"Systemized" Program Analyses – A "Big Data" Perspective on Static Analysis Scalability

Harry Xu and Zhiqiang Zuo
University of California, Irvine
A Quick Survey

• Have you used a static program analysis? What did you use it for?
• Have you designed a static program analysis?
• What are your major analysis infrastructures?
• Have you been bothered by its poor scalability?
This Tutorial Is About

- Big data (graphs)
- Systems
- Static analysis
- SAT solving
This Tutorial Is About

• What inspiration can we take from the big data community?

• How shall we shift our mindset from developing scalable analysis algorithms to developing scalable analysis systems?
Outline

• Background: big data/graph processing systems
• Treating static analysis as a big data problem
• Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
• BigSAT: distributed SAT solving at scale
Intimacy Between Systems and App. Areas

- Machine Learning
- Information Retrieval
- Bioinformatics
- Sensor Networks

......
Large-Scale Graph Processing: Input

- Social network graphs
  - Twitter, Facebook, Friendster

- Bioinformatics graphs
  - Gene regulatory network (GRN)

- Map graphs
  - Google Map, Apple Map, Baidu Map

- Web graphs
  - Yahoo Webmap, UKDomain
Large-Scale Graph Processing: Input Size

• Social network graphs
  – Facebook: 721M vertices (users), 68.7B edges (friendships) in May 2011

• Map graphs
  – Google Map: 20 petabytes of data

• Web graphs
  – Yahoo Webmap: 1.4B websites (vertices) and 6.4B links (edges)
What Do These Numbers Mean

[To analyze the Facebook graph] calculations were performed on a Hadoop cluster with 2,250 machines, using the Hadoop/Hive data analysis framework developed at Facebook.

– Ugander et al., The Anatomy of the Facebook Social Graph, arXiv:1111.4503, 2011
Large-Scale Graph Processing: Core Idea

- Shift our mind from developing specialized graph algorithms to developing simple programs powered by large-scale systems.
- Gather-apply-scatter: a graph-parallel abstraction

Think like a vertex

PageRank (Vertex v) {
    foreach (e in v.inEdge) {
        total += e.value;
    }
    v.value = 0.15 * (0.85 + total);
    foreach (e in v.outEdge) {
        e.value = v.value;
    }
}
Large-Scale Graph Processing: Classification I

• Distributed systems
  – GraphLab, PowerGraph, PowerLira, GraphX, Gemini
  – Challenges in communication reduction and partitioning

• Single machine systems
  – Shared memory: Ligra, Galois
  – Out of core: GraphChi, X-Stream, GridGraph, GraphQ
  – Challenges in disk I/O reduction
Large-Scale Graph Processing: Classification II

• **Vertex-centricity**
  – When computation is performed for a vertex, all its incoming/outgoing edges need to be available
  – GraphChi, PowerGraph, etc.

• **Edge-centricity**
  – Computation is divided into several phases
  – Vertex computation does not need all edges available
  – X-Stream, GridGraph, etc.
One Stone, Two Birds

• Present a simple interface to the user, making it easy to develop graph algorithms

• Push performance optimizations down to the system, which leverages parallelism and various kinds of support to improve performance and scalability
Outline

• Background: big data/graph processing systems
• Treating static analysis as a big data problem
• Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
• BigSAT: distributed SAT solving at scale
Where Is PL’s Position in Big Data?

Programming languages is a big source of data.
PL Is Another Source of Big Data

System Solutions

Big Data Systems

PL Problems

SAT Solver, Program Analysis, Model Checking, …

Existing Work

Our Work

Scalable Results
Static Analysis Scalability Is A Big Concern

• An important PL problem: Context-sensitive static analysis of very large codebases

- Pointer/alias analysis
- Dataflow analysis
- May/must analysis
- ...
Context-Free Language (CFL) Reachability

- A program graph $P$

  \[ c \text{ is } K\text{-reachable from } a \]

- A context-free Grammar $G$ with balanced parentheses properties

  \[ K \rightarrow l_1 l_2 \]

Reps, Program analysis via graph reachability, IST, 1998
A Wide Range of Applications

• Pointer/alias analysis

• Dataflow analysis, pushdown systems, set-constraint problems can all be converted to context-free-language reachability problems

Source: Sridharan and Bodik, Refinement-based context-sensitive pointsto analysis for Java, *PLDI*, 2006
A Wide Range of Applications (Cont.)

