# SAT with Memory

Md Solimul Chowdhury

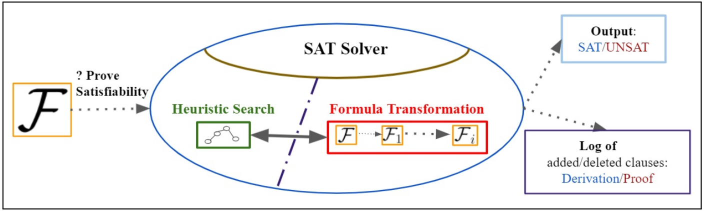
Automated Reasoning Group
Amazon Web Services

*chowmdso@amazon.com*

September/24/2021

# Boolean Satisfiability (SAT) Solving

- SAT solvers are well-known engines of proof generation
- SAT task ⟶ to **prove Satisfiability** of a Boolean formula $\mathcal{F}$
  - Finding an assignment to the variables such that $\mathcal{F}$ is *True* (SAT)
  - Otherwise, report Unsatisfiablity (UNSAT)

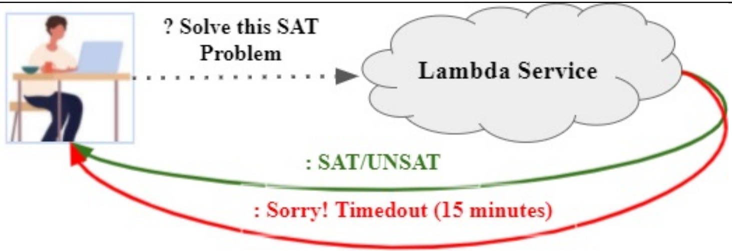


- important use cases in AWS
- e.g., verification of C programs via model checking

# SAT Meets the Cloud

- **Lambda Service**
  - serverless || program executables → **convenience**

**Use Case 1: SAT as a Lambda Service.**



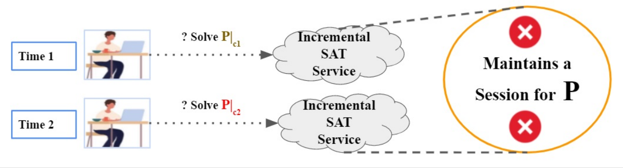
**Issue:** Solves 99% within time out. The other 1%?

- In cloud, we require **migration** of SAT jobs!

# SAT Meets the Cloud

- Incremental Solving is in many applications of SAT
  - given a SAT problem **P** and a set of constraints $\{C_1, \dots C_n\}$,
    - $n$ incremental calls: $\texttt{solve}(\mathbf{P})|_{C_1} \dots \texttt{solve}(\mathbf{P})|_{C_n}$
  - For an increment, no need to solve **P** from the scratch
- Example: Reachable paths to a node A from X
  - $\texttt{solve}(\text{X,A})|_{\texttt{C}_1} \dots \texttt{solve}(\text{X,A})|_{\texttt{C}_n}$

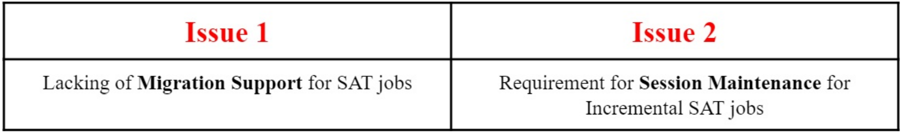
**Use Case 2: Incremental SAT queries with sessions**


Time 1
? Solve $\mathbf{P}|_{c1}$
Incremental SAT Service
Time 2
? Solve $\mathbf{P}|_{c2}$
Incremental SAT Service
Maintains a Session for **P**

**Issue:** Maintains a session for a SAT problem **P**.

- In cloud, we prefer **session free** incremental SAT solving!

# Our Solution : Saving and Reusing SAT States

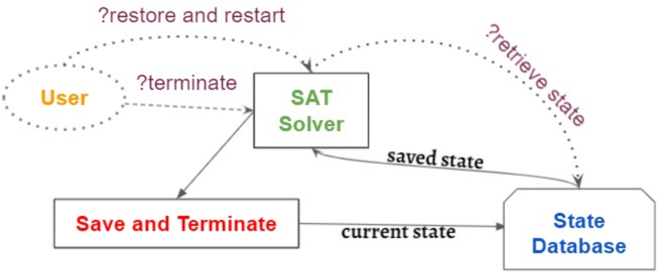


| Issue 1 | Issue 2 |
|---|---|
| Lacking of **Migration Support** for SAT jobs | Requirement for **Session Maintenance** for Incremental SAT jobs |

- To solve these issues, we propose
  - to save states at the end of a SAT call given a timeout
  - and reuse them in the next call
- Our solution,
  - preserve provability,
  - has low overhead, and
  - has a low storage requirement

