VECoS 2020

Multi-path Coverage of all Final States for Model-Based Testing Theory using Spark In-memory Design


Wilfried Yves Hamilton


Moez Krichen


Tarik Nahhal


Abdeltif Elbyed

Keywords: Model-Based Testing · Coverage · Big Data · Big Graphs · Apache Spark · Apache Hadoop · Parallel and Distributed Computing

14th International Conference on Verification and Evaluation of Computer and Communication Systems

26-27 October 2020

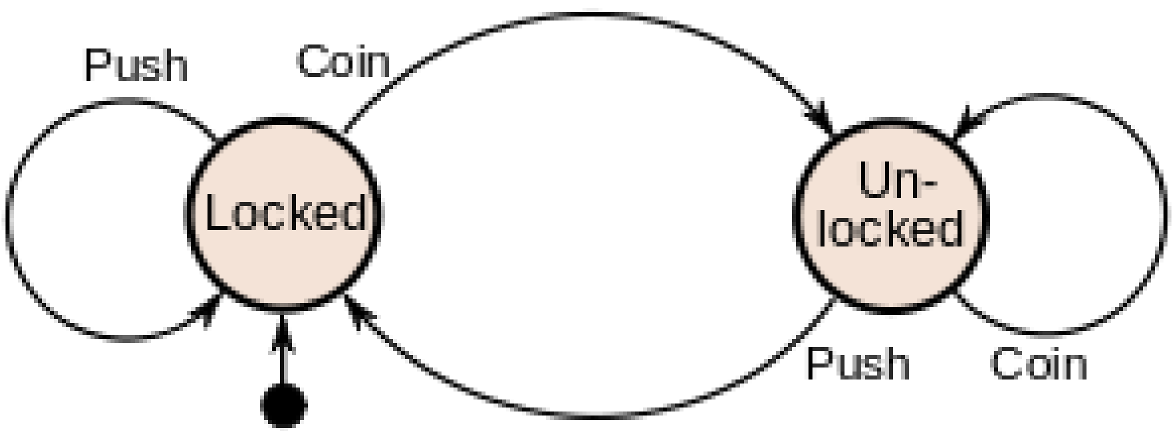
VECoS 2020

Topics

- Context
- Problem of coverage
- Proposed framework
- Experimental tests
- Conclusion and future directions

Finite State Machine (FSM)

- Overview
  - ❑ FSM is a mathematical model of computation.
  - ❑ It is an abstract machine that can be in exactly one of a finite number of states at any given time.
  - ❑ The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition



Some use cases of FSM

- Traffic light

- Elevator

- Vending machine

- Combination lock

Problem of coverage

Let $A = (Q, \sum, s_0, F, \delta)$ be a determinist finite automaton (DFA) such as

- Q is a finite set of states.
- δ is a finite set of inputs.
- $s_0 \in Q$ is the initial state.
- $F \subseteq Q$ is a set of final states.
- $\delta : Q \times \sum \rightarrow Q$ is the transition function.

Goal: from s_0, find all paths that cover the set of final states of F

Challenge: with state explosion, the search of all paths covering the final states of the system is very expensive.

Complexity: with NP-Hard problem

Related works

- **Adamatzky (1996)**
  - ❑ Automata coverage based on all-shortest paths computation
- **S. Rafe et . (2013)**
  - ❑ State explosion problem
  - ❑ Explosion problem in model checking
  - ❑ However, it is expensive in the execution time
- **Bensalem et al. (2005)**
  - ❑ Distributed coverage automata
  - ❑ Based on automatic generation of observers
- **Moez et al. (2012)**
  - ❑ Distributed coverage framework
  - ❑ Adapted for timed automata

Motivations

- Automata with state explosion takes enough of time.
- The coverage of big automata can become computationally expensive
- Shortest path computation requires very costly hardwares to achieve the computation

