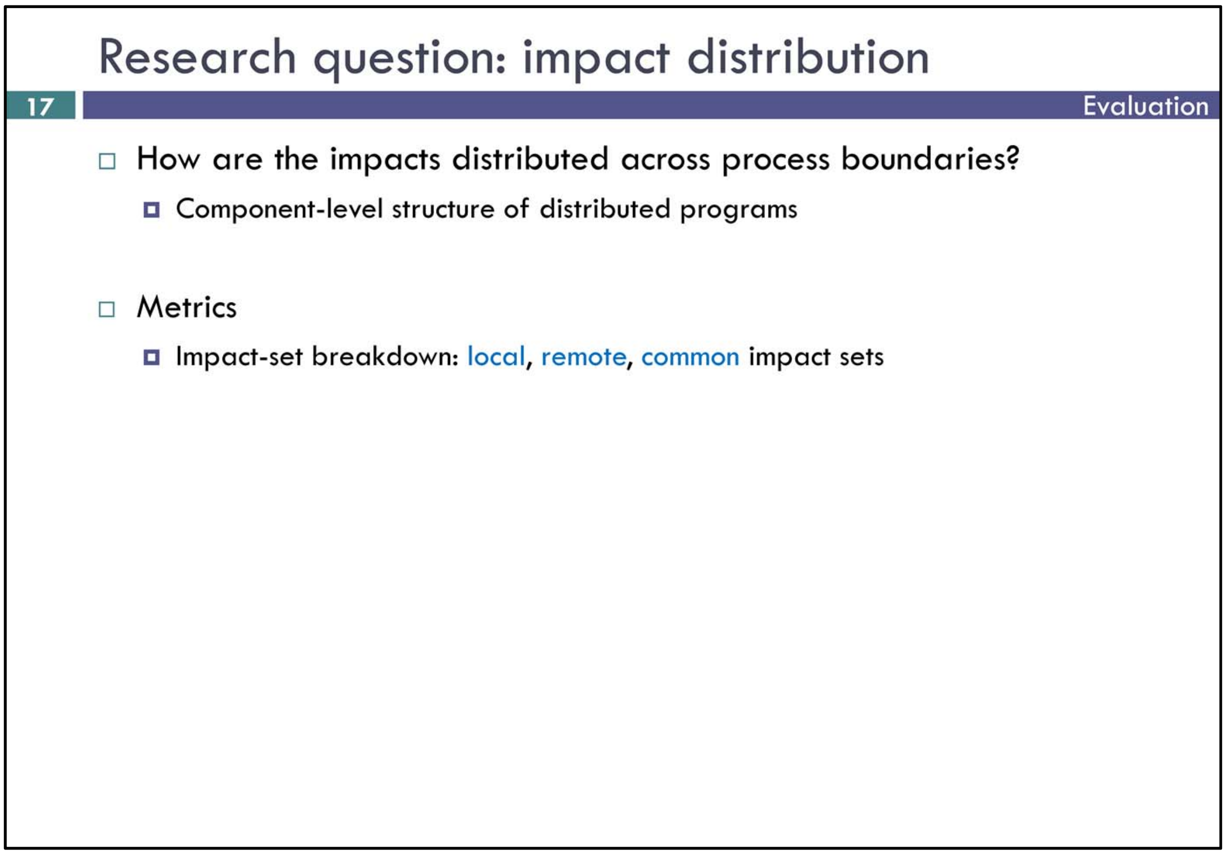
Phase	Range	Mean
Instrumentation	12~165 seconds	62 seconds
Run-time slowdown	1~21%	8%
Impact-set querying	4~114 milliseconds	66 milliseconds

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

*Storage cost < 1MB

Given its lightweight nature, we expected the high efficiency of DistIA, regardless which version is considered.

Typically, the entire analysis can be finished in about one minute, causing 8% run-time overhead and negligible storage space of 1MB (for the execution traces).