# The Concept

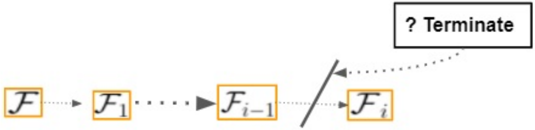
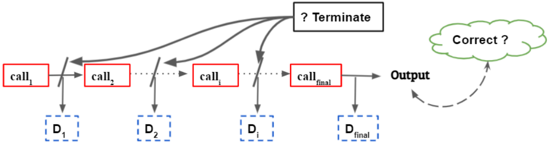
- Stop (at request), Save State, Solve (at request)



- We propose SAT$_{\text{mem}}$, *SAT with Memory*

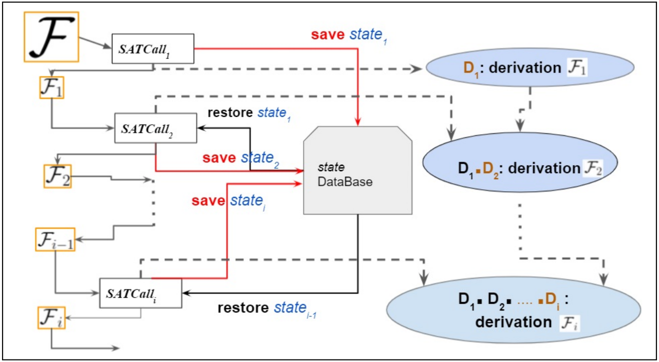
eifjccikedgrdlguklhjknntudeghrkkrburtfttdntf
eifjccikedgrcffdhjbeijjcdkvgrbknkbltvjgvgflh

# Design Options for $\text{SAT}_{\text{mem}}$

| | |
|---|---|
| **1. Transformed Formulas** |   ? Terminate<br>$\mathcal{F}$<br>$\mathcal{F}_1$<br>$\mathcal{F}_{i-1}$<br>$\mathcal{F}_i$ |
| **2. State Features** | |
| **3. Provability** |   ? Terminate<br>Correct ?<br>$\text{call}_1$<br>$\text{call}_2$<br>$\text{call}_i$<br>$\text{call}_{\text{final}}$<br>Output<br>$D_1$<br>$D_2$<br>$D_i$<br>$D_{\text{final}}$ |

# The SAT$_{\text{mem}}$ Architecture

| Save and Reuse | Derivation Logging |
|---|---|
| A. States<br>B. Transformed Input Formula<br>- Solver changes $\mathcal{F}$ | A. Log Derivations by concatenation.<br>B. Verification of combined derivation.<br>- Debugging.<br>- UNSAT verification.<br>- Satisfiability Assignment Extraction for $\mathcal{F}$. |

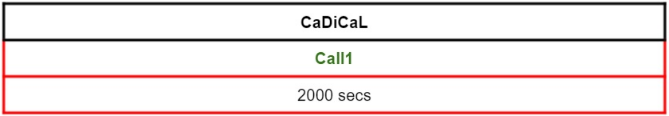


# Implementation and Benchmarks

- **Implementation:**
  SAT$_{\text{mem}}$ on top of CaDiCaL (2021)
  - Top performing SAT solver + supports incremental solving
- CaDiCaL → CaDiCaL$_{\text{mem}}$
  - Number of calls CaDiCaL
  - each call feeds itself state saved by the previous call
- **Benchmarks:** 297 instances (SR19) from SAT Race-2019
  - CaDiCaL runs more than 300 seconds for each of these instances
  - In CaDiCaL$_{\text{mem}}$, we use 300 secs as timeout threshold

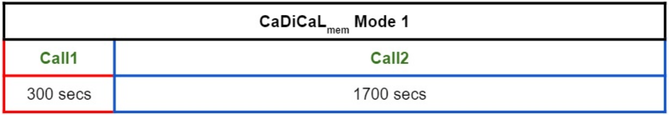
# Experimental Setup

- CaDiCaL: 297 SR19 instances with a timeout of 2000 seconds

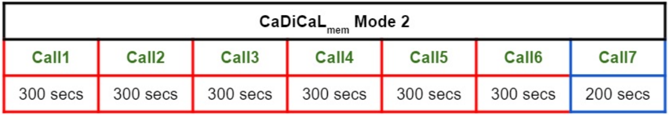


| CaDiCaL |
|---|
| Call1 |
| 2000 secs |

- CaDiCaL$_{mem}$: Same 297 instances, but two different modes of termination



| CaDiCaL$_{mem}$ Mode 1 | |
|---|---|
| Call1 | Call2 |
| 300 secs | 1700 secs |



| CaDiCaL$_{mem}$ Mode 2 | | | | | | |
|---|---|---|---|---|---|---|
| Call1 | Call2 | Call3 | Call4 | Call5 | Call6 | Call7 |
| 300 secs | 300 secs | 300 secs | 300 secs | 300 secs | 300 secs | 200 secs |

# Experimental Setup

Which State Features?

