CONTENTS

Preface  R K Shyamasundar, Editor-in-Chief

Article R1  11 Pages  New Frontiers in Formal Methods: From Theory to Cyber-Physical Systems, Education and Beyond  Sanjit A. Seshia

Article R2  10 Pages  Static Analysis and Symbolic Code Execution  Dhiren Patel, Madhura Parikh and Reema Patel

Article R3  11 Pages  LFTL: A multi-threaded FTL for a Parallel IO Flash Card under Linux  Srimugunthan, K. Gopinath, Giridhar Appaji Nag Yasa

Article R4  10 Pages Towards An Executable Declarative Specification of Access Control  N. V. Narendra Kumar and R. K. Shyamasundar

Article R5  8 Pages  An Empirical Study of Popular Matrix Factorization based Collaborative Filtering Algorithms  Shamsuddin N. Ladha, Hari Manassery Koduvely and Lokendra Shastri
# Editorial Board

**Editor-in-Chief**

Prof. R K Shyamasundar, Tata Institute of Fundamental Research

**Editorial Board Members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution/University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof. Srinivasa Aluru</td>
<td>IIT Bombay &amp; Iowa State Univ.</td>
</tr>
<tr>
<td>Prof. Subhasis Banerjee</td>
<td>IIT Delhi</td>
</tr>
<tr>
<td>Dr. Anup Bhattacharjee</td>
<td>BARC</td>
</tr>
<tr>
<td>Prof. Pushpak Bhattacharya</td>
<td>IIT Bombay</td>
</tr>
<tr>
<td>Dr. Satish Chandra</td>
<td>IBM Research</td>
</tr>
<tr>
<td>Prof. P P Chakrabarti</td>
<td>IIT Kharagpur</td>
</tr>
<tr>
<td>Prof. B B Chaudhuri</td>
<td>ISI, Kolkata</td>
</tr>
<tr>
<td>Prof. Sajal Das</td>
<td>Univ. of Texas at Arlington</td>
</tr>
<tr>
<td>Dr. Sukhendu Das</td>
<td>IIT, Madras</td>
</tr>
<tr>
<td>Dr. Sudhir Dixit</td>
<td>HP Labs</td>
</tr>
<tr>
<td>Prof. Nikil Dutt</td>
<td>UC Irvine</td>
</tr>
<tr>
<td>Prof. Naveen Garg</td>
<td>IIT Delhi</td>
</tr>
<tr>
<td>Dr. Rahul Garg</td>
<td>Opera Solutions, Noida</td>
</tr>
<tr>
<td>Prof. Manoj Gaur</td>
<td>NIT, Jaipur</td>
</tr>
<tr>
<td>Prof. Rajesh Gupta</td>
<td>UCSD, La Jolla</td>
</tr>
<tr>
<td>Prof. J Haritsa</td>
<td>IISc</td>
</tr>
<tr>
<td>Prof. D Janakiram</td>
<td>IIT Madras</td>
</tr>
<tr>
<td>Dr. Ravi Kannan</td>
<td>Microsoft Research</td>
</tr>
<tr>
<td>Dr. Raghu Krishnapuram</td>
<td>IBM Research</td>
</tr>
<tr>
<td>Prof. Vipin Kumar</td>
<td>Univ of Minnesota</td>
</tr>
<tr>
<td>Prof. Rupak Majumdar</td>
<td>MPI Software Systems</td>
</tr>
<tr>
<td>Prof. Bhuvaneswar Mishra</td>
<td>Courant Institute of Mathematical Sciences</td>
</tr>
<tr>
<td>Prof. Y Narahari</td>
<td>IISc</td>
</tr>
<tr>
<td>Prof. K Narayana Kumar</td>
<td>CMI</td>
</tr>
<tr>
<td>Dr. Srinivas Padmanabhuni</td>
<td>Infosys Labs</td>
</tr>
<tr>
<td>Prof. N R Pal</td>
<td>IIS, Kolkata</td>
</tr>
<tr>
<td>Prof. C Pandurangan</td>
<td>IIT Madras</td>
</tr>
<tr>
<td>Prof. Dhiren K. Patel</td>
<td>NIT, Surat</td>
</tr>
<tr>
<td>Prof. Susbil K Prasad</td>
<td>Georgia State Univ.</td>
</tr>
<tr>
<td>Prof. Arun K. Pujari</td>
<td>University of Hyderabad</td>
</tr>
<tr>
<td>Prof. Krithi Ramamritam</td>
<td>IIT Bombay</td>
</tr>
<tr>
<td>Prof. R Ramujiyam</td>
<td>IMSc</td>
</tr>
<tr>
<td>Dr. S Ramesh</td>
<td>GM, India Science Lab</td>
</tr>
<tr>
<td>Prof. Rajeev Sangal</td>
<td>IIT, Hyderabad</td>
</tr>
<tr>
<td>Prof. Abdul Sattar</td>
<td>Griffith University</td>
</tr>
<tr>
<td>Prof. Sanjit Seshia</td>
<td>UC Berkeley</td>
</tr>
<tr>
<td>Dr. Lokendra Shastri</td>
<td>Infosys Labs</td>
</tr>
<tr>
<td>Prof. Sandeep Shukla</td>
<td>Virginia Tech</td>
</tr>
<tr>
<td>Prof. T V Sreenivas</td>
<td>IISc</td>
</tr>
</tbody>
</table>

© Computer Society of India™ Copyright Protected.

## Online Advertisement

The Journal of Computing, published by the Computer Society of India offers good visibility of online research content on computer science theory, Languages & Systems, Databases, Internet Computing, Software Engineering and Applications. With the potential to deliver targeted audience regularly with each issue, The Journal of Computing is an excellent media for promotion of your IT Technology products, IT Technology services, Conferences, Technology Recruitments etc through advertisement. Additionally, the cost of advertisement in The Journal of Computing is substantially low.

The advertisement rates for The Journal of Computing is as follows:

### Printed Edition

<table>
<thead>
<tr>
<th>Inside Pages</th>
<th>Other Countries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>India</strong></td>
<td><strong>India</strong></td>
</tr>
<tr>
<td>No. of Insertions: 1</td>
<td>No. of Insertions: 1</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 10,000; in Colour: Rs 20,000</td>
<td>Full page in B&amp;W: $300; in Colour: $650</td>
</tr>
<tr>
<td>Half page in B&amp;W: Rs 6,000; in Colour: Rs 12,000</td>
<td>Half page in B&amp;W: $200; in Colour: $325</td>
</tr>
<tr>
<td>No. of Insertions: 6</td>
<td>No. of Insertions: 6</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 50,000; in Colour: Rs 100,000</td>
<td>Full page in B&amp;W: $1500; in Colour: $3000</td>
</tr>
<tr>
<td>Half page in B&amp;W: Rs 30,000; in Colour: Rs 60,000</td>
<td>Half page in B&amp;W: $1000; in Colour: $2000</td>
</tr>
<tr>
<td>No. of Insertions: 12</td>
<td>No. of Insertions: 12</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 10,000; in Colour: Rs 200,000</td>
<td>Full page in B&amp;W: $450; in Colour: $750</td>
</tr>
<tr>
<td>Half page in B&amp;W: Rs 60,000; in Colour: Rs 120,000</td>
<td>Full page in B&amp;W: $2250; in Colour: $3500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inside Cover Pages</th>
<th>Inside Cover Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Insertions: 1</td>
<td>No. of Insertions: 1</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 15,000; in Colour: Rs 25,000</td>
<td>Full page in B&amp;W: $450; in Colour: $750</td>
</tr>
<tr>
<td>No. of Insertions: 6</td>
<td>No. of Insertions: 6</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 75,000; in Colour: Rs 125,000</td>
<td>Full page in B&amp;W: $2250; in Colour: $3500</td>
</tr>
<tr>
<td>No. of Insertions: 12</td>
<td>No. of Insertions: 12</td>
</tr>
<tr>
<td>Full page in B&amp;W: Rs 150,000; in Colour: Rs 250,000</td>
<td>Full page in B&amp;W: $2250; in Colour: $3500</td>
</tr>
</tbody>
</table>
Contents

<table>
<thead>
<tr>
<th>Article</th>
<th>Title</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>New Frontiers in Formal Methods: From Theory to Cyber-Physical Systems, Education and Beyond</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Sanjit A. Seshia</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>Static Analysis and Symbolic Code Execution</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Dhiren Patel, Madhura Parikh and Reema Patel</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>LFTL: A multi-threaded FTL for a Parallel IO Flash Card under Linux</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Srimugunthan, K. Gopinath, Giridhar Appaji Nag Yasa</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>Towards An Executable Declarative Specification of Access Control</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>N. V. Narendra Kumar and R. K. Shyamasundar</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>A Novel, Decentralized, Local Information based Algorithm for Community Detection in Social Networks</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Ramasuri Narayanam &amp; Y. Narahari</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>Mitigation of Jamming Attack in Wireless Sensor Networks</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Manju. V.C., Dr. Sasi Kumar</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Shyamli Rao &amp; Amit Joshi</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Abinaya Mahendiran, Srivatsan Sridharan, Sushanth Bhat and Shrisha Rao</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Shyamli Rao &amp; Amit Joshi</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Abinaya Mahendiran, Srivatsan Sridharan, Sushanth Bhat and Shrisha Rao</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Shyamli Rao &amp; Amit Joshi</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Abinaya Mahendiran, Srivatsan Sridharan, Sushanth Bhat and Shrisha Rao</td>
<td></td>
</tr>
</tbody>
</table>
Preface

This is the last issue of Volume 2. The first paper entitled New Frontiers in Formal Methods: From Theory to Cyber-Physical Systems, Education, and Beyond, is an invited paper by Prof. Sanjit Seshia, University of California, Berkeley, USA, highlights the challenges of Formal Methods, and Machine Learning and their application to Cyber-Physical Systems. I thank the author for sparing his valuable time and I am certain the article will be of immense help to our readers. The rest of the articles are regular papers. It is a pleasure to thank the authors for their contributions.

This is the last issue, I am editing for the Journal. I hereby place on record my appreciation for the International Editorial Board and the referees who have supported the refereeing of the papers submitted for the Journal and also provided constructive feedbacks to the authors. The success of any journal depends on contributions from authors. I am happy to note that there is a constant stream of papers submitted for favour of publication. I thank the authors and wish the Journal to attract more papers and establish its impact in the ICT arena. Finally but not least, I thank the ExecCom for the continued support for the Journal.

Prof. R K Shyamasundar
IIT Bombay
Editor-in-Chief, CSI Journal of Computing
May 2015
New Frontiers in Formal Methods: Learning, Cyber-Physical Systems, Education, and Beyond

Sanjit A. Seshia

S. A. Seshia, EECS Department, UC Berkeley, Berkeley, CA 94720-1770 E-mail: sseshia@eecs.berkeley.edu

We survey promising directions for research in the area of formal methods and its applications, including fundamental questions about the combination of formal methods and machine learning, and applications to cyberphysical systems and education.

Index Terms: Formal Methods, Machine Learning, Cyber-Physical Systems, Education.

1. Introduction

Formal methods is a field of computer science and engineering concerned with the rigorous mathematical specification, design, and verification of systems [1], [2]. At its core, formal methods is about proof: formulating specifications that form proof obligations, designing systems to meet those obligations, and verifying, through a verification technique, that the systems indeed meet their specifications. Over the past few decades, formal methods has made enormous strides. Verification techniques such as model checking [3], [4], [5] and theorem proving (see, e.g., [6], [7], [8]) are used routinely in the computer-aided design of integrated circuits and have been widely applied to find bugs in software, analyze embedded systems, and find security vulnerabilities. At the heart of these advances are computational proof engines such as Boolean satisfiability solvers [9], Boolean reasoning and manipulation routines based on Binary Decision Diagrams (BDDs) [10], and satisfiability modulo theories (SMT) solvers [11].

Even as these advances extend the capacity and applicability of formal methods, several technical challenges remain, while exciting new application domains are yet to be explored. This article expresses the author’s personal viewpoint on the new frontiers for formal methods, highlighting three areas for further research and development. First, recent results have highlighted interesting conceptual connections between formal methods and the field of machine learning (inductive learning from examples), which merits further study. Second, the field of cyber-physical systems – computational systems tightly integrated with physical processes – is growing rapidly, throwing up many problems that can be addressed with formal methods. Finally, the field of education is undergoing a sea change, with the advent of massive open online courses (MOOCs) and related technologies. Formal methods can play a significant role in improving the state of the art in technologies for education. In the rest of the paper, we examine each of these frontiers for formal methods in somewhat more depth.

2. Formal Methods and Inductive Learning

A major challenge for formal methods, in the author’s opinion, is to develop effective proof techniques that can capture the way the best mathematicians and scientists generalize from intuition and experience to synthesize the critical parts of a proof. A major opportunity for formal methods is to blend techniques from inductive (machine) learning into traditional deductive procedures to mimic such a process of generalization. In this section, we present a brief exposition of these positions. For a more in-depth treatment, the reader is referred to a prior article by the author [12], [13].

We begin by examining the traditional view of verification as a decision problem, with three inputs (see Fig. 1):

1. A model of the system to be verified, S;
2. A model of the environment, E, and
3. The property to be verified, Φ.

The verifier generates as output a YES/NO answer, indicating whether or not S satisfies the property Φ in environment E. Typically, a NO output is accompanied by a counterexample, also called an error trace, which is an execution of the system that indicates how Φ is violated. Other debugging output may also be provided. Some formal verification tools also include a proof or certificate of correctness with a YES answer, so that an independent system can use the proof to check (usually more quickly) that the property indeed holds.

![Fig. 1: Formal verification procedure.](image-url)
This traditional view is somewhat high-level and idealized. In practice, one does not always start with models $S$ and $E$—these might have to be abstracted from implementations or hypothesized manually. Also, the specification $\Phi$ is rarely complete and sometimes inconsistent, as has often been noted in industrial practice. Finally, the figure omits certain inputs that are perhaps the most crucial in successfully completing the verification task. For example, if performing a proof by mathematical induction, one might need to supply hints to the verifier in the form of auxiliary inductive invariants. Similarly, while performing a proof by abstraction, one may need to pick the right abstract domain and refinement strategy for generating suitable abstractions. Generating these auxiliary inputs to the proof procedure are often the trickiest steps in getting verification to succeed, as they require insight not only into the problem but also into the inner workings of the proof procedure.

The issues raised in the preceding paragraph can be distilled into a single challenge: the effective synthesis of three kinds of verification artifacts: models, specifications, and auxiliary inputs. There are several examples of these verification artifacts, including inductive invariants, abstractions, environment assumptions, input constraints, auxiliary lemmas, ranking functions, and interpolants, amongst others. In Sec. 2.1, we illustrate this formulation of verification in terms of synthesis with two examples. As we will see, one often needs human insight into at least the form of these artifacts, if not the artifacts themselves, to succeed in verifying the design. In Sec. 2.2, we describe an approach to the synthesis of these artifacts by a combination of inductive learning, deductive reasoning, and human insight in the form of a “structure hypothesis.”

2.1 Verification by Reduction to Synthesis

We consider here a common verification problem: proving that a certain property is an invariant of a system—i.e., that it holds in all states of that system. Let us first set up some notation. Relevant background material may be found in a recent book chapter on modeling for verification [14].

Let $M = (I, \delta)$ be a transition system where $I$ is a logical formula encoding the set of initial states, and $\delta$ is a formula representing the transition relation. For simplicity, assume that $M$ is finite-state, so that $I$ and $\delta$ are Boolean formulas. Suppose we want to verify that $M$ satisfies a temporal logic property $\Psi = G \psi$ where $\psi$ is a logical formula involving no temporal operators. We now consider two methods to perform such verification.

2.1.1 Invariant Inference

Consider first an approach to prove this property by (mathematical) induction. In this case, we seek to prove the validity of the following two logical statements:

$$I(s) \implies \psi(s) \quad (1)$$

$$\psi(s) \land \delta(s, s') \implies \psi(s') \quad (2)$$

where, in the usual way, $\psi(s)$ denotes that the logical formula $\psi$ is expressed over variables encoding a state $s$.

Usually, when one attempts a proof by induction as above, one fails to prove the validity of the second statement, Formula 2. This failure is rarely due to any limitation in the underlying validity checkers for Formula 2. Instead, it is usually because the hypothesized invariant $\psi$ is “not strong enough.” More precisely, $\psi$ needs to be conjoined (strengthened) with another formula, known as the auxiliary inductive invariant.

More precisely, the problem of verifying whether system satisfies an invariant property reduces to the problem of synthesizing an auxiliary invariant $\phi$ such that the following two formulas are valid:

$$I(s) \implies \phi(s) \land \psi(s) \quad (3)$$

$$\phi(s) \land \psi(s) \land \delta(s, s') \implies \phi(s') \land \psi(s') \quad (4)$$

If no such $\phi$ exists, then it means that the property $\psi$ is not an invariant of $M$, since at a minimum a $\phi$ characterizing all reachable states of $M$ should satisfy Formulas 3 and 4 above.

2.1.2 Abstraction-based Model Checking

Another common approach to solving the invariant verification problem is based on sound and complete abstraction. Given the original system $M$, one seeks to compute an abstract transition system $\alpha(M) = (I', \delta')$ such that $\alpha(M)$ satisfies $\Psi$ if and only if $M$ satisfies $\Psi$. This approach is computationally advantageous when the process of computing $\alpha(M)$ and then verifying whether it satisfies $\Psi$ is significantly more efficient than the process of directly verifying $M$ in the first place. We do not seek to describe in detail what abstractions are used, or how they are computed. The only point we emphasize here is that the process of computing the abstraction is a synthesis task.

In other words, instead of directly verifying whether $M$ satisfies $\Psi$, we seek to synthesize an abstraction function $\alpha$ such that $\alpha(M)$ satisfies $\Psi$ if and only if $M$ satisfies $\Psi$, and then we verify whether $\alpha(M)$ satisfies $\Psi$.

2.1.3 Reduction to Synthesis

Given the two examples above, let us now step back and formalize the general notion of performing verification by reduction to synthesis.

Suppose the original verification problem is:

Given a system $M$ (composition of system and environment models), does $M$ satisfy a specification $\Psi$?

The approach in the two examples above is to reduce this question to the form below:

Given a specification $\Omega$ and a class of formal artifacts $\Sigma$, does there exist an element of $\Sigma$ that satisfies $\Omega$?

To make things concrete, we instantiate the symbols above in the two examples.

Invariant inference. In this case, $\Psi = G \psi$, is the set of all Boolean formulae over the (propositional) state variables of $M$. $\Omega$ comprises Formulas 3 and 4 above.
Abstraction-based verification. In this case, again, $\Psi = G \psi . \Sigma$ is the set of all abstraction functions $\alpha$ corresponding to a particular abstract domain [15]; for example, all possible localization abstractions [16]. $\Omega$ is the statement $\alpha(M)$ satisfies $\Psi$ if and only if $M$ satisfies $\Psi'$.

Note that this is a reduction in the complexity and computability-theoretic sense: the verification problem has a solution if and only if the synthesis problem has one. The synthesis problem is not any easier to solve, in the theoretical sense, than the original verification problem. However, the synthesis version of the problem may be easier to solve in practice, especially if suitable additional constraints are imposed. We elaborate on solution techniques in the following section.

### 2.2 Integrating Induction, Deduction, and Structure

We have so far discussed how verification can be performed by reduction to synthesis. An effective paradigm for synthesis in recent times has been to combine induction, deduction, and structure hypotheses; an approach the author previously termed sciduction and formalized [12]. In this section, we present a brief introduction to some of the key ideas in this approach.

Induction is the process of inferring a general law or principle from observation of particular instances.\(^1\) Machine learning algorithms are typically inductive, generalizing from (labeled) examples to obtain a learned concept or classifier [17], [18]. Deduction, on the other hand, involves the use of general rules and axioms to infer conclusions about particular problem instances. Traditional formal verification and synthesis techniques, such as model checking or theorem proving, are deductive. One may wonder whether inductive reasoning may seem out of place here, since typically an inductive argument only ensures that the truth of its premises make it only likely or probable that its conclusion is also true. However, one observes that humans often employ a combination of inductive and deductive reasoning while performing verification or synthesis. For example, while proving a theorem, one often starts by working out examples and trying to find a pattern in the properties satisfied by those examples. The latter step is a process of inductive generalization. These patterns might take the form of lemmas or background facts that then guide a deductive process of proving the statement of the theorem from known facts and previously established theorems (rules). The process usually iterates between inductive and deductive reasoning until the final result is obtained.

Sciduction [13], [12] is a formalization of such a combination of inductive and deductive reasoning.\(^2\) The key in integrating induction and deduction is the use of structure hypotheses, mathematical hypotheses used to define the class of artifacts to be synthesized within the overall verification or synthesis problem. Sciduction constrains inductive and deductive reasoning using structure hypotheses, and actively combines inductive and deductive reasoning: for instance, deductive techniques generate examples for learning, and inductive reasoning is used to guide the deductive engines. It is beyond the scope of this paper to explain the concept of sciduction in detail. However, we provide below an intuitive explanation illustrated by counterexample-guided abstraction refinement (CEGAR) [19], a very effective approach to model checking.

#### 2.2.1 Sciduction and Counterexample-Guided Inductive Synthesis

In sciduction, one combines an inductive inference procedure $\mathcal{L}$, a deductive engine $\mathcal{D}$, and a structure hypothesis $\mathcal{H}$ to obtain a synthesis algorithm. The structure hypothesis syntactically constrains the space of artifacts being searched. This constraint is also referred to as syntax guidance [20].