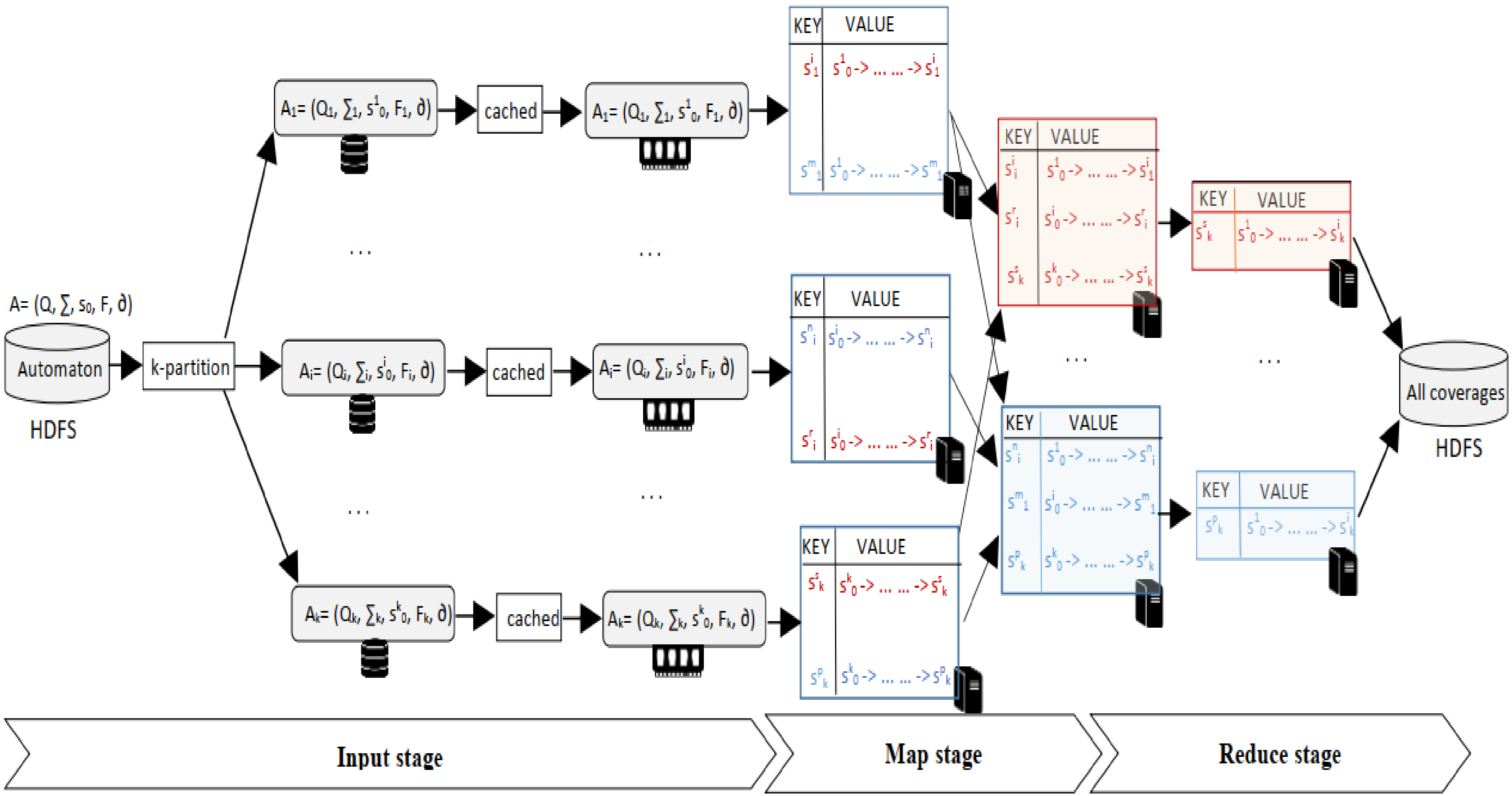
- **Large-scale automata: No works dealing with the case of state space explosion problem in model chechking.**

Fast distributed coverage approach based on Spark

- **Our coverage is builded on top of Spark**
  - ❑ Suited for automata with state explosion
  - ❑ Suited for dynamic automata
  - ❑ Use commodity hardware
  - ❑ Support Fault-tolerance