- At termination, the current call saves the following state features:
  1. Transformed Formula
  2. Internal to External Map / External to Internal Map
  3. Heuristic score of variables
  4. Phases
  5. Learned Clauses
- Details will be available in: `https://w.amazon.com/bin/view/ARG/ProofPlatformsTeam/SAT_with_Memory`

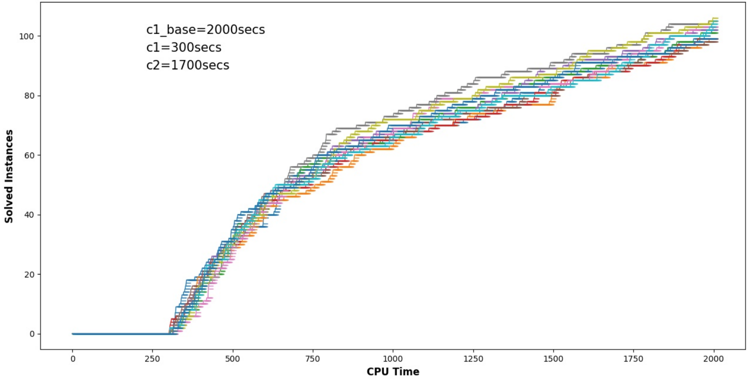
# Run-time Variation with Termination by Wall Clock

- In two different runs, at termination, a **call** may end into two different states This induces
  - non-determinism in the **next call**
  - and runtime variation
- For experiments, we needed to perform multiple runs to estimate the average case ...

# CaDiCaL$_{\text{mem}}$ Mode 1 results

| CaDiCaL$_{\text{mem}}$ Mode 1 | |
|---|---|
| Call1 | Call2 |
| 300 secs | 1700 secs |

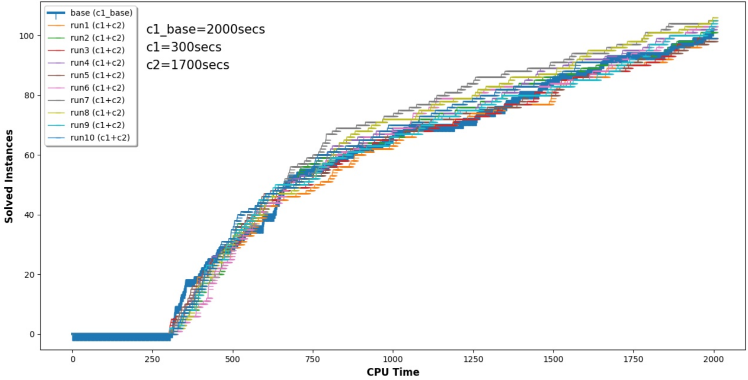
- Compared 10 runs of CaDiCaL$_{\text{mem}}$ (Mode 1) with CaDiCaL (base)
  - Which line is the base? Any guess?



# CaDiCaL$_{\text{mem}}$ Mode 1 results : Low overhead

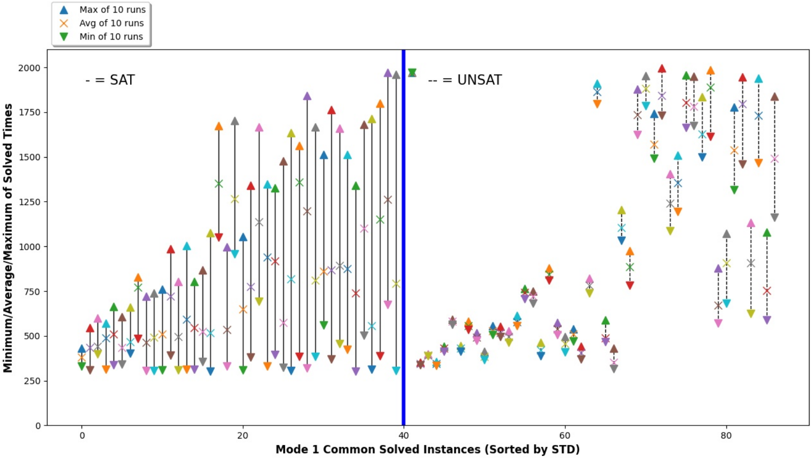
- Compared 10 runs of CaDiCaL$_{\text{mem}}$ (Mode 1) with CaDiCaL (base)

| CaDiCaL$_{\text{mem}}$ Mode 1 | |
|---|---|
| Call1 | Call2 |
| 300 secs | 1700 secs |



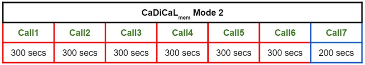
# Larger Runtime Variation with SAT instances (Mode 1)