In machine learning, such a restricted space is called the concept class, and each element of that space is often called a candidate concept. A common sciductive approach is to formulate the overall problem as one of active learning using a query-based model. Active learning is a special case of machine learning in which the learning algorithm can control the selection of examples that it generalizes from and can query one or more oracles to obtain both examples as well as labels for those examples. (Typically the labels are positive or negative.) We refer the reader to a paper by Angluin [21] for an overview of various models for query-based active learning.

The query oracles are often implemented using deductive procedures such as model checkers or satisfiability solvers. Thus, the overall synthesis algorithm usually comprises a top-level inductive learning algorithm constrained by the structure hypothesis that invokes deductive procedures (query oracles). When the top-level strategy is inductive, we typically refer to the approach simply as “inductive synthesis” (even when deductive procedures are used). There are two important choices one must make to fix an inductive synthesis algorithm: (1) search strategy: How should one search the concept class? and (2) example selection strategy: Which examples do we learn from?

Counterexample-guided inductive synthesis (CEGIS) [22] shown in Figure 2 is perhaps the most popular approach to inductive synthesis today. The defining aspect of CEGIS is its example selection strategy: learning from counterexamples provided by a verification oracle. The learning algorithm, which is initialized with a particular choice of concept class and possibly with an initial set of (positive) examples, proceeds by searching the space of candidate concepts for one that is consistent with the examples seen so far. There may be several such consistent concepts, and the search strategy determines the chosen candidate. This is then presented to the verification oracle, which checks the candidate against the correctness specification. If the candidate is correct, the synthesizer terminates and outputs this candidate.

---

1. The term “induction” is often used in the verification community to refer to mathematical induction, which is actually a deductive proof rule. Here we are employing “induction” in its more classic usage arising from the field of Philosophy.
2. sciduction stands for structure-constrained induction and deduction.
Otherwise, the verification oracle returns a counterexample to the learning algorithm, which adds the counterexample to its set of examples and repeats its search. It is possible that, after some number of iterations of this loop, the learning algorithm may be unable to find a candidate concept consistent with its current set of (positive/negative) examples, in which case the learning step, and hence the overall CEGIS procedure, fails.

![Diagram of CEGIS process](image)

**Fig. 2:** Counterexample-Guided Inductive Synthesis (CEGIS)

Several search strategies are possible, and the choice depends on the application domain; see [20] for a more detailed discussion.

### 2.2.2 Counterexample-Guided Abstraction-Refinement

Counterexample-guided abstraction refinement (CEGAR) [19] is a precursor to the CEGIS approach described above. CEGAR solves a synthesis subtask of generating abstract models that are sound (they contain all behaviors of the original system) and precise (any counterexample for the abstract model is also a counterexample for the original system). One can view CEGAR as an instance of sciduction as follows:

![Diagram of CEGAR process](image)

**Fig. 3:** Counterexample-guided abstraction refinement (CEGAR) as inductive synthesis.

- The *abstract domain*, which defines the form of the abstraction function, is the structure hypothesis. For example, in verifying digital circuits, one might use localization abstraction [16], in which abstract states are cubes over the state variables.

- The inductive engine \( I \) is an algorithm to learn a new abstraction function from a spurious counterexample. Consider the case of localization abstraction. One approach in CEGAR is to walk the lattice of abstraction functions, from most abstract (hide all variables) to least abstract (the original system). This problem can be viewed as a form of learning based on version spaces [17], although the traditional CEGAR refinement algorithms are somewhat different from the learning algorithms proposed in the version spaces framework. Gupta, Clarke, et al. [23] have previously observed the link to inductive learning and have proposed versions of CEGAR based on alternative learning algorithms (such as induction on decision trees).

- The deductive engine \( D \), for finite-state model checking, comprises the model checker and a SAT solver. The model checker is invoked on the abstract model to check the property of interest, while the SAT solver is used to check if a counterexample is spurious.

As a point of contrast, the cone-of-influence approach [5] of computing an abstract model is a purely deductive approach. It is often the case that the abstract model computed via cone-of-influence is too detailed to provide the efficiency benefits of abstraction.

### 2.2.3 Directions for Future Work

There are four major differences between traditional machine learning algorithms (and their applications) and the ones used in formal verification and synthesis. First, in many traditional uses of machine learning, the learning algorithm operates by drawing (labeled) examples at random from some source with no control over the examples it draws. In contrast, the form of learning in verification and synthesis more commonly tends to be *active*, where the learning algorithm actively selects the examples it learns from. Second, in traditional machine learning, concept classes tend to be simpler than the ones considered in verification. In verification and synthesis, we are usually interested in synthesizing fairly general logical artifacts and programs. Third, in traditional machine learning, the learning algorithm tends to be a special-purpose algorithm designed for the specific concept class being learned. On the other hand, in inductive synthesis, the learning algorithm tends to be a general purpose logical reasoning engine, such as a SAT or SMT solver. Fourth, while machine learning is almost entirely data-driven, inductive learning in formal verification typically operates via a combination of data-driven and model-driven algorithms.

These differences mean that there is much more to be done to develop a theory of inductive learning in the context of formal methods. Specifically, one needs to understand the impact of different logical reasoning engines on accuracy and efficiency of learning, as well as the properties of a learning algorithm that might make it well-suited for use in an overall verification or synthesis task.

In this regard, there are two results obtained by the
author and colleagues that provide some initial insights. First, the notion of teaching dimension, elucidated by Goldman and Kearns [24], provides a lower bound on the number of examples needed by inductive synthesis techniques (such as CEGIS) to converge to the correct concept [25]. Further, one can formally analyze CEGIS to understand the impact of the quality of counterexamples provided by the verifier on the overall convergence to the correct concept [26]. Given the centrality of counterexample-guided learning in formal methods today, these results provide an initial basis for developing a deeper theoretical insights into its operation, and provide guidance for the use of other learning techniques for formal methods. An extended treatment of these results along with a theoretical framework may be found in [53].

3. Cyber-Physical Systems

Cyber-physical systems (CPS) are computational systems that are tightly integrated with the physical world [27]. Depending on the characteristics of CPS that are emphasized, they are also variously termed as embedded systems, the Internet of Things (IoT), the Internet of Everything (IoE), or the Industrial Internet. Examples of CPS include today’s automobiles, fly-by-wire aircraft, medical devices, power generation and distribution systems, building control systems, robots, and many other systems. CPS have been around for a long time, but it is only recently that the area has come together as an intellectual discipline. Many CPS operate in safety critical or mission-critical settings, and therefore it is important to gain assurance that they will operate correctly, as per specification. Thus, formal methods is an essential tool in the design of CPS. However, CPS also operate in high dynamic and uncertain environments, and therefore they must be designed to be robust and adaptive. This imposes additional challenges on the modeling, verification, and synthesis of such systems. We examine in this section some of the main challenges in CPS design and how formal methods can help in addressing these.

3.1 Distributed Cyber-Physical Swarms

One of the biggest trends in CPS today is the large-scale networking of devices. The author, along with several colleagues, is involved in a multiyear, multi-institution effort to develop systematic scientific and engineering principles to addressing the huge potential (and associated risks) of pervasive integration of smart, networked sensors and actuators into our connected world [28]. These distributed CPS are being termed as the “swarm” and are identified to have the following important characteristics [28]:

- **Large-scale**: the swarm comprises a vast number of nodes generating corresponding “big data;”
- **Distributed**: components of the swarm are typically networked, and are potentially separated physically and/or temporally;
- **Cyber-physical**: the swarm fuses computational processes with the physical world;
- **Dynamic**: the environment evolves continually;
- **Adaptive**: the system must adapt to its dynamic environment, and thus the distinction between “design-time” and “run-time” is blurred, and
- **Heterogeneous**: swarm components are of various types, requiring interfacing and interoperability across multiple platforms and models of computation.

This unique combination of characteristics necessitates advances in several topics in formal specification, verification and synthesis. In particular, there is a need for correct-by-construction synthesis techniques that allow quick adaptation and deployment of components into the swarm while ensuring that certain key design requirements are always satisfied.

In this section, we give the reader a flavor of the problems in the design of distributed cyberphysical systems through a sample problem: multirobot motion planning for complex requirements specified in temporal logic.

![Fig. 4: Compositional SMT-Driven Multi-Robot Motion Planning: (a) Top view of sample execution and associated simulation, and (b) Nano-quadrotor platform from KMel Robotics [29] (reproduced from [30]).](image)

Multi-Robot Motion Planning with Temporal Logic Objectives

In Robotics, the traditional motion planning problem is to move a robot from Point A to Point B while avoiding obstacles. However, more recently, there is growing interest in extending this problem along two dimensions. The first extension is to impose more complex requirements on the robot, such as visiting certain locations “infinitely often.” Such requirements can be conveniently specified in a formal notation such as linear temporal logic (LTL). The second extension is to handle swarms of many robots executing
coordinated plans. Such problems arise in many application settings, including persistent surveillance, search and rescue, formation control, and aerial imaging. More complex requirements require more sophisticated methods to ensure that the synthesized plans are provably correct. Scaling planning algorithms to larger swarms requires more efficient algorithms and design methodologies.

Recent work [30] addresses these challenges with a two-pronged approach. First, a compositional approach is employed, where pre-characterized motion primitives, based on well-known control algorithms, are used as a component library. Each motion primitive is specified in a suitable combination of logical theories. Second, using an encoding similar to the one used for bounded model checking, a satisfiability modulo theories (SMT) solver [11] is used to find a composition of motion primitives that achieves the desired LTL objectives. Figure 4 depicts a sample result of this approach, showing the top view of four nano quadrotor robots achieving a desired LTL specification.

These results are only a small first step. There are many more problems that remain to be solved, including inferring effective logical characterizations of motion primitives, handling dynamic and uncertain environments, dealing with non-linear encodings, incremental planning, and scaling up to an order of magnitude more robots.

3.2 Human-in-the-Loop Systems

Many cyber-physical systems are interactive, i.e., they interact with one or more human beings, and the human operators’ role is central to the correct working of the system. Examples of such systems include fly-by-wire aircraft control systems (interacting with a pilot), automobiles with “self-driving” features (interacting with a driver), and medical devices (interacting with a doctor, nurse, or patient). We refer to the control in such systems as human-in-the-loop control systems. The costs of incorrect operation in the application domains served by these systems can be very severe. Human factors are often the reason for failures or “near failures”, as noted by several studies (e.g., [31], [32]). Correct operation of these systems depends crucially on two design aspects: (i) interfaces between human operator(s) and autonomous components, and (ii) control strategies for such human-in-the-loop systems.

A particularly interesting domain is that of automobiles with “self-driving” features, otherwise also termed as “driver assistance systems”. Such systems, already capable of automating tasks such as lane keeping, navigating in stop-and-go traffic, and parallel parking, are being integrated into high-end automobiles. However, these emerging technologies also give rise to concerns over the safety of an ultimately driverless car. For these reasons, the field of semi-autonomous driving is a fertile application area for formal methods.

In this section, we give an overview of some of the challenges in the design of human-in-the-loop CPS (h-CPS), including:

- **Modeling** What distinguishes a model of a h-CPS from a typical CPS?
- **Specification** How does the formal specification change for a h-CPS?
- **Verification** What new verification problems arise from the human aspect?
- **Synthesis** What advances in controller synthesis are required for h-CPS?

For lack of space, we only give a brief preview of the main ideas. Further detail can be obtained in the papers by Li et al. [33] and Sadigh et al. [34].

3.2.1 Modeling

For a typical (fully autonomous) control system, there are three entities: the plant being controlled, the controller, and the environment in which they operate. In an h-CPS, we additionally have the human operator(s) and therefore, also need a subsystem that mediates between the human operator(s) and the autonomous controller.

Formally, we model a h-CPS as a composition of five types of entities (components) [33], as illustrated in Fig. 5. The first is the plant, the entity being controlled. In the case of automobiles, these are the sub-systems that perform the various driving maneuvers under either manual or automatic control. The second is the human operator (or operators); e.g., the driver in an automobile. For simplicity, this discussion uses a single human operator, denoting her by HUMAN. The third entity is the environment. HUMAN perceives the environment around her and takes actions based on this perception and an underlying behavior model. We denote by HP HUMAN’s perception of the environment and by HA the actions by HUMAN to control the plant. In the case of a fully-autonomous system, the human operator is replaced by an autonomous controller (AUTO, for short). In practice, each specialized function of the system may be served by a separately designed autonomous controller; however, for the sake of simplicity, we model the autonomous controller as a single component. AUTO perceives the environment through sensors (denoted by ES, “environment sensors”) and provides control inputs to the plant.

Fig. 5: Structure of a Human Cyber-Physical System. ES denotes environment sensing, HS denotes human sensing, HP denotes human perception, HA denotes human actions, and u denotes the vector of control inputs to the plant.
The distinctive aspect of h-CPS arises from its partial autonomy. We capture this by including a fifth component, the advisory controller (Advisor, for short) [33]. The function of Advisor is to mediate between Human and Auto. This mediation can take different forms. For instance, Advisor may decide when to switch from full control by Human to full control by Auto, or vice-versa. Advisor may also decide to combine the control inputs from Human and Auto to the plant in a systematic way that achieves design requirements. This is indicated in Fig. 5 by the yellow box adjoining plant, between it and the Auto and Human components. We note that for legal and policy reasons, it may not be possible in many applications (including driving) for Advisor to always take decisions that override Human. Hence, we use the term Advisor, indicating that the form of control exercised by Advisor may, in some situations, only provide suggestions to Human as to the best course of action.

3.2.2 Specification

h-CPS have certain unique requirements which need to be formalized as formal specifications for verification and control. We illustrate these by the example of semi-autonomous driving.

Recognizing both the safety issues and the potential benefits of vehicle automation, in 2013 the U.S. National Highway Traffic Safety Administration (NHTSA) published a statement that provides descriptions and guidelines for the continual development of these technologies [35]. Particularly, the statement defines five levels of automation ranging from vehicles without any control systems automated (Level 0) to vehicles with full automation (Level 4). We focus on Level 3 which describes a mode of automation that requires only limited driver control:

“Level 3 - Limited Self-Driving Automation: Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with sufficiently comfortable transition time. The vehicle is designed to ensure safe operation during the automated driving mode.”[35]

Essentially, this mode of automation stipulates that the human driver can act as a fail-safe mechanism and requires the driver to take over control should something go wrong. The challenge, however, lies in identifying the complete set of conditions under which the human driver has to be notified ahead of time. Based on the NHTSA statement, we have identified [33] four important criteria required for a human-in-the-loop controller to achieve this level of automation.

1. Effective Monitoring. The advisory controller should be able to monitor all information about the h-CPS and its environment needed to determine if human intervention is needed.

2. Conditional Correctness. When the autonomous controller is in control (and not the human operator) it must satisfy a given formal specification (e.g., provided in temporal logic).

3. Prescience. The advisory controller must determine if the above specification may be violated ahead of time, and issues an advisory to the human operator in such a way that she has sufficient time to respond.

Li et al. [33] show how these requirements can be made precise for the case of controller synthesis from linear temporal logic.

3.2.3 Verification and Control

h-CPS are not only an application area for formal verification and control, but they have also given rise to new classes of problems.

For verification, one such problem is the verification of quantitative models of h-CPS inferred from experimental data. An example is the work on probabilistic modeling and verification of human driver behavior by Sadigh et al. [34]. Here the aim is to infer a Markov Decision Process (MDP) model for the whole closed-loop system (including human, controller, plant, and environment) from experimental data obtained from an industrial-scale car simulator. Since the model is inferred from an empirical data set that is incomplete, the model has estimation errors, e.g., in the transition probabilities. Therefore, in this case we infer a generalization of an MDP called a Convex-MDP (CMDP) [36], where the uncertainty in the values of transition probabilities is captured as a first-class component of the model. Puggelli et al. [36] show how one can extend algorithms for model checking to the CMDP model. Sadigh et al. [34] use that model to infer desired properties about human driver behavior, such as a quantitative evaluation of distracted driving.

In the setting of control of h-CPS, given that we have new kinds of specifications (as described earlier), control algorithms have to be modified to synthesize not only the autonomous controllers but also the advisory controllers. Li et al. [33] show how to modify standard algorithms for synthesis from LTL for this purpose. It is still an open problem to adapt other kinds of control algorithms to the h-CPS setting.

4. Education

The advent of massive open online courses (MOOCs) [37] promises to bring world-class education to anyone with Internet access. Moreover, it has placed a renewed focus on the development and use of computational aids for teaching and learning. MOOCs present a range of problems to which the field of formal methods has much to contribute. These include automatic grading, automated exercise generation, and virtual laboratory environments. In automatic grading, a computer program verifies that a candidate solution provided by a student is “correct”, i.e., that it meets certain instructor-specified criteria (the specification). In addition, and particularly when the solution is incorrect, the automatic grader (henceforth, autograder) should provide feedback to the student as to where he/she went wrong.
exercise generation is the process of synthesizing problems (with associated solutions) that test students’ understanding of course material, often starting from instructor-provided sample problems. Finally, for courses involving laboratory assignments, a virtual laboratory (henceforth, lab) seeks to provide the remote student with an experience similar to that provided in a real, on-campus lab.

In this section we briefly describe two applications of formal methods to education. Both applications have been to an undergraduate course at UC Berkeley, *Introduction to Embedded Systems* [38]. In this course, students learn theoretical content on modeling, design, and analysis [39], and also perform lab assignments on programming an embedded platform interfaced to a mobile robot [40]. Thus, this course provides a suitable setting to investigate the range of problems associated with MOOCs that are mentioned the preceding paragraph.

4.1 Exercise Generation

Consider first the task of automatic exercise generation. It is unrealistic and also somewhat undesirable to completely remove the instructor from the problem generation process, since this is a creative process that requires the instructor’s input to emphasize the right concepts. However, some aspects of problem generation can be tedious for an instructor, and moreover, generating customized problems for students in a MOOC is impossible without some degree of automation. Additionally, creating many different versions of a problem can be effective at reducing cheating by blind copying of solutions.

Examining problems from all three parts of the Lee and Seshia textbook [39], Sadigh et al. [41] take a template-based approach to automatic problem generation. Specifically, several existing exercises in the book are shown to conform to a template. The template identifies common elements of these problems while representing the differentiating elements as parameters or “holes”. In order to create a new problem, the template essentially must be instantiated with new parameter values. However, it is often useful to create new problems that are “similar” in difficulty to existing hand-crafted problems. To facilitate this, new problems are generated using a bounded number of mutations to an existing problem, under suitable constraints and pruning to ensure well-defined results. An instructor can then select results that look reasonable to him or her.

For brevity, we outline some of the main insights reported in [41] as they relate to the application of formal methods. The first insight relates to the structure of exercises. After investigating the exercises from certain relevant chapters (Ch. 3, 4, 9, 12, 13) of Lee and Seshia [27], we found that more than 60% of problems fit into the model-based category, where the problem tests concepts involving relationships between models, properties and traces. Fig. 6 is an illustration of the three entities, and their characteristics. At any point, given one or two of these entities, we can ask about instances of the unknown entity. Table 1 groups exercises into different classes based on what is given and what is to be found. Each group represents an interaction between models, properties and traces. The first column shows the given entity, and the second column is the unknown entity. The third column shows some of the variations of the same class of problem.

Table 2 states a solution technique for each problem category listed in Table 1. Note that major topics investigated in formal methods such as model checking, specification mining, and synthesis can be applied to various tasks in exercise generation. Moreover, since textbook problems are typically smaller than those arising in industrial use, their size is within the capacity of existing tools for synthesis and verification.

4.2 Grading and Feedback in Virtual Laboratories

Lab-based courses that are not software-only pose a particular technical challenge for MOOCs. A key component
of learning in lab-based courses is working with hardware, getting “one’s hands dirty.” It appears to be impossible to provide that experience online. And yet, it would be useful to provide a learning experience that approximates the real lab as well as possible. Indeed, in industrial design one often prototypes a design in a simulated environment before building the real artifact.

In an ideal world, we would provide an infrastructure where students can log in remotely to a computer which has been preconfigured with all development tools and laboratory exercises; in fact, pilot projects exploring this approach have already been undertaken (e.g., see [42]). However, in the MOOC setting, the large numbers of students makes such a remotely-accessible physical lab expensive and impractical. A virtual lab environment, driven by simulation of real-world environments, appears to be the only solution at present.

The author and colleagues have taken initial steps towards addressing this challenge for lab-based education [43], [44]. The main contribution to date is CPSGrader, which combines virtual lab software with automatic grading and feedback for courses in the areas of cyber-physical systems and robotics [44], [45], [46]. In particular, CPSGrader has been successfully used in Introduction to Embedded Systems at UC Berkeley [38] and its online counterpart on edX [47]. In the lab component of this course, students program the Cal Climber [40] robot (see Fig. 7) to perform certain navigation tasks like obstacle avoidance and hill climbing. Students can prototype their controller to work within a simulated environment based on the LabVIEW Robotics Environment Simulator by National Instruments (see Figure 8 and 9).