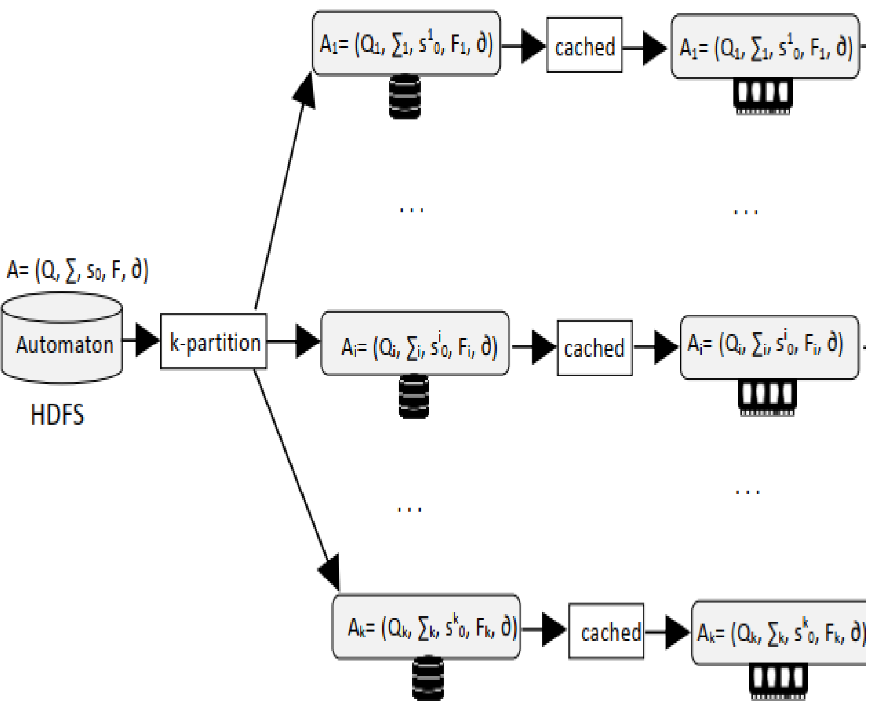
- **Spark-Coverage based approach**
  - ❑ Based on Spark in-memory design
  - ❑ Inspired by Adoni et al. (2018)

Spark-coverage overview



Coverage time $t_\sigma = t_{map} + t_{red}$

Input phase: automaton partition