Research question: impact distribution

17

Evaluation

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

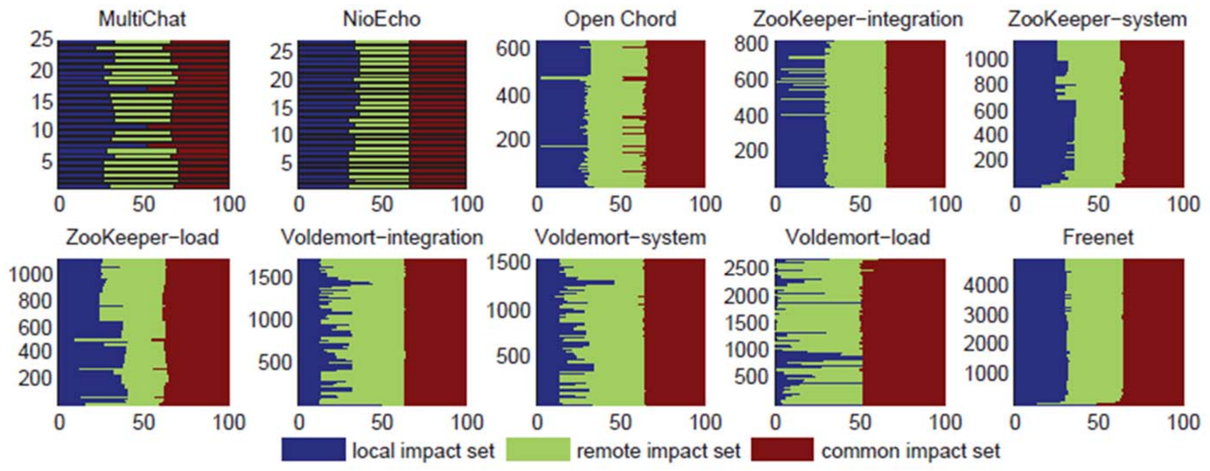
We also wanted to explore how the DistIA results may help with distributed program understanding.

We thus examined the impact distribution across process boundaries,
(show the metrics) looking at the breakdown of each impact set into local and remote impact sets, and their intersection, we call common impact set.

Result: impact distribution

18

Evaluation

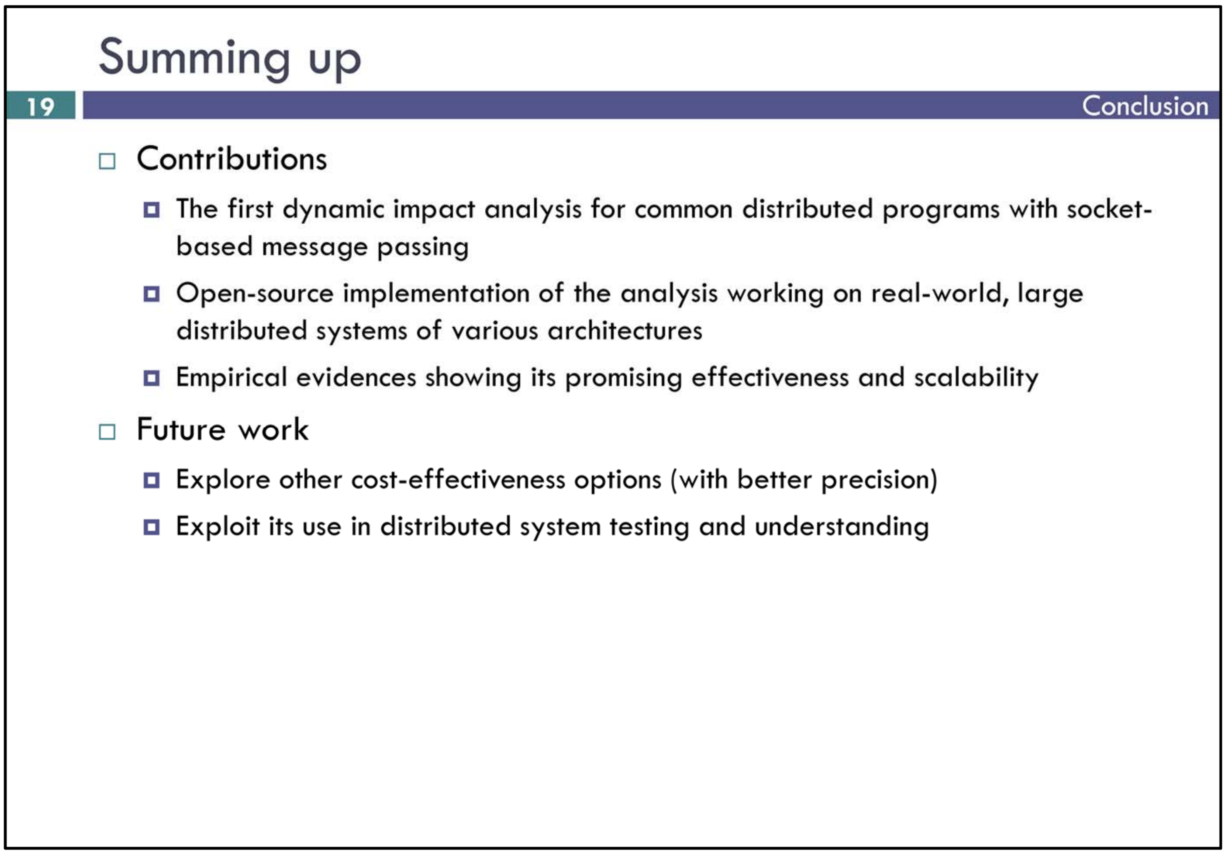


Breakdown of each impact set (given by DistIA-basic) into three disjoint subsets

In each chart, the Y axis shows the no. of each query and X axis is the percentage breakdown of corresponding impact set.
Note here to show the distribution, the common impacts are removed from local and remote impact sets, thus the three subsets are disjoint.