The virtual lab dynamical model is a complex, physics-based one. CPSGrader employs simulation based verification, the only scalable formal approach. Correctness and the presence of certain classes of mistakes are both checked using test benches formalized in Signal Temporal Logic (STL) [48]. However, coming up with these STL properties can be tedious and error-prone, even for instructors well-versed in formal methods. Therefore, Juniwal et al. [44] show how these temporal logic testers can be generated via machine learning from solutions that have the fault (positive examples) and those that do not (negative examples). An active learning framework has also been developed to ease the burden of labeling solutions as positive or negative [45]. In machine learning terminology, this can be thought of as the training phase. The resulting test bench then becomes the classifier that determines whether a student solution is correct, and, if not, which fault is present. CPSGrader was used successfully in the edX course EECS149.1x offered in May–June 2014 [47].

### 5. Conclusion

In summary, in this paper we have outlined some exciting trends in formal methods and its applications. These include the combination of formal methods and inductive machine learning, and new applications to cyber-physical systems and education. Apart from these trends, there are many other exciting topics including new research in computational engines for formal methods (e.g., [49]) and emerging applications to areas such as synthetic biology [50] and computational music [51], [52].

### Acknowledgments

The author thanks Prof. R. K. Shyamasundar who encouraged him to write this article. Some of the work outlined in Sec. 2 is joint with Susmit Jha. The research described in Sec. 3.1 is joint with Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, and George Pappas, that in Sec.

---

**Table 2: Techniques to Find Solutions for Model-Based Problems (reproduced from [41])**

<table>
<thead>
<tr>
<th>Given</th>
<th>Find</th>
<th>Solution Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td>( M )</td>
<td>Constrained Synthesis or Repair</td>
</tr>
<tr>
<td>( M )</td>
<td>( \psi )</td>
<td>Simulation of Model</td>
</tr>
<tr>
<td>( M )</td>
<td>( \phi )</td>
<td>Specification Mining</td>
</tr>
<tr>
<td>( M ) &amp; ( \phi )</td>
<td>( \psi )</td>
<td>Simulation with Guidance</td>
</tr>
<tr>
<td>( M ) &amp; ( \phi )</td>
<td>( \psi )</td>
<td>Model Checking</td>
</tr>
</tbody>
</table>

---

**Fig. 8: Cal Climber in the LabVIEW Robotics Environment Simulator.**

There are several interesting directions for future work, including developing quantitative methods for assigning partial credit, mining temporal logic testers to capture new mistakes, and online monitoring of these testers to improve responsiveness.

**Fig. 9: Simulator with auto-grading functionality used in EECS 149.1x.**

---

**Fig. 10: Diagram showing the process of generating temporal logic testers using machine learning.**

---

**Fig. 11: Diagram showing the process of using machine learning to classify solutions as positive or negative.**

---

**Fig. 12: Diagram showing the process of using machine learning to predict the presence of certain classes of mistakes.**

---

**Fig. 13: Diagram showing the process of using machine learning to improve the responsiveness of automated grading systems.**

---

**Fig. 14: Diagram showing the process of using machine learning to improve the accuracy of diagnosis systems.**

---

**Fig. 15: Diagram showing the process of using machine learning to improve the efficiency of automated grading systems.**

---

**Fig. 16: Diagram showing the process of using machine learning to improve the reliability of automated grading systems.**
3.2 is joint with Wenchao Li, Dorsa Sadigh, and S. Shankar Sastry, and the work in Sec. 4 is joint with Alexandre Donz’e, Mona Gupta, Jeff Jensen, Garvit Juniwal, Edward A. Lee, and Dorsa Sadigh. The work described in this paper was supported in part by an Alfred P. Sloan Research Fellowship, NSF CAREER grant #0644436, NSF Expeditions grant #1139138, and TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References


About the Author

Sanjit A. Seshia received the B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Bombay, India in 1998, and the M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University, Pittsburgh, PA, USA, in 2000 and 2005 respectively.

He is currently an Associate Professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, CA, USA. His research interests are in dependable computing and computational logic, with a current focus on applying automated formal methods to problems in embedded and cyber-physical systems, electronic design automation, computer security, and synthetic biology. His Ph.D. thesis work on the UCLID verifier and decision procedure helped pioneer the area of satisfiability modulo theories (SMT) and SMT-based verification. He is co-author of a widely-used textbook on embedded systems. He led the offering of a massive open online course on cyber-physical systems for which his group developed novel virtual lab auto-grading technology based on formal methods.

Prof. Seshia has served as an Associate Editor of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, and as co-chair of the Program Committee of the International Conference on Computer-Aided Verification (CAV) in 2012. His awards and honors include a Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House, an Alfred P. Sloan Research Fellowship, the Prof. R. Narasimhan Lecture Award, and the School of Computer Science Distinguished Dissertation Award at Carnegie Mellon University.
As software becomes increasingly pervasive and affects critical areas of application, verifying the correctness of programs can no longer be neglected. Several approaches in the past have utilized testing to prove program correctness, but this is an incomplete approach. The second alternative is to perform static analysis and model checking. While this is an exhaustive approach, it has several limitations viz; high cost, poor scalability, spurious warnings. In this paper, we explore Symbolic execution that takes the middle way between these two extremes, and has the potential to be applicable in real world settings. Initially proposed in the 1970’s, symbolic execution has recently gained focus amongst researchers. Starting from its birth, we survey tools that have successfully implemented symbolic execution and the modifications that have been proposed to make it widely applicable. We also examine the research challenges that exist with this approach and how well-received it has been in industry.

Index Terms: Static Analysis, Symbolic Execution, Dynamic Test Generation, Concolic Execution, Compositional Testing.

1. Introduction
As software is being used in increasingly complex and critical applications, the importance of program verification cannot be stressed enough. Conventionally two approaches to verification have been cited - static analysis and dynamic analysis. Both these approaches however, have their own shortcomings that have curtailed their use in practice. We first introduce these conventional methods of verification, examining their pros and cons. We then look at the idea of symbolic execution, as it was originally proposed by King [1] in 1976. Symbolic execution was essentially a static analysis method. However it has recently evolved to a new form, imbibing advantages of both static and dynamic code analysis. The possible areas of application of this novel form of symbolic execution are then discussed in later sections.

1.1 Static Analysis of Programs
An excellent introduction and review of static analysis can be found in the book by Nielson [2]. Static analysis of code, as the name suggests, tries to analyze the program behavior without actually executing the code. Thus the correctness of the program is analyzed by applying formal techniques on the source code or the object code. One of the earliest examples of a static analysis tool is the well-known Lint program for C. Such bug finding tools generally use static analysis only in a shallow way, meaning that they look out for unexpected constructs in the program. This means that they will also raise an increased number of false alarms, while missing out on subtle errors. Some important concepts associated with any static analysis approach are whether the approach is sound and complete. A sound approach is guaranteed to find any true error in the program; a complete one will find only errors that are true. Some of the commonly used methods of formal static analysis are summarized below:

1.1.1 Type Systems
Types are most widely used in static analysis. This is because most of the major programming languages use the concept of types. A type is just a set of values. For e.g the type Int would denote all the integers whereas the type Bool in a language like C++ would represent the set true, false. The concept of types in fact follows from the concept of abstract interpretation, which is used in any static analysis approach. Ideally static analysis should be able to represent all possible executions of a program. This is however un-decidable in practice. So a compromise is to consider only abstract values rather than the infinite set of all possible concrete values. Thus Int is an abstraction for the infinite possible integers.

1.1.2 Data Flow Analysis
Data flow analysis usually requires us to construct a Control Flow Graph (CFG). This is a directed graph in which each node is a statement and edges represent the flow of control. Data flow analysis tries to find facts about the program by considering all possible states of the program. It includes different analyses such as liveness analysis, reaching definitions, etc., that are used in optimizing compilers. Detailed literature on data-flow analysis may be
found in any standard text on compilers such as [3].

1.1.3 Model Checking

Model checking usually uses some form of temporal logic such as finite automata, to represent the specifications of a program. The state space is then exhaustively searched, to ensure that the specifications are met correctly. It is an approach that suffers from state space explosion. Bounded model checking examines only a prefix of all possible executions. However this may result in later errors being missed.

1.2 Dynamic Analysis of Programs

Dynamic analysis is mainly done through testing and debugging. Testing tries to run a program on a subset of the possible inputs to observe if a failure occurs or not. Since it is not possible to run the program exhaustively over all possible inputs, testing has the problem of absence of coverage. One important concern is that the test case generation should be automated if it is to be applicable to commercial programs that are thousands of lines long. The way these tests are generated is important, since naively generating the test cases in any random manner may likely miss out on several errors.

Debugging involves pinpointing the exact location of errors, and fixing these up. In earlier times, debugging was usually achieved by inserting print statements at suspicious locations in the program. With the advent of IDEs and better debuggers, this situation is somewhat eased, however debugging even today is highly arcane and tedious, requiring much manual involvement.

A good visualization of these different methods is shown in Figures 1 2 3. In each of these figures x(t) is a vector of the input state and output of the program that evolves as a function of time t. Figure 1 shows the possible different trajectories that a program may take up when executed. As Figure 2 shows, model checking tries to cover each of the trajectories, however to be practical, it must be bounded. Thus it misses out on any late errors. On the other hand testing as shown in Figure 3 will leave out some of the trajectories entirely while some are covered completely. Clearly even this is not satisfactory.

1.3 Symbolic Execution : Old Paradigm Gets New Life

The concept of symbolic execution was first proposed by King in his seminal paper [1]. We first try to understand the concept in the form it was originally proposed. As we have already noted, program verification conventionally is done either through proving by formal techniques, or by testing with a random input set. Symbolic execution is a middle way between these two extremes. In symbolic execution, the program is executed, but the input fed to the program is symbolic rather than concrete. This means that each symbolic input stands for an entire class of inputs rather than individual input. Thus while testing tries to check the program only on a very small subset of the possible inputs and verification checks the program over all of the possible inputs, symbolic execution checks the program over classes of the input. So it may be considered to be a more generalized form of testing or a less stringent verification.

Whenever a program is executed concretely, the program state usually consists of the program counter and the values of the program variables. When a program is to be executed symbolically, one additional piece of information must also be maintained in the so-called symbolic program state. This is the path condition (pc). The pc is the accumulation of all the criteria that the input must satisfy for an execution to follow the associated path [1]. As we encounter, conditions or branches in the program flow, such as IF statements, these conditions are conjoined with the pc along that path. Thus a pc exists, corresponding to each alternative path that the program may follow. This pc is thus a Boolean formula. In order to reason about whether a particular control flow is possible for the program, we should check whether the corresponding Boolean formula that represents its pc is satisfiable. For this powerful SMT/SAT solvers are used. Of course while many satisfiability problems are beyond...
the scope of these solvers, many solvers do exist for linear equations, e.g. Z3, STP, Yices, etc.

Thus, corresponding to each program, we can generate the execution tree that shows all possible execution paths of the program. One such symbolic execution tree and the corresponding code snippet appear in the Figure 4. Here it is seen that at each point in a particular path, the pc is updated with some additional constraints if necessary. If the final pc for a particular path is found to be un-satisfiable, then that state of the program is unreachable. The various path conditions accrued may be used to generate test cases. All concrete values that satisfy the path condition for a particular path are guaranteed to follow that path when inputted to the program.

It is important to note that classical symbolic execution, will work only for sequential programs. In recent times, several extensions to this classical approach have been proposed. Excellent reviews of the current trends in symbolic execution are available in [6] [7]. These new approaches include ideas like generalized symbolic execution, concolic symbolic execution, compositional symbolic execution, etc. They have made it possible to extend symbolic execution to multi-threaded programs and advanced programming constructs, such as recursive data-structures. We shall survey a few of these ideas in a later section.

1.4 Applications of Symbolic Execution

The paper by Visser et al [8] covers a depth of areas to which symbolic execution may be applied. Here we enlist some major areas of focus.

1.4.1 Test case generation

This is one of the traditional uses of symbolic execution. Since symbolic execution deals with classes of input, it can be used to automate generation of test-cases with a high degree of coverage. This has recently become well known as whitebox testing.

1.4.2 Proof of program properties

Symbolic execution can be successfully applied for proving that certain assertions, say for instance loop invariants, are indeed maintained when a program is executed.

1.4.3 Static detection of run time errors

This uses symbolic execution to detect if program states, that may lead to run time violations, exist. Such analysis is especially useful in the case of malware analysis, where it may be expensive to run the code in order to observe its noxious behavior.

1.4.4 Invariant inference

By using symbolic execution, we may predict the pre or post conditions that are likely to be maintained by the program.

1.4.5 Parallel numerical program analysis

This is a new application of symbolic execution to establish that the parallel program is indeed equivalent to its corresponding sequential counterpart.

1.4.6 Differential symbolic execution

This is again an emerging area. Here two programs are compared to find out the logical difference between them. It may be used in software development and maintenance to ensure for instance that re-factored code is equivalent to the original source.

The rest of the paper is organized as follows: In section 2 we discuss some tools such as KLEE that are developed using symbolic execution. Some exciting new trends that are emerging in symbolic execution are described in section 3. Section 4 examines some of the major challenges that symbolic execution techniques must counter. Finally, in section 5, we see how well accepted the various symbolic execution tools have been in industry.

2. Symbolic Execution: Case Studies

In this section we explore two tools that have successfully used symbolic execution for program analysis. Several breakthrough tools have been proposed, but we focus on only two such contemporary tools. Our first case study is on KLEE - which is an automatic test case generation framework for C. Since C is a conventional statically typed language, the study of KLEE will acquaint us with how symbolic execution can be applied in such a mainstream language. Our second case study deals with Kudzu - which is an automated vulnerability analysis tool for JavaScript. A study of Kudzu will give us a flavour of how symbolic execution may be applied to solve string constraints, a major application challenge.

2.1 Case Study 1: KLEE

KLEE is one of the most successful implementations of symbolic execution as a means for automated generation of high coverage test cases. The paper that provides detailed technical overview of KLEE is [9]. KLEE has two major execution goals:

1) Try to hit every line of code. This will help achieve high coverage.
2) At each dangerous operation that is detected, try to
generate the test case that could potentially cause the error. Do this by solving that path’s path condition.

The authors of this paper were previously involved in the development of a similar tool EXE [10]. They have based KLEE on their previous experiences and have shown that KLEE generated tests could indeed achieve high test coverage of over 90%, sometimes even beating manually developed test suites. Here we briefly explore some major features.

2.1.1 Architecture

The KLEE framework is designed for:

Simplicity: It operates on the well known LLVM byte code. This means that it can be applied to a number of languages that are supported by LLVM. LLVM has a RISC like instruction set and the byte code is generated from the source code, which is compiled using the well known GCC compiler. Thus no special modifications need to be made in order to have the code analyzed by KLEE. KLEE can take in the raw code directly and generate the test cases, and is therefore quick and easy to use.

Scalability: Since there can be an exponential increase in the number of states, KLEE uses compact state representation. It borrows the well-known principle of copy on write (vfork vs. fork) from Linux, to reduce memory requirements. It also implements the heap as immutable and shares the heap amongst multiple states. Finally it uses compact expression representation (e.g. when it has expression x+1=10 simply store this as the expression x=9.)

Speed: It achieves speed by trying to minimize the calls to the SAT solver, since these are the most expensive. It also uses simplified expressions as input to the constraint solver, and uses the concept of caches to dramatically improve speed.

The figure 5 shows how the different components of KLEE integrate to produce very accurate test cases for the sample code snippet on the lower-left in the figure.

2.1.2 Solver Optimization

The major time consumed by most symbolic execution tools is in trying to solve the constraints. Thus KLEE has used some clever techniques to greatly reduce this time cost. It uses the concept of constraint independence to give only relevant portions of the memory to the solver for a particular constraint rather than the complete store of all the variables.

It uses the concept of counter-example cache to store results of previous queries. These results are checked to see if they satisfy a current query, since it is much easier to check the solution rather than solve. Also it tries to see if the cached solution has some subset, superset relation to the current constraint’s solution state.

2.1.3 Major Challenges

KLEE has tried to answer two challenges:

1) How to model the interaction of the program with the environment: Many programs have command line arguments, environment variables, file handling, etc. To provide a realistic environment in such cases, KLEE makers have tried to provide a symbolic environment, which the program sees when it is executed symbolically. This tries to model its interaction with real-life system calls and OS resources.

2) How to address exponential paths through the code: The state space may grow exponentially. So KLEE uses two strategies to determine which state to choose next from the current state. The first strategy is Random Path Selection - this uses a binary tree to represent active states, and the tree is randomly traversed from the root. The approach is biased towards states higher up in the tree where simpler constraints need to be solved. The second is Coverage Optimized Search where it uses some heuristics to compute weights for each state and then randomly selects the state that is likely to offer better coverage.
2.1.4 Evaluation Highlights

The metric for evaluation is line coverage. KLEE generated test cases for Coreutils, Minix, Busybox, etc. It was able to outperform manually designed test suites developed over a period of 15 years. The coverage was over 90%. It found several major flaws in GNU Coreutils, which is one of the most heavily tested program suite. Thus it was shown to have great potential for real world application.

2.2 Case Study 2: Kudzu

Kudzu [13] is an automated tool for finding client side code injection vulnerabilities in JavaScript. It is the first attempt of this kind. In a language like JavaScript, the input is in the form of strings. This means that the solver should be able to solve string constraints. To enable this, Kudzu introduces an expressive constraint language and a constraint solver Kaluza specifically geared for handling string constraints, rather than numeric ones.

2.2.1 System Design

For the rich web applications that are created using AJAX, the input space is conceptually divided into 2: event space - that deals with various event handler code e.g. mouse clicks or form submissions, that may occur in any order - and secondly value space - these are the values provided by the user in form fields, text areas, URL parameters, etc. Kudzu uses the concept of GUI exploration to handle vulnerabilities associated with the event space while using symbolic execution to handle the untrusted input. Next, it incorporates the powerful string constraint solver Kaluza. For this, the original JavaScript instructions are translated to a simple language - JASIL. A block diagram for the system architecture appears in Figure 2.1.1. Kudzu is the automated tool for finding security vulnerabilities in JavaScript code.

2.2.2 Constraint Language

The constraint language is very expressive for string constraints. Various constraints are provided to check if a given string matches a given regular expression, comparing two strings for equality or concatenation, comparing the length of two strings and checking the length of a string against some integer. All these facilities make the constraint solver more powerful than its contemporaries. Moreover in-spite of its high expressiveness, the constraint language is simple, making the constraint solver efficient.

2.2.3 Constraint Solver

Kaluza is a SAT based SMT constraint solver. It first of all takes in as input the JavaScript code and translates it to the core constraint language using a DFA-based approach. Next, it solves for various constraints such as length and integer constraints, etc. Finally it takes the input strings and translates them into bit vector notation, by concatenating the binary representation of consecutive characters in the string. It then checks to see if the bit vector notation of the string contents satisfies the constraints using the SMT solver. It also uses the concept of k-boundedness, and expects the user to input the value of k - the maximum length of the strings to be included in the search space. This is essential as otherwise the problem would become unbounded and unsolvable practically.

2.2.4 Evaluation

The solver is very successful for detecting client side code injection attacks. It was able to detect 2 new vulnerabilities when tested on 18 popular web apps and also 9 known vulnerabilities that had earlier been detected using manual testing. The constraint solver is also highly efficient, when a constraint is satisfiable, it can find the solution in under a second, when it is not it may take up-to 50s to report failure. The only requirement is that the user must provide an upper bound on the search space.

3. Exciting New Areas in Symbolic Execution Research

Several recent trends have emerged that attempt to carry out some form of hybrid analysis that tries to combine symbolic execution with other techniques in an attempt to improve the overall performance. Several such techniques have been proposed for testing critical applications such as in NASA [14] [15]. We survey some such trends in the present section.

3.1 Concolic Execution: Combining Symbolic and Concrete Analysis

A very concise tutorial on this technique is available in [16]. Concolic execution combines both random testing as well as symbolic execution. Whenever a concrete execution is performed using random inputs, the path conditions for that particular path are also simultaneously constructed. In the next run the gathered constraints are used to generate new inputs, which will drive the execution along a different branch. For e.g. by negating one constraint at a branch point, the new input generated will then guide the program along the other branch. This is repeated until no new constraints can be generated. Thus significant coverage is attained. At the same time, if some constraints are too difficult for the solver, then the concrete input comes to rescue, by replacing some of the symbolic variables with concrete values. A good case study of such a hybrid system is DART [17]. DART is an algorithm for dynamic test case generation using random testing and symbolic execution. It is one of the precedents in this area, originally developed at Bell Labs. Here the major advantages of this approach are that the random test generation is fully automated, not requiring any manual intervention, thanks to the path constraints generated by symbolic execution. Unlike random testing it is also more effective, since the program execution may be directed along specific paths by using the path constraints. It also tackles the difficulties of pure
symbolic execution where constraints may be too difficult to solve. Table 1 shows some recent tools that have been used to facilitate symbolic execution.

### 3.2 Compositional Symbolic Execution

Compositional symbolic execution is an attempt to extend concolic execution to make it more scalable by intelligently avoiding state space explosion. The seminal paper introducing this concept is [18]. Here the major concept used is to generate summaries for individual functions. These include information such as the function pre and post conditions, etc. These summaries are generated in top down manner in context of the caller-callee relationships between the functions. Thus if a function f() calls a function g(), then g()'s summary is generated first and used later when the function f() is being analyzed. By using such an approach, the overall complexity would be lesser than if both the functions are analyzed together. With this new approach, the complexity may be curbed while at the same time not sacrificing the code coverage.