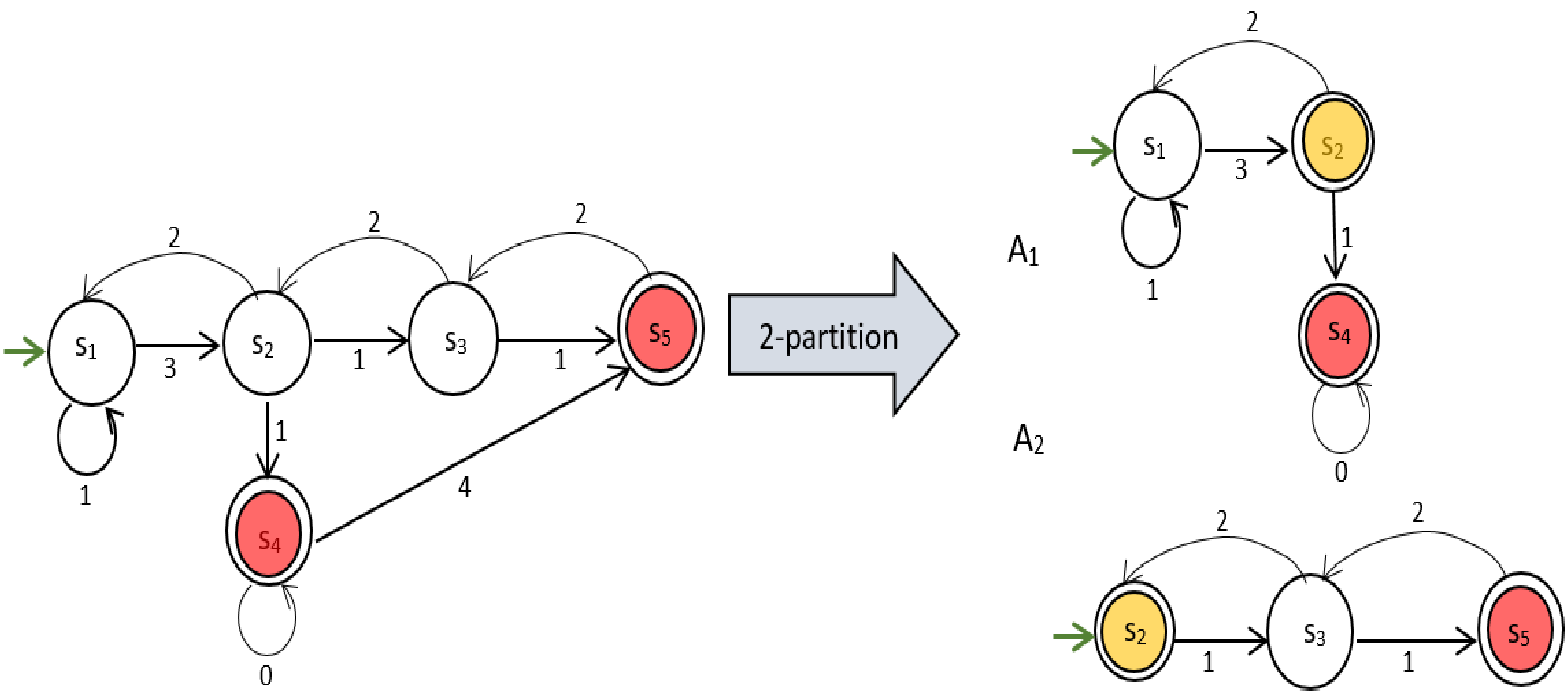
- **Edge-partition technique** **(Guerrieri et al., 2015)**
  - ❑ The automata A is partitioned under *n* sub-automaton.
  - ❑ A $=\{A_1, A_2, \dots, A_k\}$ such as *k is the* the number of nodes into the cluster.