One interesting observation is that impacts commonly propagate beyond local processes and the propagation can be quite significant.

Another observation is that common impact sets are extensive, implying that component-level functionality reuse is pretty common and significant in distributed program executions. The common impact set sizes here could be potentially used as a measure of inter-component couplings.



Summing up

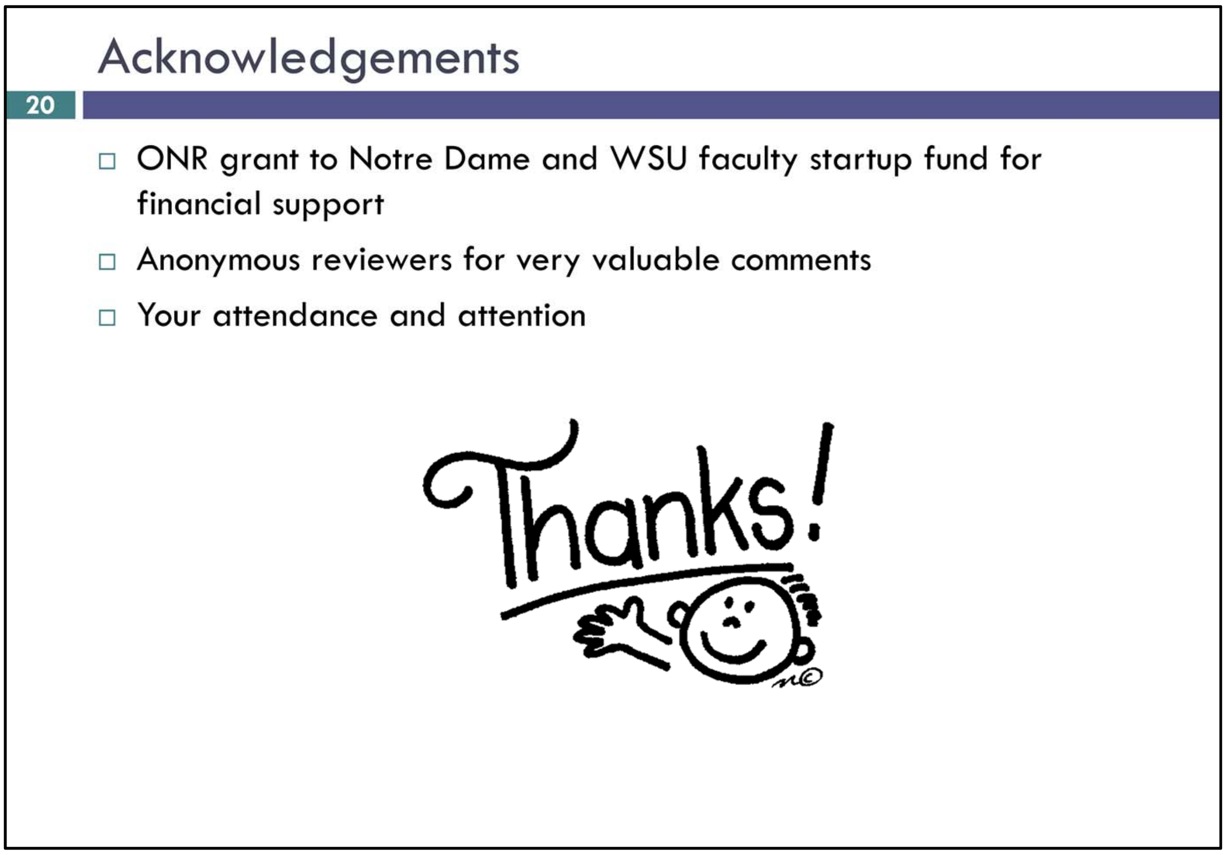
19

Conclusion

- 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

To sum up, DistIA provides the first dynamic impact analysis for common distributed programs, with an open-source implementation that actually works with real-world, large distributed systems of different architectures. We showed its promising cost-effectiveness through extensive empirical evidences.

In the future, we are interested in exploring other cost-effectiveness options based on DistIA, pushing the precision at reasonably higher costs. We are also planning to exploit how it can be used for testing and understanding distributed systems and their run-time behaviors.