For this approach, the authors of [18] have proposed an algorithm called SMART. They have reported that SMART is able to achieve the same level of coverage as the earlier DART algorithm. The Fig. 7 shows an experimental comparison of the number of runs with SMART and DART. SMART shows only a linear growth as compared to DART, which may show exponential scaling in the number of possible execution paths. The major challenge is to generate the summaries intelligently, rather than naively generating all possible summaries, since we want to counter path explosion.

![Fig. 7: Experimental comparison of the number of runs with SMART and DART [18]](image)

### 3.3 Automated Whitebox Fuzz Testing

Fuzz testing is a heavily used random testing method for detecting security vulnerabilities. This basically involves sending input data to the application and then randomly mutating the input and then testing the results. It is a form of black box testing. Whitebox fuzz testing [19] is an attempt to combine fuzz testing with symbolic execution, in an attempt to facilitate dynamic and automated test generation. It leverages off DART and other predecessors. Whitebox fuzz testing is much more efficient than the conventional black box methods. It can find several errors that are beyond the reach of the black box fuzzers.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolic (Java)</td>
<td>It is a test generation system based on the symbolic execution model, based on the NASA Java PathFinder (JPF), which initially used model checking. It implements generalized symbolic execution, that adds multi-threading and other capabilities in extension to classical symbolic execution. It has helped discover subtle bugs in several critical NASA systems.</td>
</tr>
<tr>
<td>PathFinder</td>
<td>It is a generation system based on the symbolic execution model, based on the NASA Java PathFinder (JPF), which initially used model checking. It implements generalized symbolic execution, that adds multi-threading and other capabilities in extension to classical symbolic execution. It has helped discover subtle bugs in several critical NASA systems.</td>
</tr>
<tr>
<td>DART</td>
<td>It tries to combine random testing with symbolic execution, to maximize the coverage, while being more efficient as well. It is an example of both symbolic and concrete execution or concolic execution. It was developed for C.</td>
</tr>
<tr>
<td>CUTE and jCUTE</td>
<td>It extends DART to handle multi-threaded programs. Can also perform pointer analysis and solves pointer constraints. CUTE is for C whereas jCUTE is for Java.</td>
</tr>
<tr>
<td>CREST</td>
<td>It is an open source tool designed for C. It is also extensible and has been used as a basis for several other tools.</td>
</tr>
<tr>
<td>SAGE</td>
<td>It is an automated whitebox fuzz tester, that uses compositional symbolic execution. It extends unit testing to whole application testing.</td>
</tr>
<tr>
<td>Pex</td>
<td>It is a test generation tool for .NET code. It was developed at Microsoft as a Visual Studio Tool.</td>
</tr>
<tr>
<td>EXE</td>
<td>It is a symbolic execution tool for C, written especially to check complex code and low level systems code. It introduces several optimizations, to achieve considerable speed up.</td>
</tr>
<tr>
<td>KLEE</td>
<td>It is the successor of KLEE which is based on the LLVM framework. It extends EXE by considering environmental interaction and better memory management for storing states.</td>
</tr>
</tbody>
</table>
Whitebox fuzzing introduces several new concepts to extend symbolic testing to whole applications. It tries to deal with path explosion and other obstacles by introducing a novel parallel state space search algorithm called generational search algorithm. Also because of other technological improvements such as more advanced SMT solvers and concise constraint representations, it can be applied to programs having millions of lines - such as large file parsers, etc.

The SAGE [20] fuzzer based on this technique has become a highly popular tool and is used regularly at Microsoft Corporation. It has saved millions of dollars and found several bugs. It represents the largest ever use of a SMT solver. We examine SAGE in more detail in section 5.

4. Symbolic Execution : Major Challenges and Proposed Solutions

There are several challenges that are an area of concern for symbolic execution. Some of these are enlisted below:

1) How to extend symbolic execution to handle complex and recursive data structures and loops?
2) How to handle multi-threaded and nondeterministic programs with symbolic execution?
3) How to extend the constraint solver to solve string constraints and support other native code features?

A good survey of these and other challenges is presented in [8]. Here we focus on the two major issues that have become a point of focus for researchers. These are:

1) How to curb the path space explosion?
2) How to optimize constraint solving?

Here we present 3 different approaches that attempt to overcome one or both of the above mentioned hurdles.

4.1 RWSet Analysis Based Path Pruning

The idea of RWSet analysis for countering path explosion was proposed in [21].

4.1.1 Approach

Here the authors propose to make symbolic execution more scalable by using the concept of Read-Write Set (RWSet) analysis, to discard all paths in the execution tree that will produce the same effect as a previously executed path. Here two major ideas are used to prune the paths. Firstly whenever an execution reaches a program point in exactly the same state as a previous execution then it should be pruned. Secondly if an execution differs from a previous execution only in values that are never going to be written, then it cannot cause a different execution flow, so even it should be discarded. By truncating suffixes of all such paths, significant gains are expected, because at each branch point these suffixes could spawn an exponential number of paths.

4.1.2 Implementation

The constraint cache is used to record the states corresponding to which a program state was reached. Now for the currently executing path, if we ever get a cache hit, then obviously the execution of that path should be aborted. The concept of write set is used to store only the difference of various concrete states from a common initial state, since storing the complete states may be too expensive. The concept of read set is used to maintain the values that will be read beyond the current program point. All other values should be discarded from comparison. Also the cache must be call site sensitive as otherwise a state stored in the cache for one function call may lead us to erroneously discard a path in some completely unrelated function call.

4.1.3 Evaluation

The tool EXE [10] was run, both with RWSet and without it on some benchmarks. It was found that RWSet significantly reduced the number of paths explored to achieve the same coverage. In all cases less then half the number of paths were explored with RWSet, in a few cases the number dropped to as low as just 11% of the original EXE. The overhead of enabling RWSet was also very less; approximately just 4%.

4.2 Parallel Symbolic Execution

Parallelizing the symbolic execution has been proposed in [22].

4.2.1 Approach

The paper proposes to reduce the time spent in exploring paths of the symbolic execution tree, by means of parallelization. The authors use a set of pre conditions to partition the execution tree and distribute the symbolic execution amongst different workers. The partitioning is done such that each worker can independently execute its subtree without any need of interprocess communication (IPC). This is important because otherwise the overhead of IPC may completely negate the gains from parallelization. Initially a shallow form of symbolic execution is run to generate the preconditions which will help partition the execution tree. The authors have named this as Simple Static Partitioning.

4.2.2 Implementation

The authors attempt to parallelize symbolic execution in a generic manner to make it applicable to different environments such as multi core computers, grid or clouds. The concept of Simple Static Partitioning is used to distribute work amongst different workers. For this first of all an initial set of constraints is generated via shallow symbolic execution. It is important to obtain a set of constraints that are disjoint, complete and useful. This ensures that each worker will do some useful work that does not overlap with any of the other workers. The different partitions that are thus obtained are stored in a queue. The user can control both the depth of the initial symbolic execution and the queue size. Both of these are critical factors that determine the success of the algorithm. The size of the constraint queue will determine how effectively the load balancing will be whereas the initial depth will determine the quality of the generated partitions.
4.2.3 Evaluation

While the paper cites several detailed metrics for evaluation, the results in a nutshell show that the speed up is possible in all systems. The results also show that the number of parallel workers (NPW) and the constraint queue size affect the speed up. There was 90x speedup in analysis time with NPW = 128 and 70x speed up in automatic test generation, with NPW = 64.

4.3 Memoized Symbolic Execution

In the final approach we present, the authors [12] use memoization to counter both the path explosion problem and the constraint solving cost.

4.3.1 Approach

The main idea is that usually symbolic execution is run several times, for instance whenever an error is detected, the program is modified and then re-run. The authors attempt to leverage from the information gained during earlier runs. To this end they maintain a trie structure, that helps in recording efficiently the states of a particular execution. The computations can be re-used during the next run. For example paths that were not useful in prior runs may be pruned and the constraint solver may be turned off for constraints that were previously solved. At each run the trie structure must also be appropriately modified.

4.3.2 Implementation

The trie data structure is used to store the information from each run. A new trie node is created whenever a branch is encountered during the symbolic execution. The node is very lightweight - storing just the method and instruction offset and the choice taken. The trie is first of all initialized. During the first execution the trie is constructed to guide future executions. The nodes in the trie are split into bound nodes and unsatisfiable nodes. This trie can later be searched efficiently the states of a particular execution. The nodes in the trie are split into bound nodes and unsatisfiable nodes. This trie can later be searched using either BFS or DFS. During the memoized analysis phase, the trie is loaded into the memory. It guides the execution; it is appropriately modified and also compressed to remove irrelevant features in the current context. Finally in the trie merging phase the compressed trie may be optionally merged with the older trie to obtain the complete trie.

4.3.3 Evaluation

It was found that the savings obtained depend on how the program is changed and where it is changed, during for e.g. regression analysis. During each execution, however, it was observed that by memoization there was a dramatic drop in the number of calls to the solver. However while this did not reflect significant time gains for simple-to-analyze programs, the authors predict that memoization may be very useful for critical and complex programs.

5. How applicable is Static Analysis in the real world?

In this section, we discuss three static analysis tools to see how useful they actually are in practice.

5.1 The FindBugs Experience

FindBugs is an open source static analysis tool that was developed at the University of Maryland. In the year 2009, Google held a large scale FindBugs Fixit. This involved over 300 engineers, reviewing thousands of warnings. This was an attempt to evaluate how effective FindBugs would be in practice. Over the course of two days, it was found that static analysis does catch mistakes. However most of these are not very important. Static analysis may at best catch about 5% to 10% of software quality problems. However it is cheaper. If used at an earlier stage static analysis can reduce the development cost. Overall however the results of the review were disappointing, as most of the issues reported by the tool were low on priority and did no cause any significant misbehavior.

5.2 Coverity: Static Analysis in Real World

In this popular paper [23] that was downloaded a record number of times from the CACM website, the authors provide some interesting analyses of why static analysis tools fail to do well in practice.

- Firstly to make a tool popular some sort of trial run or demo must be given to customers. Many a times the customer’s do not allow the tool to access some of their code for safety and privacy concerns. Thus the tool is unable to find the serious or important bugs and fails to convince the potential customers.
- Many a times the compilers that the company uses to write the code are in-fact buggy. They often deviate from the laid down standards for languages like C or Java. This leads to totally unexpected behavior in some cases when the tool is used along with such compilers. However the companies are unwilling to make any changes in their development environment which forces some ugly workarounds.
- The programmers often don’t take the bugs reported by the tool seriously. If they can’t understand a reported bug they pass it off as a false positive, rather than trying to analyze the code base.
- Often the bugs reported by two or more static analysis tools are different. Often the static tools themselves are erroneous so that the developers don’t have much faith in them.
- If the tool reports false positives of more than 30% then people generally tend to ignore the tool.
- Often the programmers write up any sort of junk for the code. The tool must be designed to take into consideration some of the foolishest errors.

5.3 SAGE : Whitebox Fuzz Tester

In this paper [20] again from the CACM, the picture is remarkably different and rosy from the previous two experiences. We have already described white box fuzzing in an earlier chapter. SAGE the tool designed, based on this
has made a massive impact in the development scenario at Microsoft. We list some of these below:

- SAGE has been running 24/7 on 100+ machines since 2008.
- Since SAGE has extended testing to whole application level, it has found hundreds of bugs in apps, media players, etc that were missed by everything else.
- It has found 33
- Due to SAGE millions of dollars lost in bugs and patches have been saved.
- It holds the record for the largest computational use of the SMT solvers.
- SAGE is today used daily in many groups at Microsoft. It is easy to deploy and fully automated.

This shows that when static analysis is combined with symbolic execution, there may possibly be dramatic rise in applicability.

6. Conclusion & Research Directions

In this paper, we discuss program testing and verification, focusing on symbolic execution. We explore tools - KLEE and Kudzu that apply symbolic execution, next we look at the upcoming paradigms in symbolic execution - concolic and compositional execution. Finally, we have highlighted major challenges to symbolic execution and three possible solutions namely - parallelization, memoization and pruning, to address them. Research directions to go from here are toward improving the scalability of symbolic execution, and heuristic searches that will help to explore more interesting program states. Another interesting area would be to develop more generalized and powerful constraint solvers.

References


About the Authors

Dr. Dhiren Patel

Dr. Dhiren Patel is currently a Professor of Computer Engineering at NIT Surat. He carries over more than 20 years of experience in Academia, R&D and Secure ICT Infrastructure Design. Prof. Dhiren has authored a book “Information Security: Theory & Practice” published by Prentice Hall of India (PHI) in 2008. He chaired IFIP Trust Management VI conference in 2012. He has been a Visiting Professor at University of Denver Colorado USA (2014) and at IIT Gandhinagar India (2009-11).

Madhura Parikh

Madhura Parikh did her B.Tech in Computer Engineering from NIT Surat (2009-13) and her Masters in Computer Science from The University of Texas at Austin (2013-14). Her interests are in Computer Systems, Machine Learning and Artificial Intelligence. She is also an open source enthusiast.

Reema Patel

Reema Patel, Ph.D in Computer Engineering Department, NIT Surat. She received her B.E., and M.Tech., degrees in Computer Engineering from NIT Surat, in 2006 and 2010 respectively. Her main areas of research interests are Formal Verification, Modeling and Analysis Methods for Stochastic Systems and Information Security.
LFTL: A multi-threaded FTL for a Parallel IO Flash Card under Linux

Srimugunthan\textsuperscript{1}, K. Gopinath\textsuperscript{2} and Giridhar Appaji Nag Yasa\textsuperscript{3}

\textsuperscript{1} Indian Institute of Science, Email: srimugunth@csa.iisc.ernet.in
\textsuperscript{2} Indian Institute of Science, Email: gopi@csa.iisc.ernet.in
\textsuperscript{3} NetApp, Email: giridhar@netapp.com

New PCI-e flash cards and SSDs supporting over 100,000 IOPs are now available, with several use cases in the design of a high performance storage system. By using an array of flash chips, arranged in multiple banks, large capacities are achieved. Such multi-banked architecture allow parallel read, write and erase operations. In a raw PCI-e flash card, such parallelism is directly available to the software layer. In addition, the devices have restrictions such as, pages within a block can only be written sequentially. The devices also have larger minimum write sizes (>4KB). Current flash translation layers (FTLs) in Linux are not well suited for such devices due to the high device speeds, architectural restrictions as well as other factors such as high lock contention.

We present a FTL for Linux that takes into account the hardware restrictions, that also exploits the parallelism to achieve high speeds. We also consider leveraging the parallelism for garbage collection by scheduling the garbage collection activities on idle banks. We propose and evaluate an adaptive method to vary the amount of garbage collection according to the current I/O load on the device.

1. Introduction

Flash memory technologies include NAND flash, NOR flash, SLC/MLC flash memories, and their hybrids. Flash memory technologies, due to the cost, speed and power characteristics, can be used at different levels of the memory hierarchy.

However, flash memory has some peculiarities such as that a write is possible only on a fully erased block. Furthermore, a block can be erased and rewritten only a limited number of times. Currently, the granularity of an erase (in terms of blocks) is much bigger than the write or read (in terms of pages). As the reuse of blocks depends on the lifetimes of data, different blocks may be rewritten different number of times; hence, the reliability of a block can vary unless special wear leveling algorithms are used.

Flash memory chips were dominantly used in embedded systems for handheld devices. With recent advances, Flash has graduated from being only a low performance consumer technology to also being used in the high-performance enterprise domain. Presently, Flash memory chips are used as extended system memory, or as a PCI express card acting as a cache in a disk based storage system, or as a SSD drive completely substituting disks.

Multiple terabyte-sized SSD block devices made with multiple flash chips are presently available. New PCI-e flash cards and SSDs supporting high IOPS rate (eg. greater than 100,000) are also available. Due to limitations in scaling of the size of flash memory per chip, parallelism is an inherent feature in all of these large flash storage devices.

Flash memory scaling:

IC fabrication processes are characterized by a feature size, the minimum size that can be marked reliably during manufacturing. The feature size determines the surface area of a transistor, and hence the transistor count per unit area of silicon. As the feature size decreases, the storage capacity of a flash memory chip increases. But as we reduce the process feature size, there are problems like, the bits can not be stored reliably. Sub-32 nm flash memory scaling has been a challenge in the past. Process innovations and scaling by storing multiple bits per cell have enabled bigger flash sizes per chip. As of this writing, the minimum feature size achieved is 19nm, resulting in an 8GB flash chip storing 2 bits per cell. But storing multiple cells per chip adversely affect the endurance of the flash[1] bringing down the maximum erase cycles per block in the range of 5000 to 10,000. It has been shown that scaling by packing more bits per cell 2 degrades the write performance[10]. Also the bus interface from a single NAND chip is usually a slow asynchronous interface operating at 40MHz. So in order to achieve larger capacities and faster speeds, multiple flash memory chips are arrayed together and they are accessed simultaneously. These kind of architectures are presently used in SSDs[7].

SSD architecture and interface: An SSD package internally has its own memory (usually battery backed DRAM), a flash controller and some firmware that implements some basic wearlevelling algorithms and exposes a block interface. Hard disks can be readily substituted by SSDs, because the internal architecture of SSDs is hidden behind the block
interface. Because of the blackbox nature of SSDs, some changes have been proposed in the ATA protocol interface such as the new TRIM command, that marks a block as garbage as a hint from an upper layer such as a filesystem to the SSDs[2].

Raw flash card vs SSDs:

Though most flash is sold in the form of SSDs, there are also a few raw flash cards used in some scenarios. Such flash cards pack hundreds of gigabytes in a single PCI-e card with an SSD-like architecture but without any firmware for wearlevelling or any on-board RAM.

The flash card that we studied is made of several channels or interfaces. Several of flash chips make a bank and several of a banks make a channel or interface. It is possible to do writes or reads simultaneously across banks. The flash card have an in-built FPGA that supports DMA write size of 32KB and DMA read size of 4KB. Pages within a block can only be written sequentially one after another.

It is important to mention that we have used a raw flash card and not a SSD for our study.

Flash translation layer:

The management of raw flash can either be implemented within a device driver or other software layers above it. The layer of software between the filesystem and the device driver that does the flash management is called Flash translation layer (FTL). The main functions of the FTL are address mapping, wearlevelling and garbage collection. Working in the context of Linux, we have designed a flash translation layer for a raw flash card which exploits the parallel IO capabilities. In addition to the main functionalities our FTL also does caching of data, before it writes to the flash.

- The following are the main contributions of this work: We present a design of a Flash translation layer(FTL) under Linux, which can scale to higher speeds. Our FTL also copes up with device limitations like larger minimum write sizes by making use of buffering inside the FTL.
- We exploit the parallelism of a Flash card with respect to block allocation, garbage collection and for initial device scanning. The initial scan is time consuming for larger flash; hence exploiting parallelism here is useful.
- We give a adaptive method for varying the amount of on-going garbage collection according to the current I/O load.

Section 2 presents the background with respect to flash memories, flash filesystems, FTLs and the flash card used. Section 3 describes the design of our FTL. Section 4 presents the results. Section 5 is a discussion of some issues in our FTL and also about future directions of work and Section 6 concludes.

2. Background

Flash memory has a limited budget of erase cycles per block. For SLC flash memories it is in hundreds of thousands while for MLC flash it is in tens of thousands.

Wearlevelling is a technique, that ensures that all the blocks utilise their erase cycle budget at about the same rate. Associated with wearlevelling is garbage collection. Wearlevelling and garbage collection have three components:
- Out of place updates (referred to as Dynamic wearlevelling): A rewrite of a page is written to a different page on the flash. The old page is marked as invalid.
- To reclaim invalided pages (referred to as garbage collection): To make the invalid pages writable again, the block with invalid pages has to be erased. If the block has some valid pages, it has to be copied to another block.
- Move older unmodified data to other blocks of similar lifetimes (referred to as Static wearlevelling)

Because of the out-of-place update nature of the flash media, there is a difference between a logical address that a filesystem uses and a physical address that is used for writing to the flash. This logical address to physical address translation is provided 3 by the flash translation layer.

Note that both garbage collection and static wearlevelling incur some extra writes. The metric, write amplification, quantifies the overhead due to garbage collection and static wearlevelling. Write amplification is the ratio between the number of user writes and the total writes that happened on the flash. A good wearlevelling algorithm minimises the write amplification as much as possible.

When Garbage collection and static wearlevelling are done during idle time when there is no I/O happening on the system, the overhead is only the extra writes and erases. But, under a heavy, continuous I/O load, when the amount of free space drops low, it is necessary to invoke garbage collection as part of a user write, to create a fresh supply of free blocks. In this case garbage collection increases the latency of every write request. A policy decides when the garbage collection is to be triggered and how much space should be freed in one invocation.

Based on these goals several interesting algorithms have been proposed. The reference[9] is an excellent survey of flash memory related algorithms.

2.1 FTLs and Flash file systems

The FTL exposes a block device interface so that any disk file system can be used. The FTL does out-of-place updates on the flash by maintaining a table of logical to physical address translation entries. There are different FTLs based on how the translation table is maintained. Some well known FTLs are NFTL[3], FAST FTL[14], and DFTL[11].

Instead of using FTLs, there are flash specific filesystems like yaffs2, logfs and UBIFS that implement some or all of the FTL functionalities like wearlevelling and garbage collection in a filesystem. Popular flash filesystems are Yaffs2, logfs and UBIFS.

We next discuss the Flash card that we used.

2.2 Flash Card Architecture

The flash card, studied is of the type that is used for I/O acceleration, by caching recently used user data and
metadata in a SAN/NAS storage system. The flash card is of size 512GB.