$$k = N_{node} \times N_{core}$$

  - ❑ Each sub-automata A_i is assigned to each node i
  - ❑ Each sub-automata A_i is delimited by its boundary.
  - ❑ Frontier state x_i is used to communicate between two sub-automata.

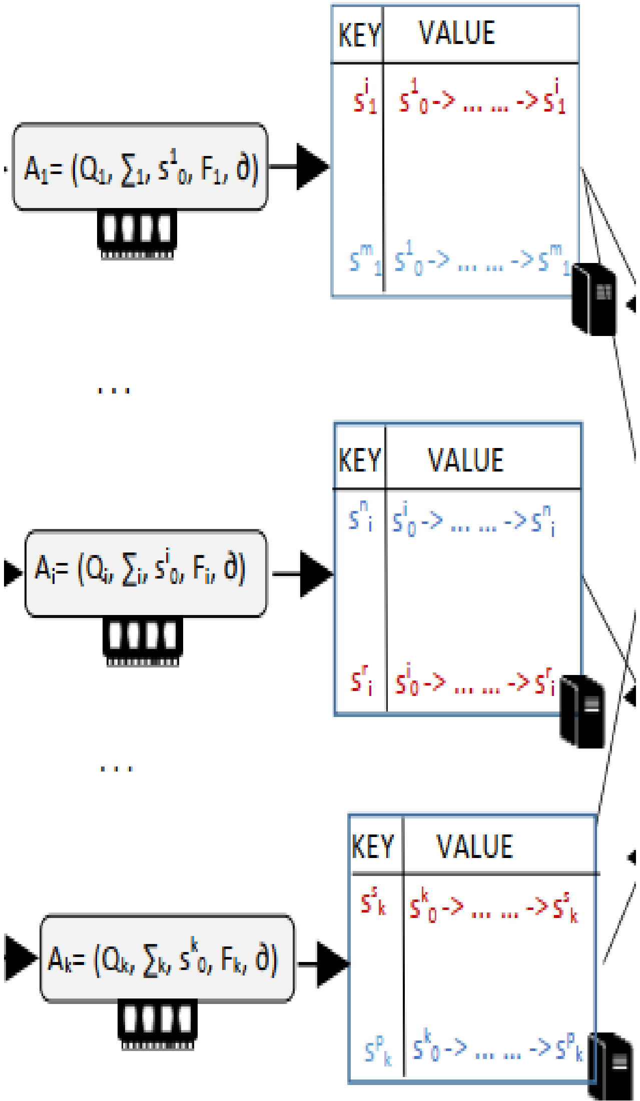
Input phase: automaton partition

- **Illustration of 2-partition**



VECoS 2020

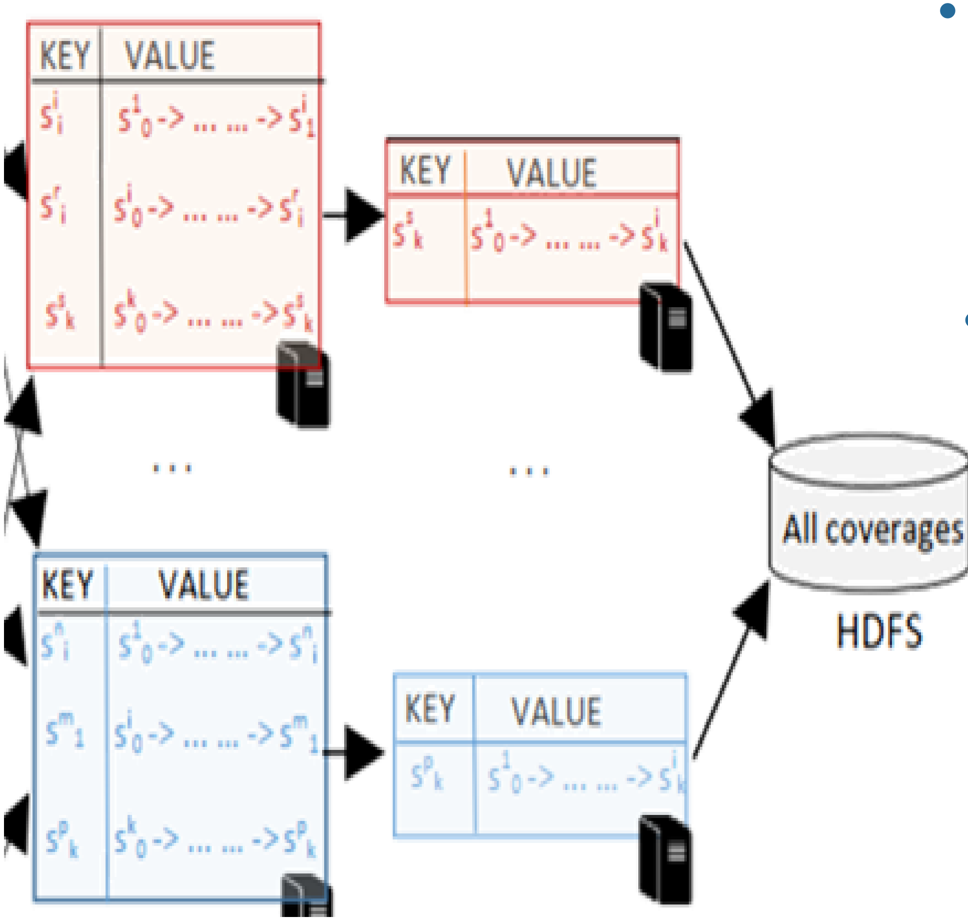
Map phase: intermediate states coverage



- **All coverage paths computating**
  - ❑ Each mapper is assigned to each sub-automaton
  - ❑ Run A* program on each sub-automaton
- **Map time**
  - ❑ The total time to complete all map tasks is calculated as follow:

$$t_{map} = \max\left(\sum_{i=1}^{N_{node}} \sum_{j=1}^{N_{map}} \sum_{k=1}^{|F_i|} \Delta m_{i,j,k} \right)$$