- Statistics on Common solved instances (86) over 10 runs

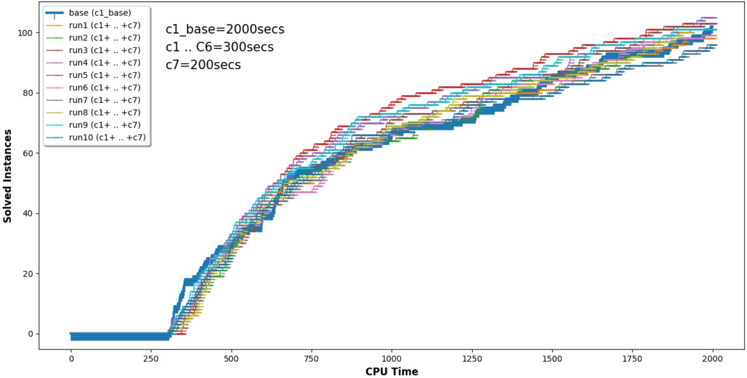


# CaDiCaL$_{mem}$ Mode 2 results: Low overhead

- Compared 10 runs of CaDiCaL$_{mem}$ (Mode 2) with CaDiCaL (base)



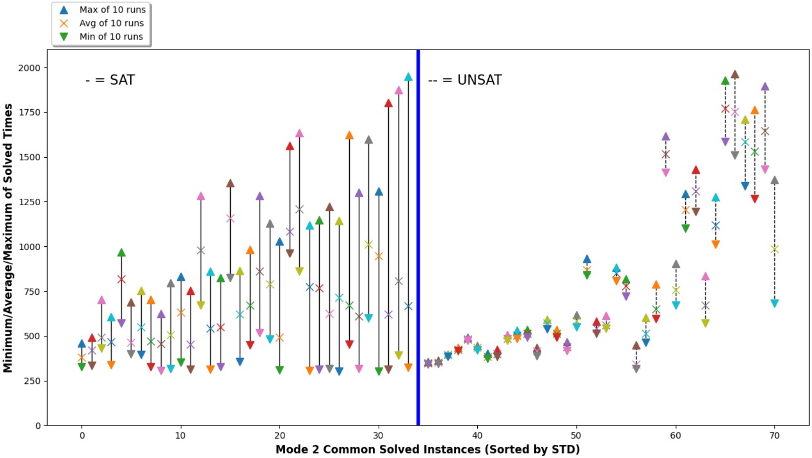
| CaDiCaL$_{mem}$ Mode 2 | | | | | | |
|---|---|---|---|---|---|---|
| Call1 | Call2 | Call3 | Call4 | Call5 | Call6 | Call7 |
| 300 secs | 300 secs | 300 secs | 300 secs | 300 secs | 300 secs | 200 secs |



c1_base=2000secs
c1 .. C6=300secs
c7=200secs

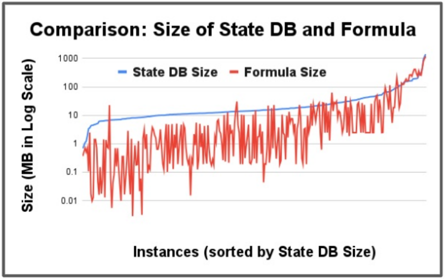
# Larger Runtime Variation with SAT Instances (Mode 2)

- Statistics on Common solved instances (70) over 10 runs.

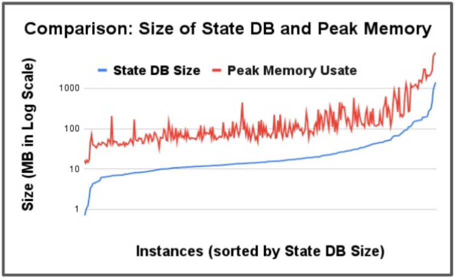


# Low Storage Requirement with CaDiCaL$_{mem}$

- Single run of Mode 1
- Peak Memory with Call 1
- State DB in text file (no compression)



Comparison: Size of State DB and Formula



Comparison: Size of State DB and Peak Memory

# Summary and Future Steps

- **Summary**
  - Many shot SAT solving with memory. For AWS
    - Migration of SAT Jobs
    - Session free Incremental SAT jobs
    - Restart SAT jobs
  - Small overhead
  - Light storage requirement
  - Proof producing $\longrightarrow$ verifiable
- **Whats Next?**
  - Experiments with few more sets of SAT competition benchmarks
  - Experiments focusing the runtime variations
    - Exploitable in parallel settings?
  - Experiments with incremental case
  - Constructions of Transformed formula and Learned Clauses from derivation
    - Solver independent implementation

# Acknowledgement

- Marijn Heule (Mentor)
- Mike Whalen (Manager)
- Frankie Botero