The Flash card is made of 4 interfaces or channels, 16 banks per interface and 8 data chips + 1 parity chip per bank. Chips are SLC nand chips with 1GB size and a maximum of 100,000 erase cycles as endurance limit.

The Card has a built-in FPGA controller that provides DMA channels for reads, writes and erases. A single DMA write size is 32 KB that is striped equally among all 8 chips in a bank with the parity generated in the ninth chip. The DMA read size is 4KB and it reads the striped data across 8 chips. DMA erase makes 2MB of data across 8 chips to become all ones. For every 512 bytes, BCH checksum of 8 bytes is generated by hardware. An extra 8 bytes per 512 bytes is available for user spare data. So there is as much as 512 spare bytes per 32KB page available for storing any user metadata. The device has programmable registers through PCI interface. We have written a driver for this device that exposes a 512GB mtd device and with a write size of 32KB and an erase size of 2MB.

The device offers parallel read, write and erase support across banks. There are 32 DMA read queues, 4 DMA write queues and 4 DMA erase queues. For write and erase queues, the device driver assigns each of the four queues to each of the 4 interfaces and queues requests correspondingly.

The hardware FPGA controller services these queues in round robin fashion. For read queue, the device driver assigns 8 queues per interface. Each queue is assigned for 2 successive banks of an interface. Each queue can hold 256 requests. When a request is completed, the device responds by writing a completion descriptor in a completion queue. The completion queue is of size 1024 entries. When requests proceed in parallel, it is possible to receive completion descriptors in out of order fashion.

The driver also has a rudimentary bad block management. Bad blocks are identified initially and stored in an on-disk file in the host system. The driver during initialisation reads from this file and constructs its in memory bad block table.

2.3 Parallelism in the flash device

A flash based SSD has many levels of parallelism.

Firstly, there are many channels. Each channel is made of flash packages, each of which consists of several flash chips. Within a single flash chip, there are multiple dies and finally there are several flash planes within a single die. The flash chips are mostly connected using a 40MHz bus in our device. The operating bus speed increases, as we move from inner flash chips to outer flash packages. The flash chips can be simultaneously read to internal buffers. The transfer from these internal buffers happen at faster bus speeds. Because of the hierarchical architecture of channels, flash packages, chips, dies and planes, there are several levels of parallelism available.

At the lowest level, a feature called multiplane command allows 2 operations of the same type (read, write or erase) on planes within a die to happen concurrently. Two dies within a chip share the IO bus in an interleaved fashion. Interleaving command executes several page read, page write, block erase and multi-plane read/write/erase operations in different dies of the same chip in a pipelined fashion. At the highest level, two requests targeted to 2 different channels or flash packages can execute simultaneously and independently.

Although the parallelism is known to NAND controller inside a SSD, the amount of parallelism that is exposed to software directly is very limited. Usually, SSD parallelism is exploited indirectly, like writing in larger sizes.

In the flash card that we used, there is a hierarchy of channels, banks and chips. The hardware FPGA employs DMA read or write that stripes the data within the chips. After the driver enqueues the read/write/erase requests in the DMA queues, the hardware FPGA services these queues in a Roundrobin fashion. So the exposed parallelism is at the bank and channel levels.

We first wrote a driver for the flash card that exposes the linux-mtd interface. Over the mtd driver we made measurements writing to all the banks from multiple threads.

---

Fig. 1: Flash card hardware architecture
Our measurements showed that the read and write speeds can go more than 300MB/sec when the banks are utilised simultaneously. When the writes are targeted to banks across channels, we get more read and write speed compared to when the writes are targeted to banks within a channel.

In our design for FTL, for simplicity and generality, we assume only bank level parallelism in flash card.

3. Design of the Flash Translation Layer

We first discuss the performance bottlenecks in the present design of linux FTLs. We then discuss the design we adopted and write and read algorithms along with buffer management to make use of parallelism of the flash card. We further exploit the parallelism for garbage collection activity.

3.1 Performance Bottleneck in Linux FTLs

Current implementations of FTL in linux, like NFTL and mtdblock use the mtd_blkdevs infrastructure. The mtd_blkdevs basically registers for a new block device and a corresponding request processing function. This mtd_blkdevs infrastructure is used by any FTL which have their own implementations for reading and writing sectors from the device.

![Image](74x339 to 284x447)  
**Fig. 2: mtdblkdevs infrastructure in Linux**

It is to be noted that the NFTL and mtdblock are pseudo block devices. In Linux a block device’s request processing function is not supposed to sleep. However, the threads that call mtd apis can sleep, and so the mtd_blkdevs does the request processing from a separate kthread. As shown in Figure 2, the kernel thread receives the IO requests from the block layer’s request queue and calls NFTL or mtdblock’s write/read sector code. The actual reading/writing of the IO request is done by the NFTL or mtdblock’s write or read sector code. The kernel threads sleep when the request queue is empty. When a new request is enqueued the mtd_blkdevs request processing function wakes up the kernel thread, which does the actual processing.

The block layer provides one request queue for a registered device. The block layer’s request queue is used by the IO scheduling algorithms. The request queue is a doubly linked list and it is used for coalescing and merging the bios (block IO requests) to much larger requests. The IO scheduling algorithms of the block layer are specifically targeted for hard disks to minimise seek time. Flash memories don’t benefit from the IO scheduling optimisations of the block layer. The merging and coalescing optimisations also are not useful because, our FTL uses a list of buffers that are used to accumulate the IOs to larger sizes. In addition to these reasons, using a single request queue, means that we have to take a lock for dequeuing the IO request. The Figure 3 is the code snippet that is implemented in mtd_blkdevs.c and it shows the acquiring and releasing of request queue locks. The problem of request queue lock contention as a hindrance to high IOPS devices has been discussed in the linux community [4]. These reasons motivated us to not use the block layer’s request queue management.

In the code snippet of Fig. 3 a global device lock (dev->lock) is used assuming that requests to the flash device can only be done serially one after another. With the existing mtdblkdevs infrastructure, the maximum IO bandwidth that we were able to obtain was only up to 55MB/sec. The large capacity flash cards can process several IO requests simultaneously on several banks. Since banks are the minimal unit of parallel IO, the global device level locks should be split in to per bank locks.

We incorporated these changes in the design of our FTL and Figure 4 shows the design that we followed. In our FTL, we maintain multiple FIFO Fig. 4: mtdblkdevs code showing lock contention queues of our own and each of these queues has an associated kernel thread. The kernel threads processes I/O requests from its corresponding queue. So these multiple ‘FTL I/O kernel threads’ provide parallel I/O capability.

![Image](318x179 to 534x407)  
**Fig. 3: mtdblkdevs code showing lock contention**

The data structure corresponding to block layer’s I/O request is called bio. The entry point in the linux block I/O layer is the function bio_submit() which receives the bio. We intercept these block I/O requests and direct them to one of our FIFO queues. The intercepting of the block I/O requests is also done for RAID devices and the kernel exports an API for registering our custom make_request function.
The mtd API for read and write are synchronous API calls. The use of multiple FIFO queues and multiple kernel threads, in addition to removing the lock contention, also facilitates several IOs to be inflight simultaneously, even though the mtd read and write calls are synchronous. The per-queue kernel thread calls the read/write sector functions of our FTL.

Unlike NFTL we have exposed a 4KB block device size, but the minimum flash writesize is 32KB. So FTL uses a set of buffers and accumulates the eight consecutive logical 4K data to a single 32K buffer and finally writes it to the flash. The perbuffer size and the number of buffers in the FTL are module parameters. The per-buffer size should be atleast the flash page size. The number of buffers is only limited by the amount of RAM in the host system. Currently we have used 256 buffers and the buffer size is 32K.

Fig. 4: FTL design
3.2 Write and Read algorithm

The main data structures, that are used in our FTL, are listed below.

- **FreeBlksBitMap**: The FreeBlksBitMap indicates whether a given block is free or occupied. The FreeBlksBit-map data structure is protected by per-bank locks.
- **BufferLookUp Table**: The BufferLookUp Table stores the logical page number that is stored in a buffer. There is no lock taken for reading or writing the buffer Lookup table.
- **BlkInfo**: The blkinfo stores a bitmap indicating the valid pages in the block and the number of valid pages in the block.
- **BankInfo**: The bankinfo stores the number of free blocks and number of valid pages in the bank. We used linux atomic bitmap operations and atomic variable updates for modifying blkinfo and bankinfo.
- **MapTable**: The maptable stores 4 bytes/entry for every page in the flash. For 512GB flash and 32K page size, the map table is of 64MB size. We have used bit-locking, with a single bit protecting per-entry of the maptable. We used the linux atomic bitmap operations for the bit-locking. The space for bitmap need not be allocated separately, as we can steal one bit from the map table entry.
- **AllocBufBitmap**: The AllocBufBitmap indicates that we are trying to allocate a buffer for a logical page. It is used for mutual exclusion to prevent allocation of two buffers to the same logical page number.

The write algorithm is given in Algorithm 1. Initially we search the buffer lookup table, if the given lpn (logical page number) is present in any of the FTL cache buffers. If the lpn is available, we take the per buffer lock and again check if the buffer corresponds to lpn. This is similar to doublechecked locking programming idiom[13]. If the lpn is not in the FTL cache, then we have to allocate one of the buffers to this lpn. We set the bit in AllocBufBitmap for this lpn to indicate that we are trying to allocate a buffer for this lpn. After taking the bit-lock, it is necessary that we check again, that there is no buffer for this lpn.

**Algorithm 1 FTL write algorithm**

```
procedure FTLWRITESEC-
TOR(Logical sector number)
  lpn ← Corresponding logical page number
  bufnum ← Search bufLookUpTable for lpn
  if Search was SUCCESS then
    WRITELOCK BUF[bufnum] ;
    Confirm BUF[bufnum] corresponds to lpn
    Write to BUF[bufnum];
    WRITEUNLOCK BUF[bufnum] ;
  else
    SET BIT in AllocBufBitmap ;
    Confirm lpn is not in bufLookUpTable;
    bufnum ← Select a Empty/Full/Partially
    Full Buffer;
    WRITELOCK BUF[bufnum] ;
    change bufLookUpTable[bufnum] ;
    if BUF[bufnum] is not an Empty Buffer
    then
      newtempbuf ← malloc(buffer size)
      Swap( newtempbuf, BUF[bufnum]) ;
      Set flush = 1;
      write to BUF[bufnum];
      WRITEUNLOCK BUF[bufnum] ;
      UNSET BIT in AllocBufBitmap ;
      if flush == 1 then
        merge with flash, if BUF[bufnum] was
        partially FULL;
        ppn ← get_physical_page()
        Write newtempbuf to ppn ;
        Change Maptable ;
  end procedure
```

3.2.1 FTL Buffer management

The buffers are either empty or they could be allocated to some lpn. The buffers which are allocated to some lpn can be either fully dirty or partially dirty. We maintain 2 queues, one that stores buffer indexes for empty buffers and another that stores buffer indexes for full buffers. The buffer indexes that are not in empty or full buffer queues are partially full buffers. We attempt to first allocate an empty buffer for an lpn. If that fails then we select a full buffer. If that fails too, then we allocate a partially full buffer. We make use of the RCU based lock free queue implementation from liburcu library [5] for the maintenance of buffer index queues.

With the use of lock free queue and by reducing the granularity of locks to the smallest units, there is no single central lock contention point in our FTL.

The evicted full or a partially full buffer has to be written to flash. For a partially full buffer, we have to do a read-modify-write by merging with the existing content from flash and write to another page.

We also have a background thread similar to buffer flush daemon, that writes the buffers to the flash if it is not read or written for more than 60 secs.

3.2.2 Physical page allocation

When we flush a buffer to the flash, a physical page to write to flash is selected. Depending on the number of banks, we maintain a ‘currentwriting block’ for each bank. Within the current writing block the pages are written sequentially. The get_physical_page in Algorithm 1, selects one of the banks and returns the next page of the ‘currentwriting block’ in that bank. The selection of banks is based on the activities happening on the bank and is described below.

Finally, after the evicted buffer is written to flash, the
map table is updated.

The read algorithm is similar. It proceeds by first checking if the logical sector is in FTL cache. If not, it reads from the flash the corresponding sector.

3.3 Garbage collection

We have implemented a greedy garbage collection algorithm that acts locally within a bank. We select dirty blocks with less than a minimum threshold of valid pages. The valid page threshold depends on the amount of available free blocks in a bank. There are three garbage collection levels with different valid page thresholds for corresponding free block thresholds.

For the first level, the garbage collection algorithm only selects victim blocks with zero valid pages. For the subsequent levels the “number of valid pages in a block threshold” is increased. For the first level of garbage collection, there is only erase of victim blocks, that need to be done. For subsequent levels we need to do both copying and erase. The greater the level, the greater the copying that need to be done. Our garbage collection algorithm acts locally, in the sense that, the valid pages are copied to another block within the same bank.

Garbage collection can be performed
1. before a write request is started: Before servicing a write request, when a new block has to be allocated, we trigger the on-demand garbage collection, which creates a fresh supply of free blocks.
2. after a write request: In the case of some SSDs, garbage collection is scheduled to happen after the current write request is completed. This way, the current write request doesn’t suffer the additional garbage collection latencies. But if there are any subsequent write requests, those requests are delayed due to the garbage collection overhead.
3. as a background thread to run in idle time:

   We do the garbage collection as separate kernel threads which runs alongside the FTL I/O kernel threads. Garbage collecting threads try to select a bank for garbage collection which is not being written currently. Similarly when the get_physical_page() algorithm selects a bank, it tries to select a bank that is not garbage collected. So the garbage collection is scheduled in such a way that it doesn’t interfere much with the I/O.

3.4 Adaptive Garbage collection

Parallelism of the flash card is exploited, when the garbage collection is performed from multiple threads. Each garbage collecting thread can work on any one of the bank. With more threads active, more dirty blocks can be garbage collected. We control the amount of garbage collection performed by controlling the number of simultaneously active garbage collector threads. When the system is relatively idle, garbage collection is performed with full force, with all the garbage collector threads being active. When I/O happens, we tone down the garbage collection by reducing the number of active garbage collecting threads. During heavy I/O, almost only one garbage collector thread is active.

To implement the adaptive garbage collection, one of the garbage collecting threads take on the master role. The amount of I/O happening in the system is determined by the number of FTL I/O threads that are asleep and the number of threads that are active. So the status of the FTL I/O kthreads are monitored. Depending on the range of FTL I/O threads active, we define the range for the number of garbage collection threads that can be active. For example, when all the FTL I/O kthreads are active, the number of active garbage collector threads will be one and only the master garbage collector thread will be running. But when the I/O in the system drops and fall into a different range, the master garbage collector thread will wake up a few of other garbage collector threads. Every garbage collector thread will perform one round of garbage collection on some available banks, before they check the status of the FTL I/O kthreads.

For the current number of active I/O threads, if the number of GC threads is more than the permissible threshold, then a non-master garbage collector thread will put itself to sleep.

The master garbage collector thread makes sure that a bank with very few free blocks, is compulsorily garbage collected in the near future. This is done by flagging the bank as exclusiveGC and this will prevent the get_physical_page() algorithm from selecting the bank for writing. This way, we priorities one of the banks for garbage collection.

3.5 Checkpointing

Similar to yaffs2, our FTL writes the in-memory data structures to the flash when we unload the module and they are read back when it is again loaded. We write the main data structures like the mappable, freeblks map, the blkinfo and bankinfo data structures to the flash. The blocks that hold these metadata information are differentiated from the blocks that store the normal data by storing a flag in the out-of-band area indicating the type of block. When the module is loaded, the flash blocks have to be scanned for the checkpointed blocks. This block level scanning can be sped-up by parallelising the scanning, by reading the flash through several kernel threads during module load.

Actually we don’t scan all the blocks, but only a top few and bottom few blocks of every bank. We maintain the checkpoint blocks as a linked list on the flash, with one checkpoint block pointing to the next checkpoint block. We impose a restriction that the first checkpoint block is stored in the top few or bottom few blocks of any of the banks in the FTL. So the parallelised scanning only reads these few blocks and identifies the first checkpoint block. Since blocks in a bank are written sequentially, we are more likely to find a free block in the top or bottom few blocks. In case if none of the banks has a free block on the top or bottom few blocks, then we create a free block by moving the contents of a occupied block and write the first checkpoint block there.

So during module load time, after finding the first checkpoint block, the rest of the checkpoint blocks are found by following the linked chain.
4. Performance evaluation

In this section, we evaluate the resultant improvements of our parallelism modified algorithms. The flash card is mounted in a PCI express slot in HP workstation machine, with four core Intel i5 processors running at 3.2GHz. The system has 4GB of memory. We used the Linux kernel version 2.6.38.8 and the operating system is Ubuntu 10.04.

4.1 Number of Queues vs FTL Block device speed

We first give the measurement over the FTL exposed block device without any filesystem. Table 1 shows the measurement of read and write speed for various number of FTL queues/threads. Having a single thread servicing from a single request queue suppresses the hardware parallelism considerably. Increasing the number of FTL queues and associated threads we were able to get write speeds upto 360MB/sec. Due to caching effects, write speeds are greater than the read speed.

Table 1: Number of FTL Queues vs read-write speed

<table>
<thead>
<tr>
<th>Number of Queues</th>
<th>Read speed</th>
<th>Write speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53 MB/sec</td>
<td>45 MB/sec</td>
</tr>
<tr>
<td>2</td>
<td>99 MB/sec</td>
<td>91 MB/sec</td>
</tr>
<tr>
<td>4</td>
<td>163 MB/sec</td>
<td>170 MB/sec</td>
</tr>
<tr>
<td>8</td>
<td>210 MB/sec</td>
<td>282 MB/sec</td>
</tr>
<tr>
<td>16</td>
<td>223 MB/sec</td>
<td>364 MB/sec</td>
</tr>
</tbody>
</table>

4.2 Filesystem measurement

Table 2 gives the filesystem read and write speed. We mounted XFS filesystem over our FTL. The filesystem measurements are taken using tiobench. The results correspond to writing from 16 threads with each thread writing 1GB. We see that the existing flash filesystems perform poorly for multithreaded workloads. Yaffs2 for single thread was able to get upto 40MB/sec, but with increasing the number of threads, performance deteriorated. UBIFS was better, but still didn’t achieve the speeds, the device can support. We obtain good performance with our FTL used with XFS filesystem. We have also shown results, by using our FTL with the existing mtd_blkdevs infrastructure to receive block I/Os.

Table 2: Tiobench results

<table>
<thead>
<tr>
<th>Filesystem used</th>
<th>Write speed</th>
<th>Read speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>yaffs2</td>
<td>5 MB/sec</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>UBIFS</td>
<td>60MB/sec</td>
<td>55MB/sec</td>
</tr>
<tr>
<td>XFS + FTL using mtd_blkdevs frame</td>
<td>42MB/sec</td>
<td>54MB/sec</td>
</tr>
<tr>
<td>XFS + FTL using modified</td>
<td>297MB/sec</td>
<td>245MB/sec</td>
</tr>
</tbody>
</table>

4.3 Module load measurement

Table 3 gives the initialisation measurements for a 512 GB flash card. We need to do a scanning of a few blocks in the beginning of each bank to find the first checkpoint block. As the checkpoint blocks are maintained as a linked list on flash, after finding the first checkpoint block we follow the chain. Our FTL load time is only a few seconds. Any mechanism that requires a linear block level scan like UBI in 2.6.38.8 kernel, takes 2 minutes.

For Yaffs2, the time taken for module load depends upon the amount of data on flash. We observed 16 secs of module load time, when the flash is more than 80% full. For failure scenarios, for our FTL we need to do a page level scan of the flash. Doing a parallel scan of the pages on the banks still takes 30 minutes. A linear scan will take an inordinately long time, for a few hours.

4.4 Results for Garbage collection

The garbage collection measurements are measured for three policies:

- NPGC: Before a write is done on a bank, if the number of free blocks is less, we garbage collect the particular bank, and then subsequently do the write.
- PLLGC: Garbage collection is always performed by parallel co-running garbage collector threads. The write algorithm tries, if possible, to select a bank, which is not being garbage collected. Otherwise the write algorithm selects a random bank.
- PLLGC+Adaptive: Similar to PLLGC policy, the garbage collection is always performed by parallel co-running garbage collector threads. In addition, the number of garbage collector threads are varied according to the amount of I/O.

We use 64 FTL I/O kernel threads for the following measurements.

First, we compare the the NPGC policy with PLLGC policy. To avoid the long time to start garbage collection, we restrict the size of flash as 8GB by reducing the number of blocks per bank to 64. Our garbage collection tests were done after modifying the FTL data structures to reflect a long flash usage period. We artificially made the blocks dirty by modifying the FTL’s data structures by an IOCTL. The result of the data structure modification is shown in Figure 5 and Figure 6. The Figure 5 shows the distribution of free blocks per bank, which is roughly normally distributed. The Figure 6 shows the distribution of valid pages in a block. After the
modifications of the data structures, the garbage collection measurements were taken. The maximum number of GC thread is set to 1 for this measurement.

For this measurement, we do over-write of 1GB of data upto 16 times from a single threaded application. We show the latencies of write requests from the single application thread. The application program does a fsync after every 4K to ensure that the data reaches the FTL device. The Table 4 compares the NPGC and PLLGC policy showing the scatter-plot of write latencies, for every written 32KB, in microseconds. The x-axis in the scatter plot is the request ids and the y-axis is the write latency for that request.