Reduce stage: merging all states coverage

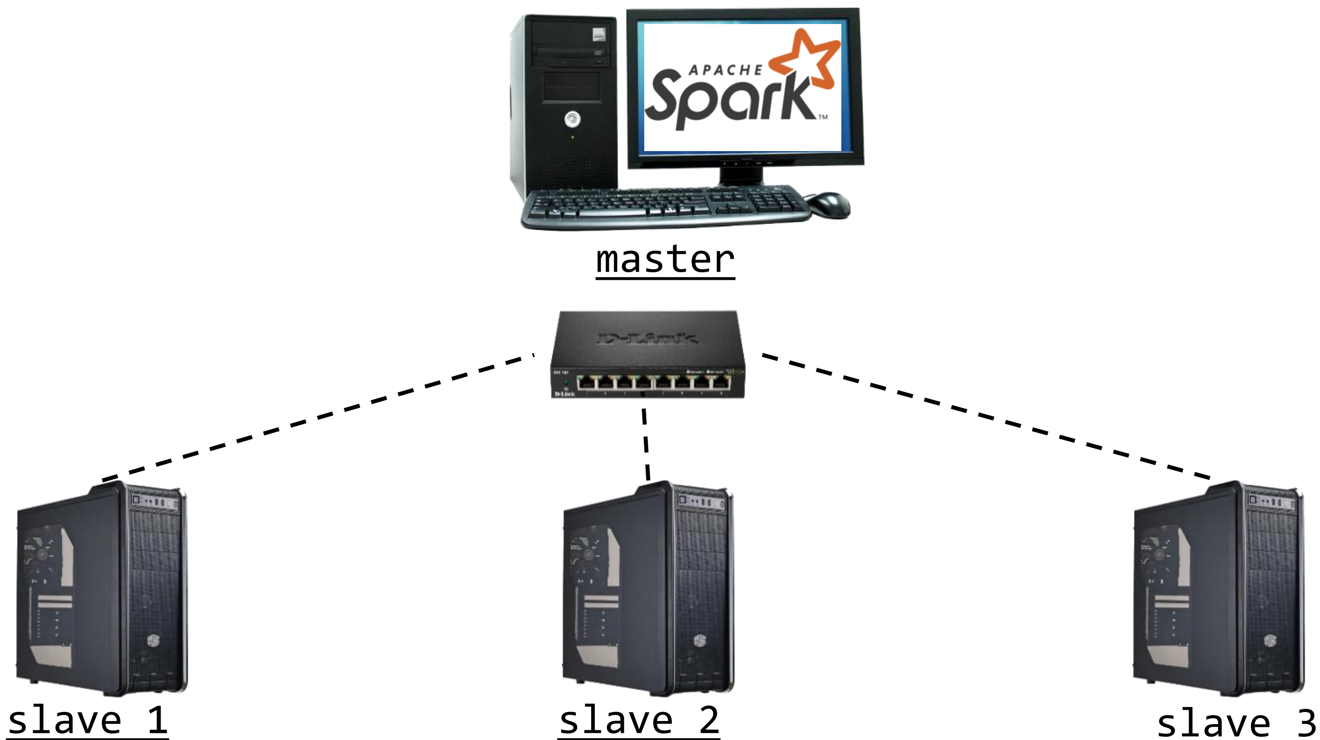


- **Aggregation of coverage paths**
  - ❑ Take mapper outputs
  - ❑ Merge all paths wich share same frontier states
- **Reduce time**
  - ❑ The total time to complete all reduce tasks is calculated as follow:

$$t_{red} = \max(\sum_{i=1}^{N_{node}} \sum_{j=1}^{N_{map}} \sum_{k=1}^{|F|} \Delta r_{i,j,k})$$

Experimental test

- **Cluster configuration**
  - ❑ Ram : 15GB
  - ❑ CPU : Intel Core i5-2410M @ 2.30 GHz
  - ❑ OS : Linux SUSE-3.0.101 32 bit
  - ❑ Spark version: 2.4.0



Experimental test

- **Dataset**

Automaton	States	Degree	Transitions	Type
ageRR	30,000	6	180,000	Directed
ageRRN	60,000	5	300,000	Directed
claroline	500,000	4	2,000,000	Directed
cpterminal	1,000,000	6	6,000,000	Directed
elsaRR	1,500,000	6	9,000,000	Directed