• Pointer/alias analysis

\[
\begin{align*}
\text{b} &= \& \text{a}; \quad \text{// Address-of} \\
\text{c} &= \text{b}; \\
\text{d} &= \text{c}; \quad \text{// Dereference}
\end{align*}
\]

• \textit{Address-of} $\&$ / \textit{dereference} $\ast$ are the open/close parentheses

Sridharan and Bodik, Refinement-based context-sensitive pointsto analysis for Java, \textit{PLDI}, 2006
A Typical PL Problem

• **Traditional Approach**: a worklist-based algorithm
  – the worklist contains reachable vertices
  – no transitive edges are added physically

• **Problem**: embarrassingly sequential and unscalable

• **Solution**: develop approximations

• **Problem**: less precise and still unscalable
No Worry About Memory Blowup

• As long as one knows how to use disks and clusters

• Big Data thinking:

  Solution =

  (1) Large Dataset + (2) Simple Computation +

  System Design
Outline

• Background: big data/graph processing systems
• Treating static analysis as a big data problem
• Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
• BigSAT: distributed SAT solving at scale
Turning Big Code Analysis into Big Data Analytics

• Key insights:
  – Adding transitive edges \textit{explicitly} – satisfying (1)
  – Core computation is \textit{adding edges} – satisfying (2)
  – Leveraging disk support for memory blowup

• Can existing graph systems be directly used?
  – No, none of them support dynamic addition of a lot of edges
    
    (1) Online edge duplicate check and (2) dynamic graph repartitioning
Graspan: A Graph System for Interprocedural Static Analysis of Large Programs

• Scalable
  – Disk-based processing on the developer's work machine

• Parallel
  – Edge-pair centric computation

• Easy to implement a static analysis
  – Developer only needs to generate graphs in mechanical ways and provide a context-free grammar to implement the analysis

4 students + 1 postdoc, 1.5 years of development; implemented in both Java and C++
https://github.com/Graspan/
How It Works?

- Comparisons with a single-machine Datalog engine:
  - Graspan is a single-machine, out-of-core system
  - Graspan provides better locality and scheduling
  - Graspan is 3X faster than LogicBlox and 5X faster than SociaLite even on small graphs
Granspan Design

Preprocessing

Edge-Pair Centric Computation

Post-Processing

- Partitions are of similar sizes
- Each partition contains an adjacency list of edges
- Edges in each partition are sorted
Computation Occurs in Supersteps

Preprocessing — Edge-Pair Centric Computation — Post-Processing
Each Superstep Loads Two Partitions

Grammar: $C := AB \quad D := BC \quad B := AD \quad A := CD$
Each Superstep Loads Two Partitions

We keep iterating until delta is 0.

Grammar: $C = AB$, $B = BC$, $E = AD$, $A := CD$
Post-Processing

• Repartition oversized partitions to maintain balanced load on memory
• Save partitions to disk
• Scheduler favors in-memory partitions and those with higher *matching degrees*
What We Have Analyzed

<table>
<thead>
<tr>
<th>Program</th>
<th>#LOC</th>
<th>#Inlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 4.4.0-rc5</td>
<td>16M</td>
<td>31.7M</td>
</tr>
<tr>
<td>PostgreSQL 8.3.9</td>
<td>700K</td>
<td>290K</td>
</tr>
<tr>
<td>Apache httpd 2.2.18</td>
<td>300K</td>
<td>58K</td>
</tr>
</tbody>
</table>

- With
  - A fully context-sensitive pointer/alias analysis
  - A fully context-sensitive dataflow analysis

- On a Dell Desktop Computer with 8GB memory and 1TB SSD
Evaluation Questions and Answers I

• Can the interprocedural analyses improve D. Englers’ checkers?
  – Found 85 new NULL pointer bugs and 1127 unnecessary NULL tests in Linux 4.4.0-rc5
Evaluation Questions and Answers II

• Sample bugs

(a) NULL deref in kernel/kthread.c

```c
void*probe_kthread_data(
  task_struct *task){
  void *data = NULL;
  probe_kernel_read(&data);
  //data will be
dereferenced after
  return.*/
  return data;
}
long probe_kernel_read
(void *dst){
  if(....)
    return -EFAULT;
  return
  __probe_kernel_read(dst);
}
```

(b) NULL deref in mm/swapfile.c

```c
#define page_private(page) ((page)->private)
bool swap_count_continued(...){
  head=vmalloc_to_page(...);
  if(page_private(head)
    != ...){
    ...
  }
  page=vmalloc_to_page(...){
    page *page = NULL;
    if (!pgd_none(*pgd)){
      //...
    }
    return page;
  }
```