![Fig. 5: Distribution of free blocks per bank. The x-axis is the bank ids and the y-axis is the number of free blocks in the bank](image)

The scatter plot shows the number of requests that cross the 2ms boundary is much more in NPGC policy because of the GC overhead. When we do the garbage collection in parallel, the GC overhead is considerably removed. It is also important to notice that we were able to garbage collect roughly the same number of blocks without incurring the additional GC overhead, by scheduling the garbage collection activity on a parallel bank. The elapsed time is also shorter in the PLLGC version. The completion of the test took 546 seconds when garbage collection is done in the write-path and only 400 seconds for PLLGC policy.

Next, we provide the measurements comparing PLLGC with PLLGC+Adaptive policy for the scenario of multithreaded applications.

<table>
<thead>
<tr>
<th></th>
<th>NPGC policy</th>
<th>PLLGC policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of blocks garbage collected</td>
<td>8312 blocks</td>
<td>8392 blocks</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>546 sec</td>
<td>400 secs</td>
</tr>
</tbody>
</table>

![Fig. 6: Distribution of valid pages in occupied blks](image)
We set the maximum number of garbage collector threads to be 8 and we did the evaluation by also reducing the number of banks on the flash to 8. We used a simulated workload with 128 application pthreads having a fixed think time. Each application pthread sleeps for 20ms after every 32K write and 10sec after every 2MB write and each pthread writes 4MB of data 16times. So in total, we write 8GB of data and we measured the average write latency as observed from each pthread.

The plot in the Figure 7 shows the normalised average latency as seen from the 128 application pthreads. The adaptive version performed better than the non-adaptive parallel garbage collector. Also the total elapsed time for the test was 217 secs for PLLGC+Adaptive policy while PLLGC policy took 251 secs to completion. This shows that by adaptively minimising the garbage collection specific read and writes, when the IO load is high, there is lesser interference and the average latency per IO request can also be minimised.

5. Discussion

Recent works in parallelism with respect to SSDs are [15], [8], [12]. Compared to these previous papers, we have used a Raw Flash Card, instead of an SSD. While FTLs in SSDs are implemented in Flash controller, in contrast, we have considered implementation of a software FTL, which is a layer above the device driver. In the context of Linux operating system, we have clearly shown the inadequacy of the present FTLs in Linux, with respect to performance.

We next discuss some weaknesses and modifications required in our present FTL implementation.

With each written page on the flash, we store some metadata in the out-of-band area(spare bytes). We store the type of block (blocks that store checkpoint data or normal data), the corresponding logical page number, and a sequence number. The sequence number is similar to the one used in yaffs2 and is used for failure scenarios, when the FTL module is unloaded incorrectly. After an incorrect unload, the FTL has to do a page-level scan of the flash. Failure handling is important in battery operated embedded devices. But there are several usecases, other than embedded systems, for the kind of Raw Flash cards, that we used. For failure handling, we also should implement the block layer barrier commands [6]. When the filesystem gives a barrier request, we have to empty the queues in our FTL and we also have to flush all of our FTL buffers to the flash. The FTL cache size is a module parameter and it is limited only by the amount of RAM in the system. So if we expect very heavy random write workload, we can load the FTL module with larger cache. It is also possible that if we can detect random IO behaviour, the FTL can adaptively increase the FTL cache size. We can also increase or decrease the per buffer size. This kind of tunable control is not possible with using the SSDs directly, but it is possible with using the raw flash card and a software FTL.

We have been liberal in the use of memory in our FTL. But it is possible to reduce the memory consumption considerably. The bits necessary for bit locking in our write algorithm can be stolen from the maptable entries. Though we have used a fully page based FTL in our implementation, it is possible to extend the purely page based FTL with DFTL[11] algorithm and reduce memory consumption of the mapping table.

There are many directions for future work. In addition to doing the garbage collection in parallel, it is also possible to do other activities like prefetching, static wear-levelling or de-duplication in parallel without affecting the write latencies.

We used a FIFO buffer replacement in our FTL, which is not ideal when there is temporal locality in the workload. One future work is to consider other locality aware cache replacement policies. It would be interesting to consider how the nonblocking buffer management schemes like [16] compare against conventional LRU schemes.

In our design of the FTL, we have avoided using the block layer’s request merging, coalescing and IO scheduling optimisations. One direction of future work is to look in to the scheduling policies, that can be used in our multi-queued design. It has been shown that by allocating blocks based on hotness of data, we can improve garbage collection efficiency and hence the long term endurance of the flash. We have to explore further if the selection of banks in which the blocks are allocated, should also be based on hotness.

6. Conclusion

In conclusion we have shown how a FTL can be designed for a multi-banked flash card. The modifications leverage the parallelism available in these devices to achieve higher performance. We show that garbage collection overhead can be considerably removed from the write path because of the available parallelism. In addition to using parallelism for greater speed, we also show that initial scanning time that is required in flash can also be reduced due to parallelism. We have also proposed an adaptive garbage collection policy, to vary the amount of garbage collection, according the IO load,
by controlling the number of active garbage collector threads.

7. Acknowledgements

Thanks to authors of open source flash filesystems (Yaffs2, UBIFS and logfs), for mailing list discussions. Special thanks to Joern Engel.

References


About the Author

Srimugunthan completed his Masters from Department of Computer Science and Automation, Indian Institute of Science, Bangalore. He holds a Bachelor’s degree in Computer Science form Madras Institute of Technology, Chennai. His research interests are in the areas of Storage systems, Big Data and Machine Learning. He is currently working with Akamai Technologies.

K. Gopinath is a Professor in Department of Computer Science and Automation, Indian Institute of Science, Bangalore. His research interests are in the areas of Operating Systems, Storage Systems, Systems Security, Verification. He is an associate editor of ACM transactions on storage.”

Giridhar Appaji Nag Yasa holds A Bachelors Degree in Computer Science from Indian Institute of Technology, Guwahati. His research interests are in the areas of Storage systems, Scalable software systems architectures and Network and Internet Protocols. He was part of Advanced Technology Group in NetApp. He is also an official Debian Developer.
Towards An Executable Declarative Specification of Access Control

N. V. Narendra Kumar¹ and R. K. Shyamasundar²

¹ Tata Institute of Fundamental Research. E-mail: n.kumar@tifr.res.in
² Indian Institute of Technology Bombay. E-mail: rkss@cse.iitb.ac.in

Logic plays a vital role in the design and analysis of security policies and models. Several security languages exist in the literature based on either Horn-clauses fragment of the first-order logic or some form of modal logics. SACL is a security language influenced by the SPKI/SDSI PKI, with several features designed to suit specification and verification of authorization in large, open, distributed systems. Although several logic based semantics exist for the naming scheme of SPKI/SDSI, there is not one that completely captures its features like authorization tags and delegation. In this paper, we provide a Logic Programming (henceforth referred to as LP for short) model for SACL which completely captures its features including groups, structure of authorization tags and delegation. One of the unique features of our approach is that we provide detailed algorithms for tag intersection based on the structure of the input tags. Naturally, the approach not only leads to declarative specification but also executional specifications. Thus, the approach leads to a clear understanding of concepts like delegation, groups, roles etc, and can be used to verify and validate properties of SACL specifications. We compare our approach with other related approaches. Using the vast body of knowledge available on LP, we can enhance the language with additional features while keeping it tractable.

Further, we show how lattice-based information flow can be specified in SACL. Such a specification allows us to specify fine-grained information flow policies which play vital role in a variety of applications ranging from secure execution of third party software in operating systems to limiting unwanted information disclosures in heterogeneous information systems where data comes from a variety of sources each with their own policies. This model subsumes several well studied security models like Role-Based Access Control, the Bell-LaPadula model, the Chinese-Wall model and the Biba integrity model, since all these models are special cases of the lattice-based flow model. Thus, SACL provides a uniform framework for specifying and analyzing heterogenous policies made up of several policy concepts.

1. Introduction

Access control is a very important aspect of systems security. In distributed systems the problem is referred to as trust management [7]. PolicyMaker[8], KeyNote[6] and SPKI/SDSI[14], [29], [15] are prominent trust management languages. SD3[19], Binder[13], SecPAL[4], Delegation Logic[22] and RT[25] are examples of logic-based distributed authorization languages. Abadi[2] presents an interesting survey and discussion of logical foundations for access control and their applications to programming security policies.

SPKI/SDSI has several novel features like linked local name spaces and delegation of authorization. The naming scheme of SPKI/SDSI has generated a lot of interest in the research community and several semantics were proposed for a clear understanding. Clarke et al.[10] presented a semantics for names by considering the name certificates as rewrite rules. They extend their semantics to also consider authorization certificates and present an algorithm for discovering a certificate chain that proves the authorization of a client to access a resource.

Howell and Kotz[18] extend the ABLP logic[21], [3] with a restricted form of delegation and provide a semantics for the SPKI/SDSI. Abadi[1] introduced a logic based semantics for the SPKI/SDSI. Halpern and van der Mayden[16] developed the Logic of Local Name Containment and also extended the logic to deal with authorization [17]. Li and Mitchell[24] provide several semantics for the SPKI/SDSI naming scheme.
based on string rewriting, set theory, LP and FOL. They extend their FOL semantics to handle authorization. They also showed the advantages of their semantics over several others.

In [27], we introduced the Secure Access Control Language (SACL) influenced heavily by SPKI/SDSI PKI, formally modelled SACL specifications as Finite State Transition Systems and have provided model checking algorithms for verifying/validating several properties of SACL. We also presented an approach to enforce such specifications using Security Automata [32].

In this paper, we give a LP model for SACL specifications and show the advantages of our semantics over Li’s FOL semantics [24] for the SPKI/SDSI. Our model has the advantages that it is declarative and leads naturally to an execution model. Further, we can analyze the policy for required properties. We also show how the naming scheme of SACL can be interpreted as inducing a hierarchy over the name space. This hierarchy can then be used to model several policy concepts and security models. In particular, this allows us to model information flow policies which lead to fine grained access control mechanisms in operating systems. This enables a uniform framework for combining mandatory and discretionary policies.

The rest of the paper is organized as follows: in Section 2, we give the LP model of SACL and discuss its properties. In Section 3, we show how several well studied security models and policy concepts can be modelled in SACL. We provide concluding remarks in Section 4.

2. LP Model of SACL

In [27], we introduced the secure access control language SACL, with several features designed to suit authorization specifications in large, open, distributed systems. In this section, we give a Horn-clause model for SACL and discuss its properties.

2.1 Overview of SACL

In this section, we provide a brief overview of the language SACL. SACL is derived from the SPKI/SDSI PKI, by treating a set of SPKI/SDSI certs as a policy specification for access control. The design goal of SACL was to extend the features of SPKI/SDSI for use in a flexible access control specification language (flexi-ACL [28]). For the purposes of this paper, we consider only that fragment of SACL which includes the name and auth certs as defined in SPKI/SDSI.

The basic components of SACL are:
- set of principals \( U \)
- set of name identifiers \( A \), used to construct names
- set of names \( N = U^*A^* \), constructed from principals using zero or more name identifiers
- set of authorization tags \( T \), used to represent sets of authorizations

Every principal in SACL has its own local name space. These name spaces can be linked using extended names. The interpretation of names is a group of principals who are its members. We interpret auth tag as a set of permissions. We say that an auth tag is more restrictive than another if the interpretation of the first tag is a subset of the interpretation of the second tag. Authorization being requested can also be encoded as a tag. We will respect the request if the requested auth tag is more restrictive than the tag permitted for that principal.

The basic units of an SACL specification are name certs and auth certs.
- Name certs allow a principal to bind principals’ names to a local name in its name space. This binding can be extended to a binding of principals to arbitrary names in a straightforward way [10], [24]
- Auth certs allow a principal to authorize the members of a name to specified operations on a resource and to possibly redelegate this authorization.

Each certificate has a validity field specifying the conditions under which the certificate is valid (typically range of dates). For the purposes of this paper, we assume that we are dealing with currently valid certificates, so we ignore the validity field of the certificates in this discussion. We now describe the format of the certificates.

A name cert in SACL is of the form \( (u, a, n) \) where,
- principal \( u \) is the issuer of this certificate
- \( (u, a) \) is the name that is defined by this certificate and
- this certificate states that all members of the name \( n \) are also members of the name \( (u, a) \).

An auth cert in SACL has the form \( (u, a, d, t) \) where,
- principal \( u \) is the issuer of this certificate
- \( t \) denotes the authorization being granted by \( u \) to members of the name \( n \)
- if \( d = 0 \), it means that principals receiving this authorization are not allowed to redelegate it. If \( d = 1 \), it means that principals receiving this authorization are allowed to delegate it.

An access control specification in SACL consists of a set of name and auth certs. Given an SACL specification and an access request we want to decide whether this access should be permitted. To compute the answer to this, we define the notion of certificate composition which is the only rule of inference in SACL.

1) Given name certs \( (u, a, (u_1, b_1, b_2, ..., b_m)) \) and \( (u, b, (u_2, c_1, c_2, ..., c_n)) \) we can compose them to obtain the name cert \( (u, a, (u_1, c_1, c_2, ..., c_n, b_1, b_2, ..., b_m)) \). If we can derive the name cert \( (u, a, u_1) \) from the given set of certificates using composition then we say that \( u_2 \) is a member of \( (u, a) \)
2) Given an auth cert \( (u, a, (u_1, b_1, b_2, ..., b_m), d, t) \) and a name cert \( (u_1, b, (u_2, c_1, c_2, ..., c_n)) \) we can compose them to obtain the auth cert \( (u, (u_2, c_1, c_2, ..., c_n, b_1, b_2, ..., b_m), d, t) \)
3) Given auth certs \( (u, u_1, 1, t) \) and \( (u, n, d, (u, t)) \) we can compose them to obtain the auth cert \( (u, n, d, TT(t, t)) \), where \( TT \) is the tag intersection operator i.e. the set of permissions authorized by both the tags

If we can derive the auth cert \( (u, u_1, d, t) \) from the given set of certificates, where \( u \) is the owner of the resource \( t \) then
we say that $u_i$ can access $t$.

We now present an example SACL specification which will be used as a running example for the rest of this section.

Consider an SACL specification $C$, which consists of the following set of currently valid certificates:

- $C_1 = (u_1, a, u_2, b)$
- $C_2 = (u_2, b, u_3)$
- $C_3 = (u_1, a, (u_1, a, b))$
- $C_4 = (u_1, a, u_4)$
- $C_5 = (u_3, c, u_5)$
- $C_6 = (self, (u_1, a), 1, t_1)$
- $C_7 = (u_3, (u_4, c), 0, t_2)$

where $t_1 = (tag\ (purchase\ (*\ range\ le\ 200)\ (*\ set\ item1, item2, item3))$ and $t_2 = (tag\ (purchase\ (*\ range\ le\ 150)\ (*\ set\ item1, item2)))$.

Given this access specification, should we permit $u_i$ access to $t_i = (tag\ (purchase\ 100\ item2))$?

1. We compose $C_i$ with $C_i$ to obtain $C'_i = (u_i, a, (u_i, a, b))$ and compose $C'_i$ with $C'_i$ to obtain $C''_i = (u_i, a, u_j)$.
2. We compose $C'_i$ with $C'_i$ to obtain $C'_i = (self, (u_i, a, b))$. Composing $C'_i$ with $C'_i$ yields $C'_i = (self, (u_i, a, b))$.
3. Composing $C'_i$ successively with $C'_i$ and $C'_i$ yields $C''_i = (self, (u_i, a, b))$.
4. We compose $C'_i$ with $C'_i$ to obtain $C'_i = (u_i, a, u_j, 0, t_i)$.
5. Finally, we compose $C'_i$ with $C'_i$ to obtain $C''_i = (self, (u_i, a, b), 0, t_i)$. Notice that the intersection of tags $t_i$ and $t_j$ yields $t_j$ since $t_j$ is strictly a subset of $t_i$.

Since the tag $t_j$ is more restricted than $t_i$, we can conclude from $C'_i$ that $u_i$ should be allowed to access to $t_j$, i.e., $u_i$ should be allowed to purchase item2 worth 100.

For a more detailed exposition of SACL, we refer the interested reader to [27].

2.2 Modelling basic elements in LP

In this section, we show how the basic elements of SACL are modelled using Horn clauses.

- principal $u$ is encoded using the unary predicate $isPrincipal(u)$
- name identifier $a$ is encoded using the unary predicate $isNameId(a)$
- names are encoded using lists. $[u]$ corresponds to the principal $u$, $[u, a]$ denotes the local name $(u, a)$, and $[u, a, a_2, \ldots, a_n]$ corresponds to the extended name $(u, a_1, a_2, \ldots, a_n)$
- authorization tags are encoded as lists. $[tag, ftp, ftp://abc.edu, [*, set, r, w]]$ is an example of an authorization tag that says that the holder can read and write from the site ftp://abc.edu.
- name cert $(u, a, n)$ is encoded using the binary predicate $name_def([u, a, n], s)$, where $s$ denotes the cert number and can be used as a succinct reference to the cert.
- acl entry $(self, n, d, b)$ is encoded using the binary predicate $acl_entry([self, n, d, b], s)$, where $s$ denotes the acl entry number.
- auth cert $(u, n, d, l)$ is encoded using the binary predicate $auth([u, n, d, l], s)$, where $s$ denotes the auth cert number.

Authorization tags in SACL are carefully designed to support a variety of applications. The structure of auth tags is as follows:

- A tag can either be $(tag\ (*)$ or $(tag\ expr)$. Auth tag $(tag\ (*)$ denotes the set of all permissions and must be used with a lot of care in delegations
- $(tag\ expr)$ can be one of byte-string, simple-expr, set-expr, prefix-expr or range-expr
- A byte-string is any word over an underlying alphabet $\Sigma$. A byte-string represents atomic permissions that can be further broken down and form the basis from which complex tags are constructed.
- simple-expr is a byte-string followed by zero or more tag-expr. simple-expr is used to represent permissions over the authorizations.
- set-expr has the structure $*\ set$ followed by zero or more tag-expr. set-expr is used to group permissions/resources together for succinct presentation.
- prefix-expr has the structure $*\ prefix$ followed by a byte-string. prefix-expr represents the set of all words which have the byte-string as their prefix. This can be used to succinctly represent access to hierarchically arranged resources such as directory structures.
- range-expr has the structure $*\ range$ followed by the ordering, $lower-limit$ and the upper-limit. ordering specifies the type of data; possibilities are alpha, numeric, binary, time and date.

Now, we present some examples for better understanding of the tags.

- Tag $(tag\ (ftp\ xyz.com\ abc))$ says that subject can do ftp to the host xyz.com as user $abc$
- Tag $(tag\ (http\ http://abc.com/xyz))$ gives subject the permission to access the web page at the given URL. To give access to the entire tree we can use the tag $(tag\ (ftp\ prefix\ http://abc.com/xyz))$.
- Tag $(tag\ (pkpfs\ //ftp.abc.net\ xyz/spki\ text\ (*\ set\ read\ write)))$ gives subject the permission to read and write the file spki.txt in the public key protected file system.
- Tag $(tag\ (purchase\ (*\ range\ le\ amount)\ (*\ set\ items)))$ gives the user permission to place a purchase order over the set of items $items$ and for amount no more than $amount$.

For example, certificates presented in the running example correspond to the following facts:

- name_def([u_1, a, u_2], [1]), name_def([u_2, b, [u_3]], [2]), name_def([u_1, a, [u_1, a, b]], [3]), name_def([u_1, a, [u_4]], [4]), name_def([u_3, c, [u_5]], [5]), acl_entry([self, [u_1, a, 1, t_1]], [6]), auth([u_3, [u_3, c], 0, t_2], [7]), where $t_1 = [tag\ (purchase, \ [*\ range, le, 200], [*\ set, item1, item2, item3])]$ and $t_2 = [tag\ (purchase, \ [*\ range, le, 150], [*\ set, item1, item2])]$.

The above predicates form the database used for making access decisions.

2.3 Name closure computation

In this section, we describe rules for computing the name closure. We have a binary predicate name_closure($[u, a, u_1, pf]$) which represents that the name cert $(u, a, u)$ is in the closure and $pf$ denotes the sequence of certificates whose
composition leads to \((a, a, u, u)\).

\[
\text{name_closure treatment}(K, A, K1, Pf) \leftarrow
\text{name_def treatment}(K, A, K1, S), Pf = S.
\]

\[
\text{name_closure treatment}(K, A, K1, Pf) \leftarrow
\text{name_def treatment}(\{K, A, N\}, S), \text{length}(N, L), \text{nth1}(1, N, K2), \text{name_closure treatment}(K2, A, K1, Pf1), \text{append}(S, Pf1, Pf).
\]

The first rule above states that if \((a, a, u, u)\) is a name cert then we can conclude that \((a, a, u)\) is in the closure. The second rule states that if \((a, a, u, n)\) is a name cert and if \((a, n, u)\) is in the closure then \((a, a, u)\) is in the closure. The third rule tells us how to compute the members of an extended name by successively computing the members of each local name appearing in it.