Experimental test

- **Time complexity : sequential vs distributed**

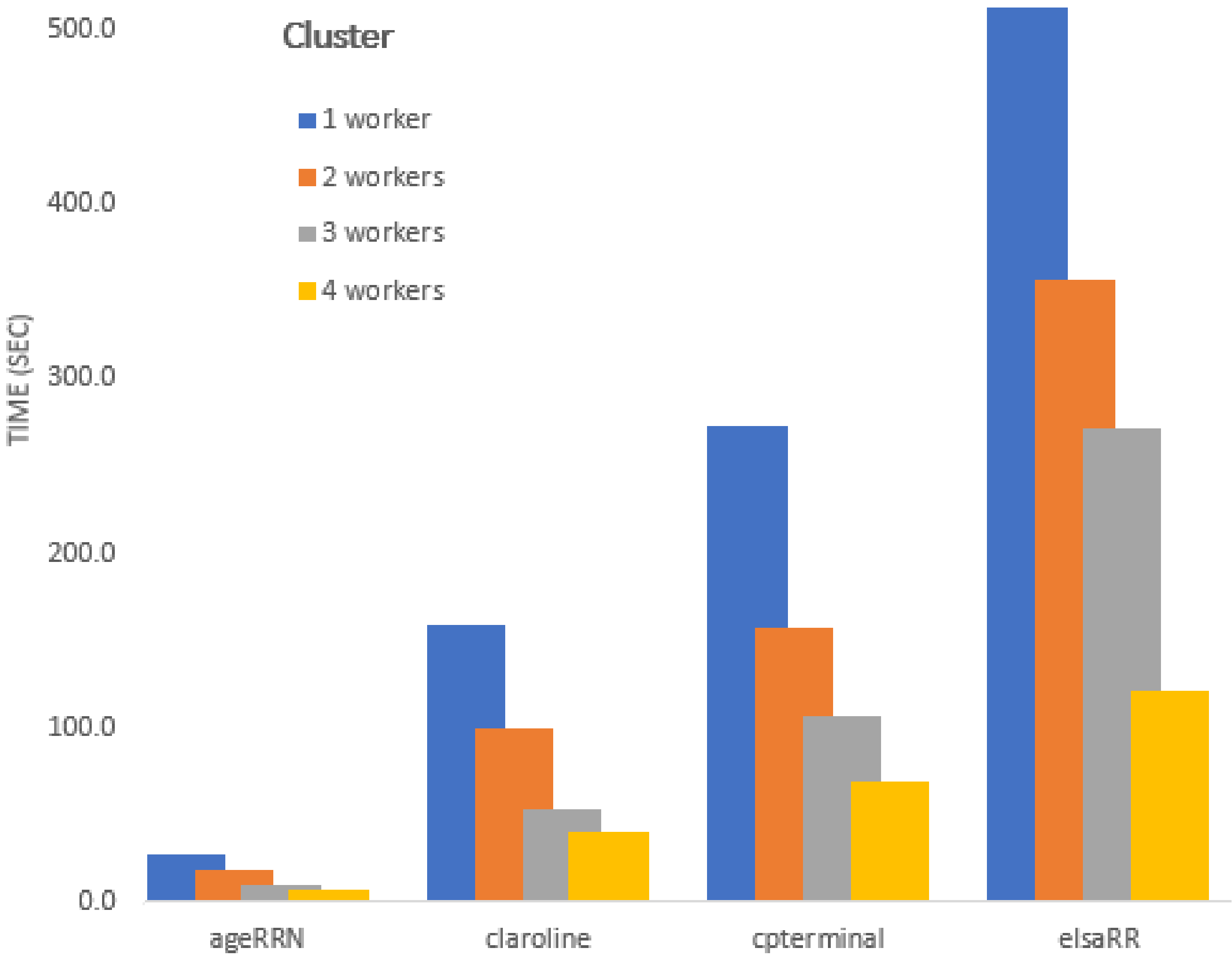
	Sequential	Spark on disk	Spark in-memory
ageRR	26 min	2 min	9.58 sec
ageRRN	1 h	4 min	26.85 sec
claroline	6.17 h	25 min	2.63 min
cpterminal	11.24 h	45.3 min	4.7 min
elsaRR	21 h	1h24 min	8.17 min

VECoS 2020

UNIVERSITÉ HASSAN II DE CASABLANCA
FACULTÉ DES SCIENCES AÏN CHOCK
FSAC
LIMSAD

Experimental test

- **Impact of number nodes on runtime**



Conclusion

- We have proposed a parallel and distributed framework for larges-cale automaton coverage.
- The time complexity decreases from exponential to linear time.
- The experimental results prove that our approach is faster and works well with very large automaton.
- But our approach presents some limitations:
  - Path optimality depends on the partitioning strategy and the number of sub-automatons
  - The computation is often memory expensive.

Futher works

- We are interested in studying the impact of automaton partitioning on the time complexity.
- Propose an extended version of the framework for the coverage of timed-automaton and distributed systems

FSAC
UNIVERSITÉ HASSAN II DE CASABLANCA
FACULTÉ DES SCIENCES AÏN CHOCK
LIMSAD

Thank you


adoniwilfried@gmail.com


moez.krichen@redcad.org


t.nahhal@fsac.ac.ma


a.elbyed@fsac.ac.ma