Evaluation Questions and Answers III

- Bug breakdown in modules

<table>
<thead>
<tr>
<th>Modules</th>
<th>NULL pointer defs</th>
<th>Unnecessary NULL Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch</td>
<td>0</td>
<td>75</td>
</tr>
<tr>
<td>crypto</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>init</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>kernel</td>
<td>4 (2)</td>
<td>65</td>
</tr>
<tr>
<td>mm</td>
<td>3 (0)</td>
<td>84</td>
</tr>
<tr>
<td>security</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>block</td>
<td>6 (2)</td>
<td>31</td>
</tr>
<tr>
<td>fs</td>
<td>19 (3)</td>
<td>84</td>
</tr>
<tr>
<td>ipc</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>lib</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>net</td>
<td>10 (8)</td>
<td>269</td>
</tr>
<tr>
<td>sound</td>
<td>15 (5)</td>
<td>83</td>
</tr>
<tr>
<td>drivers</td>
<td>25 (3)</td>
<td>286</td>
</tr>
<tr>
<td>Total</td>
<td>108 (23)</td>
<td>1127</td>
</tr>
</tbody>
</table>
Evaluation Questions and Answers IV

- Is Graspan efficient and scalable?
  - Computations took 11 mins – 12 hrs
Evaluation Questions and Answers V

- Graspan v/s other engines?
  - GraphChi crashed in 133 secs

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Graspan</th>
<th>ODA [101]</th>
<th>SociaLite [45]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CT</td>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>Linux-P</td>
<td>99.7 mins</td>
<td>46.6 secs</td>
<td>OOM</td>
</tr>
<tr>
<td>Linux-D</td>
<td>713.8 mins</td>
<td>4.2 mins</td>
<td>OOM</td>
</tr>
<tr>
<td>PostgreSQL-P</td>
<td>353.1 mins</td>
<td>4.5 mins</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>PostgreSQL-D</td>
<td>143.8 mins</td>
<td>57.1 secs</td>
<td>-</td>
</tr>
<tr>
<td>httpd-P</td>
<td>497.9 mins</td>
<td>7.6 mins</td>
<td>&gt; 1 day</td>
</tr>
<tr>
<td>httpd-D</td>
<td>11.3 mins</td>
<td>3.3 secs</td>
<td>4 hrs</td>
</tr>
</tbody>
</table>

**Evaluation Questions and Answers VI**

- **How easy to use Graspan?**
  - 1K LOC of C++ for writing each of points-to and dataflow graph generators
  - Provide a grammar file

- **Data structure analysis in LLVM**
  - More than 10K lines of code
Download and Use Graspan

- [https://github.com/Graspan](https://github.com/Graspan)

- Two versions available at GitHub
  - [https://github.com/Graspan/graspan-cpp](https://github.com/Graspan/graspan-cpp)
  - [https://github.com/Graspan/graspan-java](https://github.com/Graspan/graspan-java)

- Data structure analysis in LLVM
  - More than 10K lines of code
Outline

• Background: big data/graph processing systems
• Treating static analysis as a big data problem
• Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
• BigSAT: distributed SAT solving at scale
Outline

• Preliminaries
• DPLL & CDCL
• Parallelizability of SAT solving
• BigSAT
Boolean Satisfiability Problem (SAT)

- A propositional formula is built from propositional variables, operators (and, or, negation) and parentheses.

\[(x_1' \lor x_2') \land (x_1' \land x_2 \land x_3') \land (x_1' \land x_3 \land x_4') \land (x_1 \lor x_4)\]

- SAT problem
  - Given a formula, find a satisfying assignment or prove that none exists.
CNF formula

\[(x_1' \lor x_2') \land (x_1' \lor x_2 \lor x_3') \land (x_1' \lor x_3 \lor x_4') \land (x_1 \lor x_4)\]

- **Literal**: a variable or negation of a variable
- **Clause**: a disjunction of literals
- **CNF**: a conjunction of clauses
Why is SAT important?

