Efficient Runtime Invariant Checking: A Framework and Case Study

Michael Gorbovitski
Tom Rothamel
Y. Annie Liu
Scott D. Stoller

Computer Science Department
State University of New York at Stony Brook

Invariants

An invariant is a predicate that is expected to be true at all points during program execution

Important for correctness and optimization

- Predicates about the program state:
  e.g. no node has itself as a child

```
foreach (o in extent(Node): o in o.children):
    report("Error: ", o, " has a self-edge.")
    stop()
```

- Predicates about the history of program states:
  no new command is sent while a command is still executing

Runtime Invariant Checking

Checks invariants during program execution
i.e. checks predicates at all program execution points

\+ Can check any invariant

\- Has runtime overhead, especially high if complex invariants are checked naively

Our Framework Supports

```
foreach (query):
  action
recording history
```

- Specifying invariants using high-level queries
  - Invariant : query result is non-empty
  - Recording history data for use in queries

- Analysis and transformations for efficient checking
  - Incremental computation of query results
  - Static alias analysis and type analysis

- Mechanism for triggering *actions* for reporting errors, debugging, and prevention or remediation

Related Work

- Runtime invariant verification

  Behavioral specification languages

  Spec#/Boogie [Barnett06], JML[Leavens05]/jmlc[Cheon03],...

  not incremental for our queries, less expressive, or both

  Logic specification languages

  Jnuke[Artho04], EAGLE [Barringer04], ...

  queries over sequences of events, not data structures
- Incremental query result maintenance

  JQL [Willis06], JQL Incremental Maintenance [Willis08],...

  less expressive, e.g. no membership tests on nested objects and sets.
- AOP

  AOP[Kiczales01] – manually writing pointcuts and advices

Outline

- The problem, framework, related work
- **Specification of invariants using queries**
- **Efficient maintenance of query results**
- Implementation and experiments

Specification of Invariants using Queries

```
foreach (sp in $sending_packets,
            kt in extent(KerberosTicket):
            kt.invalid and kt.ip==sp.target_ip):
     report("Sending ", sp, " with invalid ticket!")
     stop()
de in global: $sending_packets=set()
at $x.send($p):
if type($x)==socket:
do before:
   $sending_packets.add($p)
do after:
   $sending_packets.remove($p)
```

Query

Action

Recording history

Incremental Maintenance of Query Results

```
foreach (query):
  action
```

- For every kind of update to the query's underlying sets and objects:
  generate program transformation rule that specifies how to incrementally update the query result

- For updates to the query's underlying sets and objects actually in the subject program:
  apply rules to incrementally maintain the query result
  static analysis reduces number of runtime checks

- When a new element is added to the query result, run the *action*

Generating Program Transformation Rules

```
foreach (query):
  action
```

```
foreach (
    sp in $sending_packets,
    kt in extent(KerberosTicket):
    kt.invalid and kt.ip==sp.target_ip):
  action
```

Query

↓

```
for sp in $sending_packets:
   for kt in extent(KerberosTicket):
      if kt.ip==sp.target_ip:
         if kt.invalid :
            action
```

Naive checking code

Generating Program Transformation Rules

```
for sp in $sending_packets:
    for kt in extent(KerberosTicket):
        if kt.ip==sp.target_ip :
            if kt.invalid:
                action
```

↓

```
at $sending_packets.add($sp) :
    for $k in revmapK[$sp.target_ip]:
        mapS2K[$sp].add($k)
    if $sp not in $sending_packets :
        for $k in mapS2K[$sp]:
            if $k.invalid:
                action
```

1. Eliminate loops over the updated sets
2. Use auxiliary maps to replace loops/tests over sets that are joined with the updated sets with lookups
3. Leave remaining tests
4. Update auxiliary maps when necessary

May-Alias Analysis – For Update Detection

- Only insert maintenance code at places where query results could be affected
- Compute pairs of variables and fields that may alias each other.
  - If not aliased to data that the query depends on, cannot affect results
- Uses and extends [Goyal05]
- Interprocedural, object-oriented, flow-sensitive, derivation context-sensitive
- Time complexity : $O(n^3)$

Type Analysis – For Precise Update Detection

- Do not insert maintenance code at places where query results cannot be affected

- Infer types of all expressions statically
  - If the type of expression is different than type of anything in the query, cannot affect results

- Type analysis
  - distinguishes between constants, etc
  - supports union types, e.g. *union(int(1), int(2))*

- Time complexity : $O(n \times s)$

Implementation

- Checks invariants in Python programs
  - 5000 lines of Python code
  - Takes seconds to generate rules
  - Applied to programs up to 80KLOC