Given the database in the previous section and the rules above, we can make the following conclusions:

- \(\text{name_closure treatment}(a, a, u, 1)\) by rule 1 using \text{name_def treatment}(a, a, u), [1]
- \(\text{name_closure treatment}(a, b, u, 2)\) by rule 1 using \text{name_def treatment}(a, b, u), [2]
- \(\text{name_closure treatment}(a, [a, b], 3, 1, 2)\) by rule 2 using \text{name_def treatment}(a, [a, b], [3, 1, 2]), [3] and the previous conclusion

2.4 Modeling delegation and authorization closure

We have a binary predicate \text{auth_closure treatment}(a, d, t, pf)\) to replace the name appearing in ACL entries and auth certs by its members.

\[
\text{auth_closure treatment}(K, K1, D, T, Pf) \leftarrow
\text{acl_entry treatment}(K, K1, D, T, S), \text{auth treatment}(K, K1, D, T, S), \text{length}(N, L), \text{nth1}(1, N, K2), L = \text{last}(K2, Pf), \text{append}(S, Pf1, Pf).
\]

The above rules say that if there is an acl entry or an auth cert \((a, n, d, t)\) and if \(a\) is a member of \(n\), then we can conclude that \((a, n, d, t)\) is a valid authorization. By applying the above rule we can conclude:

- \(\text{auth_closure treatment}(\text{self}, u3, 1, t1), [6, 3, 1, 2]\)
- \(\text{using acl_entry treatment}(\text{self}, \{a1, a2\}, 1, t1), [6]\)
- \(\text{and name_closure treatment}(\{a1, a2\}, 3, 1, 2)\)
- \(\text{auth_closure treatment}(u3, u5, 0, t2), [7, 5]\)
- \(\text{using auth treatment}(u3, \{a, c\}, 0, t2), [7]\)
- \(\text{and name_closure treatment}(u3, c, u5), [5]\)

We have a ternary predicate \text{grants treatment}(u1, u2, d, t, \phi, \phi)\) to compute the authorizations of principals either directly or transitively obtained (by a chain of delegations): \(\phi\) guards us against going around in loops by using the same auth certs repetitively and \(\phi\) gives us the proof of the authorization as a sequence of certs used in the derivation.

\[
\text{grants treatment}(K, K1, D, T, Pf) \leftarrow
\text{auth_closure treatment}(K, K1, D, T, Pf), \text{nth1}(1, Pf, C), \text{not}(\text{member}(C, Pf)).
\]

The first rule above looks for a direct authorization. The second rule says that if \(u\) gives to \(u\) the authorization \(t\) with permission to further delegate it, and if \(u\) grants \(t\) to \(a\), and if tags \(t\) and \(t\) combine to result in \(t\), then we say that \(u\) grants \(t\) to \(a\). Second rule captures the way authorization flows. The predicate \text{grants treatment} computes the authorization closure of a given set of certificates.

From the above rules we can conclude:

- \(\text{grants treatment}(u3, u5, 0, t2, [\{7, 5\}])\) from rule 1 using \text{auth_closure treatment}(u3, u5, 0, t2), [7, 5]
- \(\text{grants treatment}(\text{self}, u3, 0, t2, \{\{6, 3, 1, 2, 7, 5\}\})\) from rule 2 using \text{auth_closure treatment}(\text{self}, u3, 1, t1), [6, 3, 1, 2] and the previous conclusion since, tags \(t1\) and \(t2\) combine to yield \(t2\)

2.5 Tag intersection algorithm

Algorithm for combining tags depends on the structure of the tags being combined, and is defined inductively over the structure of the input tags. Table 1 describes the structure of the tag resulting due to intersection of tags, where title of row gives the structure of the first tag and title of column gives the structure of the second tag.

We have a ternary predicate \text{tag_combines treatment}(t1, t2, t)\) which denotes that tags \(t1\) and \(t2\) combine to yield \(t\). Now we present the rules used for computing \text{tag_combines treatment}.

\[
\text{tag_combines treatment}(\{\text{self}\}, X, X).
\]

The above rules state that the result of combining any tag \(X\) with \((\text{tag} (\text{self})\) is \(X\).

\[
\text{tag_combines treatment}(X, X, X) \leftarrow \text{atom}(X).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

\[
\text{tag_combines treatment}(X, \{\text{set}\{Y\}, X\} \leftarrow \text{atom}(X), \text{member}(X, Y).
\]

The above set of rules tell us how to combine tags when one of the tags is a byte-string. The first rule tells that two byte-strings can be combined iff they are equal. Second and third rules state that a byte-string \(X\) intersects with a set \(Y\) iff \(X\) is a member of \(Y\). Fourth and fifth rules express that byte-string \(X\) combines with a prefix form \(Y\) iff \(Y\) is a prefix of \(X\). The last two rules say that the byte-string \(X\) combines
with a range form Z considered under the ordering Y iff X is in the range Z under ordering Y using the predicate \textit{range}\_check. For lack of space we do not give rules corresponding to the predicate \textit{range}\_check.

\texttt{tag\_combines}(X,Y,R) :-
  \texttt{simple\_expr}(X),\texttt{simple\_expr}(Y),
  \texttt{lists\_combines}(X,Y,R).
\texttt{tag\_combines}(X,[\ast,\texttt{set}|Y],Z1) :-
  \texttt{simple\_expr}(X),\texttt{set\_combines}(Y,X,Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).
\texttt{tag\_combines}([\ast,\texttt{set}|Y],X,Z1) :-
  \texttt{simple\_expr}(X),\texttt{set\_combines}(Y,X,Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).

These rules tell us how to combine tags when one of the tags is a simple-exp. The first rule tells that for combining two simple expressions X and Y we need to combine their elements position-wise. The predicate \textit{lists\_combines} does this computation. For lack of space we do not give rules for \textit{lists\_combines}. Remaining rules state that for combining the simple-exp X with a set Y, we combine X with each element of the set Y and collect the results in a set. We have a predicate \textit{set\_combines} that combines elements of a set with a given element. Note that a simple-exp does not combine with a preffix-exp or a range-exp.

\texttt{tag\_combines}([\ast,\texttt{set}|X],[\ast,\texttt{set}|Y],Z1) :-
  \texttt{set\_combines}(X,[\ast,\texttt{prefix}|Y],Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).
\texttt{tag\_combines}([\ast,\texttt{prefix}|X],[\ast,\texttt{set}|Y],Z1) :-
  \texttt{set\_combines}(X,[\ast,\texttt{prefix}|Y],Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).
\texttt{tag\_combines}([\ast,\texttt{prefix}|X],[\ast,\texttt{prefix}|Y],Z1) :-
  \texttt{set\_combines}(X,[\ast,\texttt{prefix}|Y],Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).
\texttt{tag\_combines}([\ast,\texttt{set}|X],[\ast,\texttt{set}|Y],Z1) :-
  \texttt{set\_combines}(X,[\ast,\texttt{set}|Y],Z),
  \texttt{append}([\ast],[\texttt{set}],Z,Z1).

These rules tell us how to combine tags when one of the tags is a set-exp. The first rule says that for combining sets X and Y, we should combine each element of X with every element of Y and collect the result in a set. The predicate \textit{sets\_combines} does this computation. The remaining rules express that for combining a prefix or a range-exp with a set X we need to combine the expressions with each element of the set and collect the result in a set.

\texttt{tag\_combines}(*,[\ast,\texttt{prefix},X],[\ast,\texttt{prefix},Y],[\ast,\texttt{prefix},Z]) :-
  \texttt{string\_length}(X,L),\texttt{string\_length}(Y,M),
  \texttt{L>M} \rightarrow \texttt{(string\_concat}(X,A,X),Z=X),
  \texttt{M>L} \rightarrow \texttt{(string\_concat}(X,B,Y),Z=Y),
  X=\rightarrow Z=X.
\texttt{tag\_combines}(*,range,X|Ai,*\texttt{range},X|Bi,Z1) :-
  \texttt{range\_combines}(X,A,B,C),
  \texttt{append}([\ast,\texttt{range},X],C,Z1).

The first rule above says that the result of combining two prefix expressions X and Y is the longer of them provided that the shorter expression is a prefix of the longer expression. The second rule expresses that we can combine two range forms Y and Z iff they are considered under the same ordering X. The predicate \textit{range\_combines} (X, A, B, C) computes the range C which is the result of combining the ranges A and B when considered under the ordering X. For lack of space we do not give the rules for \textit{range\_combines}.

The process of tag combination, for the tags \(t_1\) and \(t_2\) in the running example, is illustrated in Fig. 1.

We do not see any meaningful situations in practice where it will be necessary to intersect a prefix expression with a range expression. So make the simplifying assumption that a range expression and a prefix expression fail to combine. Our tag combination algorithm is sound.

### 2.6 Access decisions

We have a ternary predicate \textit{access\_request} \((u, t, p)\) which expresses that principal \(u\) is authorized to access \(t\) and a proof is given by the sequence of certs \(p\).

\texttt{access\_request}(X,T,P) :-
  \texttt{grants}([self,X,Tl],[\texttt{set},T,P]),
  \texttt{tag\_combines}(Tl,Tl2),T=Tl2.

The rule above says that principal \(u\) is authorized to access \(t\) with proof \(p\) if, \((self, u, d, t, t)\) is in the authorization closure with proof \(p\) and the tags \(t\) and \(t\) combine to yield \(t\). The tag combination means that the requested access is a subset of the available set of permissions.

The request in the example will be encoded as \textit{access\_request} \((u, t, X)\). We can conclude \textit{access\_request} \((u, t, t, [6, 3, 1, 2, 7, 5])\) from the above rule using \texttt{grants} \(([self, u, 0, t, t], \texttt{set}, [6, 3, 1, 2, 7, 5])\) and the fact that tags \(t\) and \(t\) combine to yield \(t\). Thus the query access request \((u, t, X)\) returns \texttt{yes} with \(X = [6, 3, 1, 2, 7, 5]\). Therefore we conclude that \(u_5\) must be granted access to \(t_3\).

### Table 1: Table showing the structure of the tag resulting from intersection of two tags

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>*</th>
<th>string</th>
<th>simple</th>
<th>set</th>
<th>prefix</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>string</td>
<td>*</td>
<td>string</td>
<td>simple</td>
<td>FAIL</td>
<td>string</td>
<td>FAIL</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
<td>simple</td>
<td>FAIL</td>
<td>string</td>
<td>FAIL</td>
<td>string</td>
<td>FAIL</td>
</tr>
<tr>
<td>simple</td>
<td>simple</td>
<td>string</td>
<td>FAIL</td>
<td>set</td>
<td>FAIL</td>
<td>range</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>string</td>
<td>FAIL</td>
<td>set</td>
<td>set</td>
<td>FAIL</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>prefix</td>
<td>string</td>
<td>FAIL</td>
<td>set</td>
<td>prefix</td>
<td>FAIL</td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>range</td>
<td>range</td>
<td>string</td>
<td>FAIL</td>
<td>set</td>
<td>FAIL</td>
<td>range</td>
<td></td>
</tr>
</tbody>
</table>

---

N. V. Narendra Kumar, et. al. |  | R4 : 39

CSI Journal of Computing | Vol. 2 • No. 4, 2015

---
2.7 Prototype implementation

We developed a prototype tool for specifying and verifying authorizations in SACL based on the ideas presented above. We used SWI Prolog\(^1\) for our implementation. We hard coded the name and auth certs into the program. We implemented an initialization phase which checks currently valid certs and adds them to the database together with a cert number. We also implemented our algorithm for tag intersection.

Our tool can be used for checking various properties including:

- Is \( k \) a member of \((k, a, k')\)? Can be answered by querying \( \text{name closure} (k, a, k')\), if it is true then our tool returns a proof of this fact in \( P_f \). In the second component we can specify a list of certificates not to be included in the derivation
- What are the permissions authorized by both the tags \( t_i \) and \( t_j \)? Can be answered by querying \( \text{tag combines} (t_i, t_j, T) \), it returns fail if there are no common permissions or returns the common permissions in \( T \)
- What permissions does principal \( k \) have? We implemented a predicate \( \text{permissions} (k) \) which computes the set of permissions that \( k \) has as the union of \( \text{delegable} \_\text{permissions}(k) \) and \( \text{use only permissions}(k) \)

![Fig. 1. Figure illustrating how tags are combined position-wise](image)

![Fig. 2. Figure illustrating (a) name closure, (b) tag intersection, (c) permissions of a user and (d) decision of an access request](image)

- Is principal \( k \) allowed to access \( t \)? We simply query \( \text{access request} (k, t, P_f) \) and we get the proof of authorization in \( P_f \) if this request is permitted.

We hard coded the certs corresponding to running example in the program and queried the properties mentioned above. We present snapshots of the results in Fig. 2.

As a possible future extension we plan to create a front end where user can input certs which will be parsed before they get added to the database. Another possibility is to specify a file in which the certs are coded.

2.8 Related work

Li et al. [24], have presented a spectrum of semantics for the SPKI/SDSI's naming scheme based on set theory, LP, string rewriting and FOL. They have shown relationships among the various semantics and have argued their semantics provide advantages over the semantics due to Abadi [1] and van der Mayden [16]. They have also extended their FOL semantics to include SPKI/SDSI auth certs. In the following, we provide a detailed comparison of our approach to the approach of Li et al.

\( [\text{tag, [purchase, [*range,le,200], [*set,item1,item2,item3]]}] \) 
\( [\text{tag, [purchase, [*range,le,150], [*set,item1,item2]]}] \) 
\( [\text{tag, [purchase, [*range,le,150], [*set,item1,item2]]}] \) 
\( [\text{tag, [purchase, [*range,le,150], [*set,item1,item2]]}] \) 

In the approach of Li et al., predicate $m(K, A, K')$ is used to denote that $K'$ is a member of $KA$. They use the following rules for computing $m$.

\[ m(K, A, K') \] for every cert $(K, A, K')$

\[ m(K, A, K') \] for every cert $(K, A, (K_1, A_1))$

\[ m(K, A, K') \] for every cert $(K, A, (K_1, A_1, K_2), (K_2, A_2, K_3), \ldots, (K_n, A_n, K'))$

Given a set of name certs $C$, let $LP_1(C)$ and $LP_2(C)$ denote their program and our program respectively for computing members of names. The semantics of a logic program is defined by its least Herbrand model. Whenever an atom $A$ is in the least Herbrand model of program $L$ we write $L \models A$. Our semantics and theirs are equivalent in the following sense.

**Proposition 1.** Given a set of name certs $C$, principals $K$ and $K'$, and name identifier $A$, $LP_1(C) \models m(K, A, K')$ iff there exists a $Pf$ such that $LP_2(C) \models name\_closure((K, A, K'); Pf)$.

The above proposition is an easy consequence of the following observations: the first rule of our program and theirs compute direct associations. Second rule of our program replaces a name by its definition and the third rule then takes care of chaining the intermediate results together to find the final result. In fact, first part of our third rule generates the body of their second rule and second part generates the body of their third rule.

Li et al. have extended their model to include SPKI auth certs, wherein they treat auth tags and validity conditions as constraint domains. They have a predicate $g(k, t, k')$ which represents that $k$ gives authority $t$ to $k'$ (either directly or transitively). They define a macro $contains$ in terms of predicate $m$, which is used to capture the fact that $k$ is a member of $n$. Their program (constraint Datalog) for computing $g$ is as follows:

\[ contains[KA][K'] \] :- \ contains[N][K'] \].

\[ g(K, t, K') \] :- \ contains[N][K'], \ satisfies(t, T).

\[ g(K, t, K') \] :- \ contains[N][K'], \ satisfies(t, T).

\[ g(K, t, K') \] :- \ contains[N][K'], g(k', t, K'), \ satisfies(t, T).

where the predicate $satisfies$ captures constraint satisfaction. In the above program, the following points are to be noted:

- predicate $g$ of Li, talks about individual authorizations, whereas as our grants predicate can handle sets of authorizations. So we get a succinct representation of the authorizations that a principal has
- the predicate grants explicitly captures the delegation bit, which allows us to easily express the permissions a principal can and cannot further delegate. Whereas using the predicate $g$ it becomes a complicated query

Another advantage of our model is that we handle both auth tags and validity purely syntactically (except for evaluating the range expressions at which point using semantics is unavoidable), thus maintaining the free interpretation. We have also described algorithms for computing the intersection of auth tags based on their structure.

### 3. Capturing other security models in SACL

In this section, we show how the naming scheme of SACL induces a hierarchy which can be used to model information flow policies. In this section, we limit our attention to name certs of the form $(u_a, a_u)$ or $(u_a, a_u, a_u)$ only i.e., we do not consider extended names. The applications we consider below justify this limitation.

Given a set of name certs, consider the induced directed graph whose vertices are either local names or principals appearing in certs; and there is an edge from $(u_a, a_u)$ to $u_i$ if we have the cert $(u_a, a_u, a_i)$ and an edge from $(u_a, a_u)$ to $(u_{i_1}, a_j)$ if we have the cert $(u_a, (u_{i_1}, a_j))$. This graph can be interpreted as a dominance relation on names: name $n_1$ dominates name $n_2$, denoted $n_1 > n_2$, iff there is a path from $n_1$ to $n_2$. From the semantics of SPKI we can conclude the following whenever $n_1 > n_2$:

- all the members of $n_2$ are members of $n_1$
- all the permissions authorized for $n_1$ are authorized for $n_2$

In particular, when relation $>$ forms a lattice it becomes very useful for modeling several important applications arising in practice ranging from military organizations to corporate organizations and protection in operating systems. In the following, we show how SACL can be used to capture the lattice model of information flow [12].
3.1 Lattice model of information flow

In [12], Denning introduced the concept of lattice based information flow. He defined a flow model as the 5-tuple $(S, O, L, \rightarrow, \oplus)$, where

- $S$ denotes the set of subjects which are active entities in the system that perform actions resulting in information flow. Depending on the application, the level of abstraction of a subject could vary from a process/thread to a user name in a system.
- $O$ denotes the set of objects which are passive entities in the system used for storing information. The level of abstraction for objects can vary from a variable in a program to an entire database.
- $L$ is a finite set of labels also known as security classes or security levels. Every subject and object in the system is allotted a label, this can be done either statically or dynamically depending on the application.
- $\rightarrow$ is a binary relation on $L$, which defines permissible information flows.
- $\oplus$ is a function from $L \times L$ to $L$; $\oplus(l_1, l_2) = l$ means that when we combine information labelled $l_1$, $l_2$, the label of the resulting information must be at least $l$.

Fundamental requirements for information flow models like transitivity and reflexivity naturally force $(L, \rightarrow, \oplus)$ to be a lattice structure, where $\oplus$ denotes the join (least upper bound) operator. Restricting ourselves to read and write operations (represented by $r$ and $w$ respectively), we now show how a lattice model of information flow can be simulated in SACL. Given a flow model $FM = (S, O, L, \rightarrow, \oplus)$, we simulate it using the SACL $C_{SA}$, where

- for every subject $s \in S$ we have a principal $u \in U$.
- for every object $o \in O$ we have auth tags $(o, r) \in \mathcal{T}$ and $(o, w) \in \mathcal{T}$. Note that we abuse notation and use short forms for tags.
- for every label $l \in L$ we have two name identifiers $l^r \in \mathcal{A}$ and $l^w \in \mathcal{A}$.
- for every flow $(l_1, l_2) \in \rightarrow$ we have the name certs $(u, l_1, l_2)$ and $(u, l_2^r, l_2^w)$. Intuitively, the name certs are saying that who ever can read $l_2$ can also read $l_1$ and who ever can write $l_2$ can write $l_1$. These are precisely the implications of flow from $l_1$ to $l_2$.
- for assigning label $l$ to subject $s$ we issue the name certs $(u, l, u)$ and $(u, l^r, u)$.
- for assigning label $l$ to object $o$ we issue the auth certs $(u, (u, l), 0, (o, r))$ and $(u, (u, l), 0, (o, w))$.

Note that $u$ is the issuing principal and all names are defined in his name space. This corresponds to an administrator in a mandatory policy: for example in an organization, all the data is owned by the organization and its employees just access data according to the policy. Once we have the SACL specification, we can use the standard certificate composition or simple graph reachability algorithms to validate access requests and verify properties of such systems. Another important aspect is that a user might not want to operate at his highest level always (the level assigned to a user is typically representative of the highest level at which he can operate). In fact the principle of least privileges, an important security property, regulates that a user should not operate at a level more than needed for achieving the current task. Such principles can also be enforced by keeping track of the current label, and changing the label dynamically in such a way that it satisfies the policy.

![Fig. 3: Figure illustrating a flow model and an equivalent SACL](image)

A flow model and its equivalent SACL specification are illustrated in Fig. 3. From Fig. 3, we can conclude that principal $u_1$ can read and write $o_1$, and read $o_2$; while $u_2$ can read and write $o_3$ and write $o_4$. This matches the flow model semantics.

Note that the above simulation immediately gives us the capability to enforce Role-Based Access Control [31], the Bell-LaPadula security model [5], Biba’s integrity model [20] and the Chinese wall security model [9] as these are special cases of lattice model of information flow.

3.2 Related work

In [26], Myers et al. introduced the decentralized information flow control (called DIFC henceforth) model, based on a lattice of labels, for controlling information flow in systems with decentralized authority. DIFC model allows users to share information with untrusted code while still being able to control how this information is disseminated to others. Their model allows decentralized information declassification and supports finer data sharing.

Indrajit et al.[30], presented Laminar, a system that implements DIFC using an abstraction suitable for both operating system resources and program data structures. Users label data with secrecy and integrity labels, defining the desired security policy to be enforced at run time. Processes must access labelled data within special security regions.