• Theoretically,
  – First NP-completeness problem [Cook, 1971]

• Practically,
  – Hardware/software verification
  – Model checking
  – Cryptography
  – Computational biology
  – …

Cook, The complexity of theorem-proving procedures, TOC, 1971
DPLL

• Backtrack search

• Boolean constraint propagation (BCP)

\[(x_1') \land (x_1 \lor x_2) \land (x_2' \lor x_3')\]
DPLL

• Backtrack search

• Boolean constraint propagation (BCP)

\[(x_1') \land (x_1 \lor x_2) \land (x_2' \lor x_3') \Rightarrow x_1 = F\]
DPLL

• Backtrack search

• Boolean constraint propagation (BCP)

\[(x_1') \land (x_1 \lor x_2) \land (x_2' \lor x_3') \implies x_1 = F \land x_2 = T\]

Davis, Logemann and Loveland. A machine program for theorem proving. CACM, 1962
DPLL

• Backtrack search
• Boolean constraint propagation (BCP)

\[(x_1') \land (x_1 \lor x_2) \land (x_2' \lor x_3') \Rightarrow x_1 = F \land x_2 = T\]

Davis, Logemann and Loveland. A machine program for theorem proving. CACM, 1962
DPLL

• Backtrack search
• Boolean constraint propagation (BCP)
  \((x_1') \land (x_1 \lor x_2) \land (x_2' \lor x_3') \Rightarrow x_1=F \ x_2=T \ x_3=F\)

• Algorithm
  – Select a variable and assign T or F
  – Apply BCP
  – If there’s a conflict, backtrack to previous decision level
  – Otherwise, continue until all variables are assigned

Davis, Logemann and Loveland. A machine program for theorem proving. CACM, 1962
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'
\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
\( x_1 + x_4 \)
\( x_1 + x_3' + x_8' \)
\( x_1 + x_8 + x_{12} \)
\( x_2 + x_11 \)
\( x_7' + x_3' + x_9 \)
\( x_7' + x_8 + x_9' \)
\( x_7 + x_8 + x_{10'} \)
\( x_7 + x_{10} + x_{12'} \)
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'

x1 = 0, x4 = 1
x3 = 1

x1

x3

x3 = 1
\(x_1 + x_4\)
\(x_1 + x_3' + x_8'\)
\(x_1 + x_8 + x_{12}\)
\(x_2 + x_{11}\)
\(x_7' + x_3' + x_9\)
\(x_7' + x_8 + x_9'\)
\(x_7 + x_8 + x_{10'}\)
\(x_7 + x_{10} + x_{12'}\)
\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_12 \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'

x1
x2
x3

x1 = 0, x4 = 1
x2 = 0
x3 = 1, x8 = 0, x12 = 1
\[
x_1 + x_4
x_1 + x_3' + x_8'
x_1 + x_8 + x_{12}
x_2 + x_{11}
x_7' + x_3' + x_9
x_7' + x_8 + x_9'
x_7 + x_8 + x_{10'}
x_7 + x_{10} + x_{12'}
\]
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'

x1 = 0
x2 = 0
x3 = 1
x4 = 1
x7 = 1
x8 = 1
x11 = 1
x12 = 1
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'

x1 = 0, x4 = 1
x3 = 1, x8 = 0, x12 = 1
x2 = 0, x11 = 1
x7 = 1, x9 = 1, 0
\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
\( x_1 + x_4 \)
\( x_1 + x_3' + x_8' \)
\( x_1 + x_8 + x_{12} \)
\( x_2 + x_{11} \)
\( x_7' + x_3' + x_9 \)
\( x_7' + x_8 + x_9' \)
\( x_7 + x_8 + x_{10'} \)
\( x_7 + x_{10} + x_{12'} \)
\( x_1 + x_4 \)
\( x_1 + x_3' + x_8' \)
\( x_1 + x_8 + x_{12} \)
\( x_2 + x_{11} \)
\( x_7' + x_3' + x_9 \)
\( x_7' + x_8 + x_9' \)
\( x_7 + x_8 + x_{10'} \)
\( x_7 + x_{10} + x_{12'} \)
Conflict-driven clause learning (CDCL)

- Clause learning from conflicts
- Non-chronological backtracking
- Algorithm
  - Select a variable and assign T or F
  - Apply BCP
  - If there’s a conflict, conflict analysis to learn clauses and backtrack to the appropriate decision level
  - Otherwise, continue until all variables are assigned
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'
x3' + x7' + x8

x1 = 0, x4 = 1
x3 = 1, x8 = 0, x12 = 1
x2 = 0, x11 = 1
x7 = 1, x9 = 1, 0

x3 = 1 ∧ x7 = 1 ∧ x8 = 0
(x3 = 1 ∧ x7 = 1 ∧ x8 = 0)'
x3' + x7' + x8
\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]
\[ x_3' + x_7' + x_8 \]