- InvTS – the engine that applies generated transformation rules to subject programs
  - 18000 lines of Python code
  - Takes tens of seconds to apply rules
  - Applied to programs up to 80KLOC

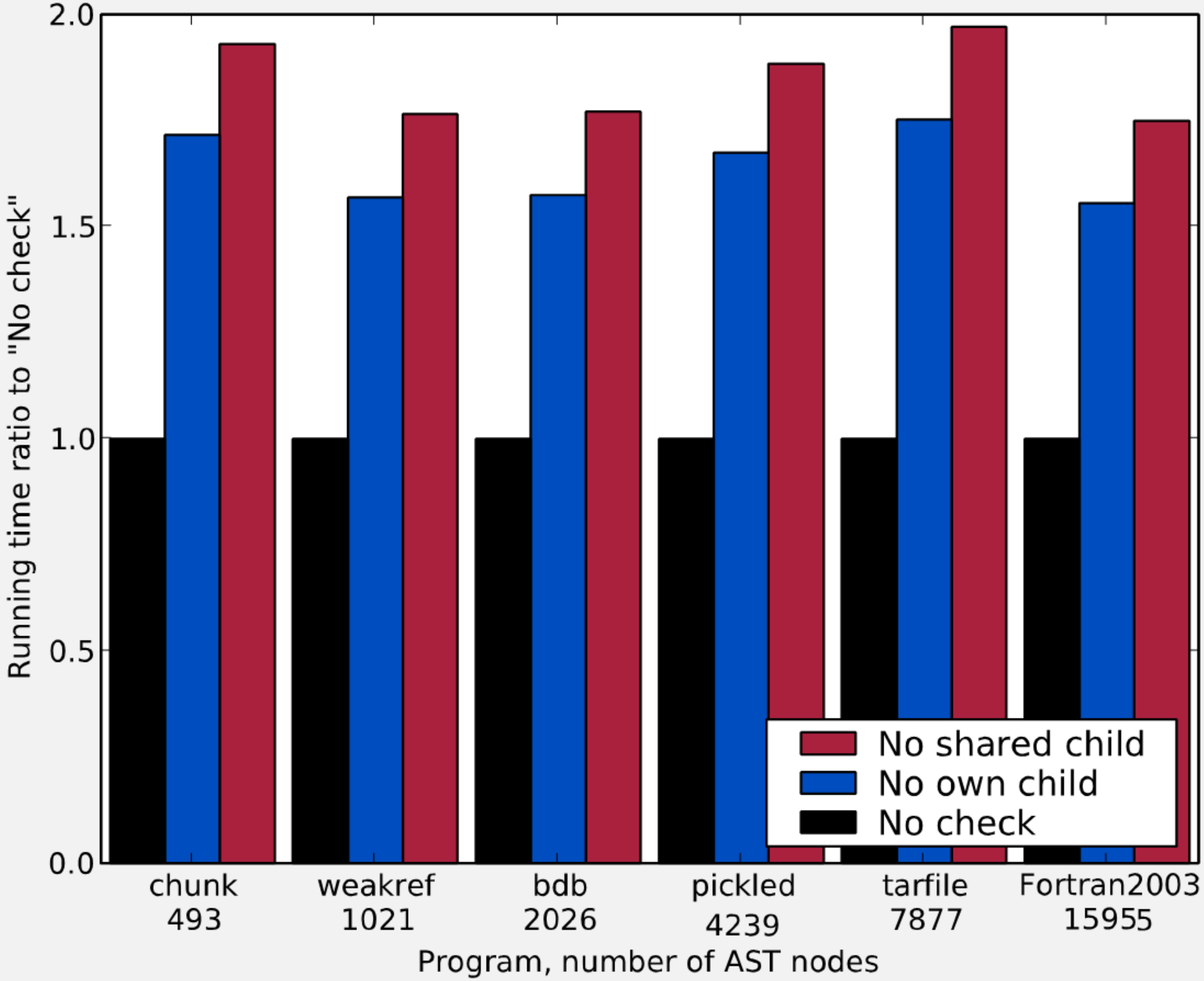
Experiments – Checking Invariants

- AST Transformations performed by InvTS – inputs from 493 to 15955 AST nodes
  - Not own child – no node has itself as a child
  - Not shared child – no two nodes have the same child

- Authentication performed by Python Samba client
  - Require valid ticket – no packets sent with an invalid ticket
  - Repeated authentication – no gratuitous reauthentication