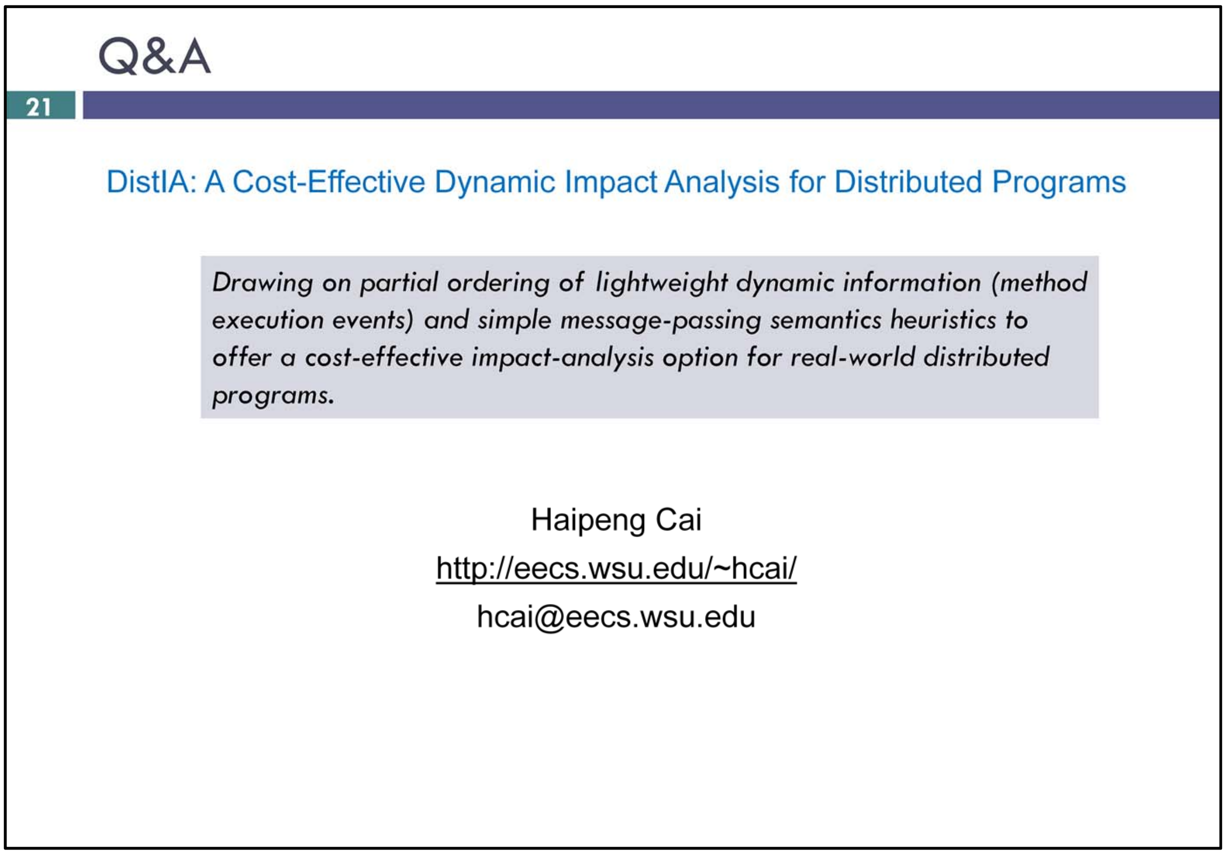
Acknowledgements

20

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

This work has been supported by an ONR grant given to Notre Dame and faculty startup fund given by WSU.

I am grateful to the anonymous reviewers for their very valuable comments. And thank you all again for being here and your attention.



Q&A

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

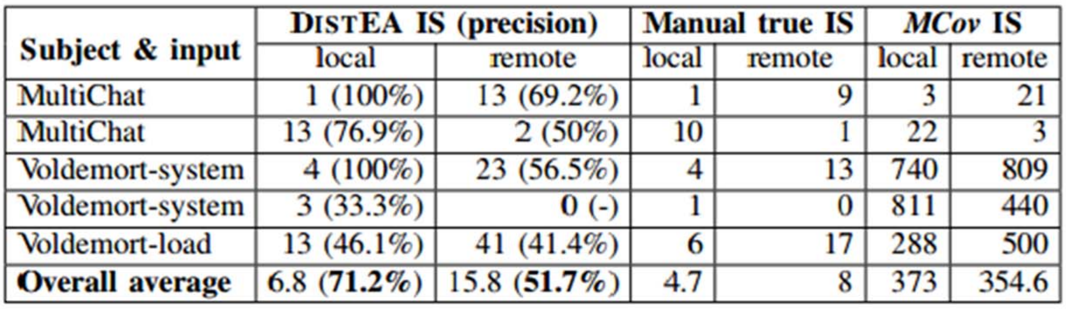
Drawing on partial ordering of lightweight dynamic information (method execution events) and simple message-passing semantics heuristics to offer a cost-effective impact-analysis option for real-world distributed programs.

Haipeng Cai
http://eecs.wsu.edu/~hcai/
hcai@eecs.wsu.edu

Now I would like to take your questions.

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