Backtrack to the decision level \( x_3 = 1 \)
Conflict-driven clause learning (CDCL)

• Clause learning from conflicts
• Non-chronological backtracking
• Others
  – Lazy data structures
  – Branching heuristics
  – Restarts
  – Clause deletion
  – etc.
DPLL vs. CDCL

DPLL: no learning and chronological backtracking

CDCL: clause learning and non-chronological backtracking
Parallel SAT solvers

• Why?
  – Sequential solvers are difficult to improve
  – Can’t scale to large problems

• Category
  – Divide-and-conquer
  – Portfolio-based
Divide-and-conquer

- Divide search space into multiple independent sub-trees via guiding-paths

- Problem: load imbalance
Portfolio-based

• Observations
  – Modern SAT solvers are sensitive to parameters

• Principle
  – Run multiple CDCLs with different parameters simultaneously
  – Let them compete and cooperate

Youssef Hamadi and Lakhdar Sais. ManySAT: a parallel SAT solver. JSAT, 2009
Portfolio-based

- Diversification
  - Restart, variable heuristics, polarity, learning scheme
- Clause sharing

Youssef Hamadi and Lakhdar Sais. ManySAT: a parallel SAT solver. JSAT, 2009
Parallelization Barriers

• Poor scalability
  – 3x faster on 32-cores

• Reasons
  – BCP is P-complete, hard to parallelize
  – Bottlenecks [AAAI’2013]
  – Load imbalance for divide & conquer
  – Diversity for portfolio-based
Bottlenecks in CDCL proofs

Figure 1: Number of clauses derived at each depth of a typical CDCL proof

BigSAT: Turning SAT (DP) into Big Data Analytics

• Big Data thinking:

  Big Data Solution

  (1) Large Dataset + (2) Simple Computation + System Design

• DPLL?

• Others?
DP

• Introduced by Davis and Putnam in 1960

• Resolution

\[(x \lor y) \land (x' \lor z)\]

\[
\frac{(y \lor z)}{}
\]

• Algorithm
  – Select a variable \(x\), and add all resolvents
  – Remove all clauses containing \(x\)
  – Continue until no variable left for resolution

Davis and Putnam, A computing procedure for quantification theory, JACM, 1960
Ordering: $x_2 > x_1 > x_3$
Ordering: $x_2 > x_1 > x_3$

Ordering: $x_2 > x_1 > x_3$
Ordering: $x_2 > x_1 > x_3$

BigSAT: Turning SAT (DP) into Big Data Analytics

• Big Data thinking:

  Big Data Solution

  (1) Large Dataset + (2) Simple Computation + System Design

• DP exhibits data parallelism

  (1) Large Num. of Clauses + (2) Simple Resolution + BigSAT
ZBDD-based resolution

• ZBDD clauses representation
  – Common prefix and suffix compression

• Multi-resolution on ZBDD
  – Resolution on a pair of sets of clauses

• Clause subsumption elimination
Ordering: x1>x2>x3>x4>x5

\[ P^+ = \left( x_1 + x_2' + x_3 + x_5 \right) \]
\[ \left( x_1 + x_2' + x_4 + x_5 \right) \]
\[ \left( x_1 + x_3 + x_4 + x_5 \right) \]

\[ P^- = \left( x_1' + x_2 + x_3' + x_4 \right) \]
\[ \left( x_1' + x_2 + x_3' + x_5' \right) \]
Ordering: $x_1 > x_2 > x_3 > x_4 > x_5$

$P^+$

$(x_1 + x_2' + x_3 + x_5)$
$(x_1 + x_2' + x_4 + x_5)$
$(x_1 + x_3 + x_4 + x_5)$

$P^-$

$(x_1' + x_2 + x_3' + x_4)$
$(x_1' + x_2 + x_3' + x_5')$
BigSAT-parallel

- Good scalability factor
- Incremental DP
BigSAT-distributed

• Bulk Synchronous Parallel DP
  – Do resolutions as soon as possible
  – Do resolutions on all buckets

• Load balancing
  – Skewed join on Spark

In progress
Conclusion

• “Big data” thinking to solve problems that do not appear to generate big data

• Two example problems
  – Interprocedural static analysis
  – SAT solving

• Future problems
  – Symbolic execution
  – Program synthesis
  – …