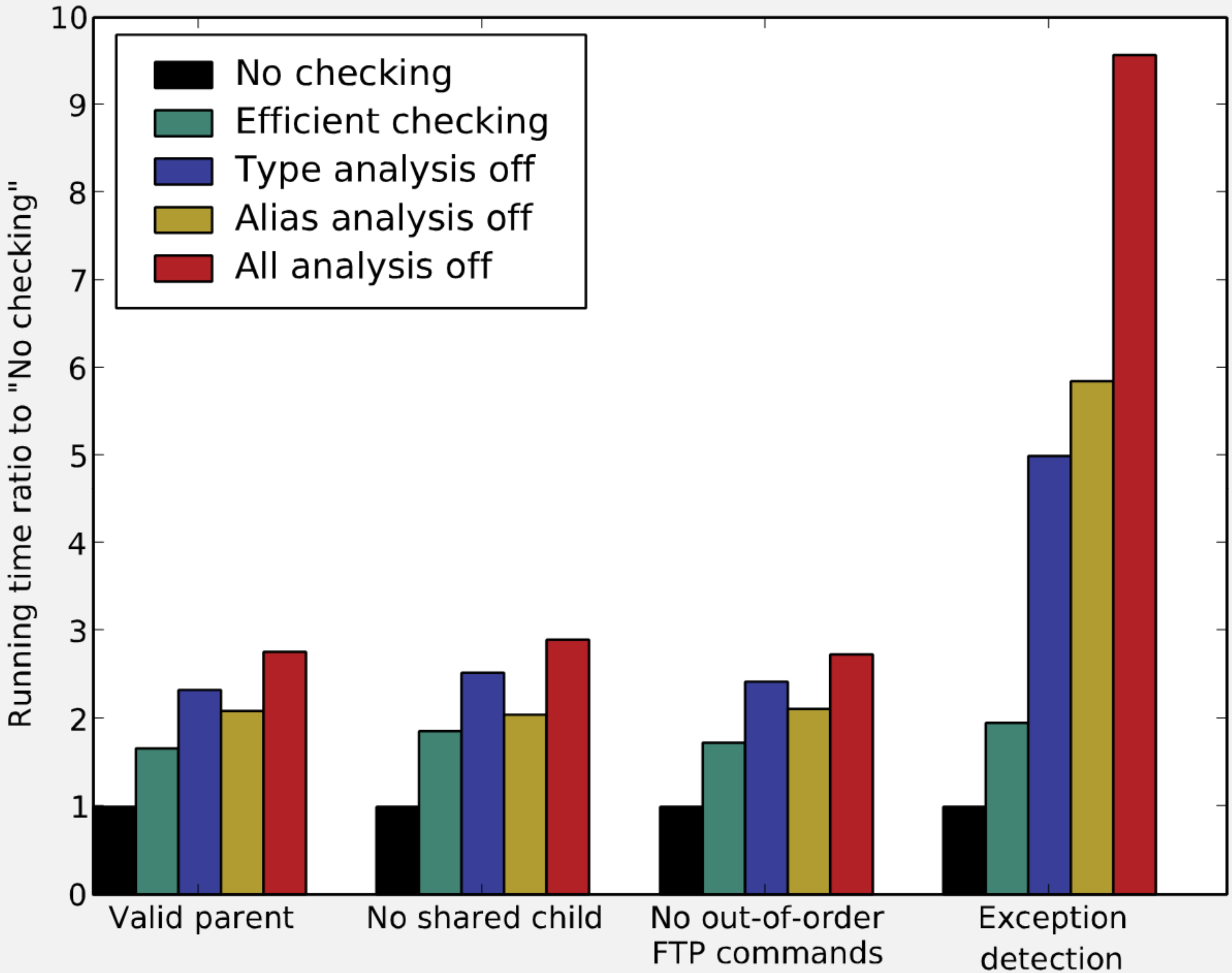
- File distribution protocol (BitTorrent)
  - No duplicate data – no unneeded duplication of data
  - No packets changed in transit – md5 of payload unchanged

Experiments – Runtime Overhead of Invariant Checking



Non-incremental versions take more than 20 min vs. 1/2 min for “No check”

Experiments – Benefits of Static Analysis



Conclusion

- An efficient runtime invariant checking framework
  - Incrementally maintaining query results drastically reduces overhead of runtime invariant checking
  - Deriving rules from queries allows the programmer to declaratively specify invariants using queries
  - Type and alias analysis provide significant further reduction of overhead in our experiments

- Other recent and on-going work
  - InvTS, Python and C program transformation system

    Generating optimized implementations, instrumentation, ...
  - Efficient query-based debugging [SCAM'08]
  - More general incrementalization technique [GPCE'08]