Li et al.[23] identify several design principles for usable access control mechanisms and introduce UMIP, a practical mandatory access control model. The objective of UMIP is preserving system integrity in the face of network-based attacks. They have also demonstrated experimentally that UMIP is easy to configure, has low overhead and defends successfully against a number of known network-attacks.

Using the techniques presented in Section 3 for
modelling lattices in SACL, we can arrive at fine grained information flow policies in operating systems. We can then use the LP model of SACL presented in Section 2 to analyze and enforce these policies, and also verify their properties. In particular this will help us handle dynamic label changes that are essential for enforcing policies like the Chinese wall security policy.

4. Conclusions

In this paper, we have provided a LP model for SACL covering all its features including the authorization tags and delegation. The significance of our approach over existing work is that we have described a detailed algorithm for tag intersection computation which plays an important role in making access decisions. To our knowledge this is the first such algorithmic description. We have also described a prototype implementation of a tool based on the model, which enables us to verify properties of SACL specifications and authorizations that can be derived by composing several tags. By exploiting the hierarchy induced by the naming scheme of SACL, we have shown how lattice-based information flow policies can be modelled in SACL. This gives us the power to capture several well studied policy concepts and models (RBAC, Bell-LaPadula model for example) in SACL which form the basis for protecting confidentiality and integrity of data from untrusted applications. This also leads to a uniform framework in which heterogenous policies can be specified and their properties analyzed.

References

About the Authors

N. V. Narendra Kumar obtained M.S. and Ph.D. degrees in Computer Science from the Tata Institute of Fundamental Research (TIFR), and is a recipient of Microsoft Research India Rising Star Award in 2011 for his Ph.D. thesis work on “Malware Detection by Behavioural Approach and Protection by Access Control”. His research interests span several topics in the field of Cyber Security like Information Flow Control, Access Control and Trust Management, Cloud and Distributed Systems Security, Malware Detection and Protection. He is currently a Research Scientist at TIFR working on a DRDO sponsored project for the development of information flow secure operating system.

R. K. Shyamasundar is a Fellow ACM, Fellow IEEE and currently JC Bose National Fellow and Distinguished Professor in the Department of Computer Science and Engineering, IIT Bombay. He is a Fellow of all the National Academies of Science and Engineering in India and Fellow TWAS.
An Empirical Study of Popular Matrix Factorization based Collaborative Filtering Algorithms

Shamsuddin N. Ladha*, Hari Manassery Koduvely** and Lokendra Shastri***

In this paper we present a systematic study to understand how the performance of different sparse matrix factorization techniques are related to the characteristics of the underlying data sets and model parameters. We compare the performance of two popular matrix factorization methods, namely Variational Bayesian and Alternating Least Squares with Weighted-λ-regularization (ALS-WR) using synthetic and real data sets with different sparsity, noise levels and latent features. In our studies we have found that Variational Bayesian method relatively outperforms ALS-WR when the data is more sparse or contains more noise or the number of latent features in the model are less than the underlying features of the data. This indicates that Bayesian methods (for example VBMF) are better at avoiding model over-fitting compared to the explicit regularization methods (for example ALS-WR). In fact, we have proved that ALS-WR is a maximum-a-posteriori (MAP) estimator. Our experiments have also revealed instances when VBMF underfits the given data. We have also compared our results with some theoretical results available for factorization of complete matrices and found that our results demonstrate consistent behavior.

Index Terms: Collaborative Filtering, Matrix Factorization Algorithms, Comparative Analysis, Variational Bayesian Matrix Factorization, Alternate Least Squares, Singular Value Decomposition, Recommendation Systems

I. Introduction

Matrix factorization is a very popular technique used for Collaborative Filtering to generate personalized recommendations in E-commerce. Its basic idea is to approximate the user-item transaction matrix X as a product of two low-rank matrices U and V

\[ X \approx UV^T \] (1)

If the X matrix has dimensions M x N then U and V will have dimensions M x K and N x K respectively. Here K << M, N represents a set of hidden (a.k.a. latent) features (also referred to as the rank of a matrix) characterizing the factors that drive consumer preferences. In E-commerce scenarios, such as major on-line retailers, typically M and N would be of the order of millions. Also the matrix X would be highly sparse with only less than 10% filled by known values. Some of these values, in general, are corrupted by noise. This makes the above matrix factorization task highly non-trivial from the point of view of both computation time (due to scale) and accuracy (due to over-fitting).

In the last decade, several machine learning methods have been developed to improve the accuracy of sparse matrix factorization methods. Well known examples are Weighted Nonnegative Matrix Factorization (WNMF) [1], Alternate Least Squares with Weighted Regularization (ALS-WR) [2] and Variational Bayesian Matrix Factorization (VBMF) [3]. The performance of these different matrix factorization techniques vary depending upon the characteristics of the data sets. For example, Hernández-Lobato and Ghahramani have shown that VBMF, compared to other matrix factorization techniques, generates more accurate predictions at the long tail part of E-Commerce data sets [4]. This was attributed to the robustness of VBMF against over-fitting due to the underlying Bayesian averaging. There was no systematic study done to understand the relative performance of different Matrix Factorization methods in the context of different data sets and the aim of this paper is to addresses some aspects of this gap in the understanding. We studied two types of matrix factorization methods, one of which uses Bayesian inference scheme and the other uses regularization. Our goal is to understand how these two methods perform when underlying parameters of datasets (sparsity and noise levels) or model (latent features) are varied. For comparison purpose we used results of the SVD method as baseline which we know is unstable and known to over-fit when data is sparse and noisy. Hence, in this paper following matrix factorization methods for collaborative filtering are compared empirically:

- Variational Bayesian based Matrix Factorization (VBMF) [3]

This work was carried out while the authors were at Infosys Labs, Infosys Ltd.
• Alternating Least Squares with Weighted-λ-Regularization (ALS - WR) [2]
• Singular Value Decomposition (SVD) [5]

[6] provides a comprehensive literature survey of different collaborative filtering techniques, however no empirical evidence is reported. In contrast, our work studies the three methods in detail with empirical analysis. There were several empirical studies done in the past—by [7], [8], [9]—but none of them compare the three methods that we have compared in this paper. Our work augments the comparison work earlier done by these authors.

There are some theoretical studies aimed at understanding how different techniques regularize the matrix factorization. Nakajima et al. [10] have done a theoretical analysis of the VBMF for the case of complete matrix (no missing values) and shown that two type of shrinkage factors are responsible for regularization. Though their results do not apply to sparse matrices, where there are large number of missing entries, we have related our results to their predictions, for noisy but complete datasets. Another set of work looks at the necessary and sufficient criteria for recovering a low-rank matrix completely from a partial observation of its entries [11], [12]. We will interpret the results of our experiments in the light of these theoretical results.

Rest of the paper is organized as follows. In Sec. 2 we describe briefly the two methods of matrix factorization namely Variational Bayesian and Alternating Least Squares with Weighted-λ-Regularization. In Sec. 3 we present our experimental design and methods of generating synthetic data for comparing these two methods. Sec. 4 contains the results of our experimental studies, proof of ALS - WR as a maximum-a-posteriori (MAP) estimator, and explanation of results using some of the theoretical results available in the literature. In Sec. 5 we summarize our findings and present some future directions for research.

2. Variational Bayesian and Alternate least squares methods of matrix factorization

In this section we briefly describe the two matrix factorization technique used in our comparison study. For details readers may refer to the papers of [3] and [2].

2.1 Variational Bayesian Matrix Factorization (VBMF)

Variational approximation is a relatively new approach to estimate the posterior distribution in a Bayesian inference problem [13], [14]. Here the idea is to propose an approximate posterior distribution which has a factorized form and is parameterized by a set of variational parameters. One then minimizes the Kullback-Leibler divergence between the variational distribution and the target posterior by using the variational parameters to arrive at a good approximate solution for the posterior distribution. Raiko et al. [3] were the first to apply variational method for matrix factorization based collaborative filtering problems. Here we describe their method briefly, for more details readers may read the paper of Raiko et al. [3].

We assume the following likelihood function for \( X \), conditional on \((U, V)\), given by the product of Normal distribution for each of its elements where \( u_i \) and \( v_j \) denotes

\[
P(X|U, V) = \prod_{i=1}^{M} \prod_{j=1}^{N} \mathcal{N}(x_{i,j}|u_i \cdot v_j^T, \sigma_X^2)
\]

(2)

the \( i \)-th and \( j \)-th rows of matrices \( U \) and \( V \) respectively. And \( \sigma_X^2 \) is the noise parameter in observations in \( X \) and is treated as a hyper-parameter. Also we assume the following factorized prior distributions for \( U \) and \( V \) respectively

\[
P(U) = \prod_{i=1}^{M} \prod_{k=1}^{K} \mathcal{N}(u_{ik}|\bar{u}_{ik}, \sigma_u^2)
\]

(3)

\[
P(V) = \prod_{j=1}^{N} \prod_{k=1}^{K} \mathcal{N}(v_{jk}|\bar{v}_{jk}, \sigma_v^2)
\]

(4)

Here \( \bar{u}_{ik} \) and \( \bar{v}_{jk} \) are the prior variances for the elements of \( U \) and \( V \) respectively. The mean prior values \( \bar{u}_{ik} \) and \( \bar{v}_{jk} \) are usually taken to be zero.

Using Bayes rule, the posterior distribution for \( U \) and \( V \) is given by

\[
P(U, V|X) = \frac{P(X|U, V)P(U)P(V)}{P(X)}
\]

(5)

To compute the posterior distribution on the L.H.S of equation (5), Raiko et al. used the following approximate variational posterior distribution \( Q(U,V) \)

\[
Q(U, V) = \prod_{i=1}^{M} \prod_{k=1}^{K} \mathcal{N}(u_{ik}|\tilde{u}_{ik}, \tilde{\sigma}_u^2) \\
\times \prod_{j=1}^{N} \prod_{k=1}^{K} \mathcal{N}(v_{jk}|\tilde{v}_{jk}, \tilde{\sigma}_v^2)
\]

(6)

Here \( \tilde{u}_{ik} \), \( \tilde{v}_{jk} \), \( \tilde{\sigma}_u^2 \) and \( \tilde{\sigma}_v^2 \) are variational parameters whose values are found by minimizing the Kullback-Leibler divergence between \( Q(U,V) \) and \( P(U,V|X) \) given by

\[
KL(Q||P) = \int Q(U, V) \ln \frac{Q(U, V)}{P(U, V|X)} dU dV
\]

(7)

The prior variances \( \sigma_u^2 \) and \( \sigma_v^2 \) and variance parameter \( \sigma_X^2 \) of the observations are also found by minimizing the KL divergence.

In Raiko et al. [3], this minimization for finding the values of \( \tilde{u}_{ik} \), \( \tilde{v}_{jk} \), \( \tilde{\sigma}_u^2 \) and \( \tilde{\sigma}_v^2 \) was cast in the form of a set of coupled equations that needs to be solved iteratively. These equations are available in Raiko et al. [3].

2.2 Alternate Least Squares with Weighted-λ-Regularization (ALS - WR)

In the case of ALS - WR, the elements of the matrix \( U \) and \( V \) are found by minimizing the following error function

\[
f(U, V) = \sum_{i,j \in O} (x_{i,j} - \tilde{u}_{ik} \cdot \tilde{v}_{jk})^2 \\
+ \lambda \left( \sum_{i \in I} n_u ||\tilde{u}_i||^2 + \sum_{j \in J} n_v ||\tilde{v}_j||^2 \right)
\]

(8)

here \( n_u \) and \( n_v \) represents the number of observed entries in the \( i \)-th row and \( j \)-th column of the \( X \) matrix, and \( O \) represents...
the set of ordered pair of (row, column) index for which data is available in \( X \). The second term acts as the regularization term which depends on the sparsity of the data set through \( n_u \) and \( n_v \). For details of minimization steps readers may refer to the paper of R. Pan et al. [2]

3. Experimental Design

The performance comparison of the candidate algorithms was done using the popular error metric, RMSE. RMSE is defined as,

\[
\text{rmse} = \left( \frac{1}{U \times N} \sum_{i,j \in O} (r_{ij} - \hat{r}_{ij})^2 \right)^{\frac{1}{2}}
\]

(9)

The performance of the algorithms was measured by varying key properties of the dataset, such as noise in the data, sparsity of the data, and variance of the data. While many experiments were done on a synthetic dataset, with known number of underlying features, some experiments were performed on real datasets. Following implementation of the algorithms was deployed for experimental purposes,

- Our Java based implementation of VBMF with standard Gaussian priors.
- Mahout based implementation of ALS - WR [15].
- Lanczos implementation of SVD, which is available in R [16].

In the sequel, different experiments performed on synthetic and real datasets are described. The performance comparison, as mentioned earlier, is solely based on RMSE, and thus hardware specifications are not provided and time to run these algorithms is not reported.

3.1 Synthetic Dataset

The primary advantage of using a synthetic dataset is the availability of ground truth against which the performance of algorithms can be analyzed. A synthetic dataset, \( X \), was designed with 3 underlying user features \( (U_i, i = 1, 2, 3) \) and 3 underlying product features \( (V_j, j = 1, 2, 3) \). Three user types \( (U_i, i = 1, 2, 3) \) were defined, such that each user type, \( U_i \), had only one corresponding underlying user feature \( U_i \). Similarly, three product types \( (V_j, j = 1, 2, 3) \) were defined, each with one corresponding product feature \( V_j \). \( U_i \) only liked \( V_j \), where \( i = j \). The dataset had 1000 users, \( U_{\text{m}} \), and 100 products, \( V_p \). Each \( U_{\text{m}} \) was randomly assigned to a \( U_i \). Likewise, each \( V_p \) was randomly assigned to a \( V_j \). The products were rated from 1–5, with a rating of 4 or 5 defined as high rating and the rest as low ratings. \( X_{\text{mn}} \), i.e., rating of \( V_j \) by \( U_m \) was high only if the underlying user type of \( U_m \) liked the underlying product type of \( V_j \). Note that \( X \) is a dense matrix with pure ratings, i.e., without any noise. Thus \( X \) is the ground truth against which all the results will be compared. In our experiments, 80% of the dataset was used as the training set and the rest was used as the probe set.

4. Results

In the following paragraphs, experimental results obtained on synthetic and real datasets with respect to the model property–rank or latent features–and dataset properties–noise and sparsity–are described.

4.1 Rank of matrix factors

As per the ground truth, the rank, \( r \), of the matrix factors of \( X \) is 3. In this set of experiments, \( r \) was varied and the corresponding RMSE was computed for the algorithms under consideration. Fig. 1 shows the resulting plot.

From the plot it is clear that VBMF gives lowest RMSE, i.e., best performance, for the different values of \( r \). The difference in RMSE between VBMF and the next best algorithm, i.e., ALS - WR is significant when \( r \) is less than the actual number of underlying features. As expected RMSE is lowest for all the algorithms when \( r = 3 \). With \( r > 3 \), RMSE is expected to increase. However, the rate of increase of RMSE is highest for SVD, followed by ALS - WR. For VBMF, the RMSE almost remains constant with increasing \( r \). This behavior of VBMF is attributed to its discriminatory nature, whereby unwanted latent features (columns of matrix factors) are suppressed. Hence, the regularization in VBMF is called model-induced regularization [10]. Thus VBMF clearly outperforms the other two algorithms in this case.

For \( r = 1 \), Fig. 2 shows the scatter plot of the underlying matrix factors, \( U \) and \( V \), for all the algorithms. Note that \( X = UV^T \). The number of user and the product features, in this case 3, are correctly identified by the VBMF estimator.

4.2 Sparsity of dataset

Sparsity of \( X \) was gradually increased from 10% to 95% and the RMSE was computed for \( r = 3 \). As the sparsity increases, less data points are available for training (and testing). This generally results in a gradual increase in RMSE with increasing sparsity. Fig. 3 shows the performance plot of the 3 algorithms with varying sparsity.

RMSE using SVD is always higher than the other two algorithms, and the RMSE gradually increases with increasing sparsity as expected. However, for VBMF and ALS - WR, initially the RMSE increases gradually but after a tipping point, in this case around 80%, both the algorithms
VBMF, ALS - WR, and SVD respectively. (d), (e), (f): Computed \( U \) and \( X \) indicating possible overfitting, the RMSE for ALS - WR shoots up significantly, whereas for ALS - WR, Fig. 4 (right), all three singular values increase drastically beyond 80% sparsity, implying severe overfitting of the model to the input data. This behavior of ALS - WR can be inferred from the \( \lambda \)-regularization proposed in [2]. Here the weights are equal to the number of entries for each row and column of \( X \). As the sparsity increases, number of entries in rows and columns decrease, thereby reducing the effect of regularization leading to overfitting. This observation triggered a question, whether ALS - WR can be represented using a probabilistic framework? In Sec. 4.3 we show that ALS - WR is indeed a MAP estimator.

The two plots in Fig. 5, show the effect of increasing \( r \) on the RMSE for different sparsity levels of the input dataset. The RMSE curve for VBMF is identical for both \( r = 3 \) and \( r = 5 \), due to VBMF’s ability to suppress unwanted latent features (for \( r = 5 \)). However, the RMSE for ALS - WR is marginally lower for \( r = 5 \) than for \( r = 3 \) for sparsity less than 80%. Beyond that, the RMSE curve shoots up significantly due to overfitting, but the RMSE is still less than that of \( r = 3 \).

Fig. 2: Visualization of the matrix factors, \( U \) and \( V \), for \( r = 1 \). (a), (b), (c): Computed \( U \) using VBMF, ALS - WR, and SVD respectively. (d), (e), (f): Computed \( V \) using VBMF, ALS - WR, and SVD respectively. Underlying user and product features, based on which \( X \) was generated, are crisply visible as distinct clusters in the visualization of the matrix factors obtained using VBMF (a & d), although \( r = 1 \) is less than the number of actual features (3 in this case).

Fig. 3: Plot of the RMSE against increasing sparsity for \( r = 3 \).
4.3 ALS - WR as MAP Estimator

To begin, assume the likelihood function for \( X \) as defined in Eq. 2, i.e., similar to that of VBMF. Also, assume following factorized prior distributions for \( U \) and \( V \) respectively

\[
P(U) = \prod_{i=1}^{M} N(u_i|0, \tilde{\sigma}_i^2 I) \tag{10}
\]

\[
P(V) = \prod_{j=1}^{N} N(v_j|0, \tilde{\sigma}_j^2 I) \tag{11}
\]

Here \( \tilde{\sigma}_i^2 \) and \( \tilde{\sigma}_j^2 \) are the prior variances for random variables \( u_i \) and \( v_j \) of \( U \) and \( V \) respectively.

Using the Bayes theorem, the posterior distribution, \( P(U|V|X) \), is proportional to the product of the prior distribution and the likelihood function

\[
P(U|V|X) \propto P(X|U, V) P(U) P(V) \tag{13}
\]

Rewriting, \( P(U) \) and taking its negative log likelihood

\[
-\ln P(U) = \frac{1}{2} \sum_{i=1}^{M} \frac{u_i^T u_i}{\tilde{\sigma}_i^2} + \frac{N + 1}{2} M \sum_{i=1}^{M} \{\ln(2\pi) + \ln(\tilde{\sigma}_i^2)\} \tag{15}
\]

In the above equation, the last term can be ignored, as it does not depend on \( U \). Similarly, the negative likelihood of \( P(V) \) is

\[
-\ln P(V) = \frac{1}{2} \sum_{j=1}^{N} \frac{v_j^T v_j}{\tilde{\sigma}_j^2} + \frac{M + 1}{2} N \sum_{j=1}^{N} \{\ln(2\pi) + \ln(\tilde{\sigma}_j^2)\} \tag{16}
\]

From [5], it is known that

\[
-\ln P(X|U, V) = \frac{1}{2} \sum_{i} \sum_{j} (x_{i,j} - u_i \cdot v_j^T)^2 \tag{17}
\]

Hence, the negative log likelihood of Eq. 13 can now be written as
\[
\frac{1}{2} \sum_{i} \sum_{j} (x_{ij} - u_i v_j^T)^2 + \frac{1}{2} \sum_{i} \tilde{u}_i u_i^T + \frac{1}{2} \sum_{j} \tilde{v}_j v_j^T
\]

(18)

Define, \( \tilde{u}_i = \frac{1}{\sigma} u_i \) and \( \tilde{v}_j = \frac{1}{\sigma} v_j \), which implies that greater the number of samples higher the precision. Now, Eq. 18, which is a MAP estimate, is similar to the ALS - WR objective function (Eq. 8).

### 4.4 Noise in dataset

Next, the performance of algorithms was assessed when entries of the input dataset are corrupted by noise. \( D \) was corrupted by adding a zero mean Gaussian noise with standard deviation, \( \sigma \). \( \sigma \) was varied from 0.5 to 5 with an increment of 0.5. The plot in Fig. 6 shows effect of noise on RMSE when \( r = 3 \).

As seen in Fig. 6, as the noise increases, the RMSE steadily increases. Again, RMSE is highest for SVD. VBMF and ALS - WR have almost similar RMSE up to \( \sigma = 2 \), thereafter RMSE for ALS - WR is higher than that for VBMF.

In addition, both \( r \) and \( \sigma \) were varied simultaneously and the corresponding changes to RMSE are plotted in Fig. 7, Fig. 8, and Fig. 9 for VBMF, ALS - WR and SVD respectively.

In [10], the authors have established theoretical non-asymptotic bounds for singular values of the predictive output of VBMF and MAP estimators for dense matrices in presence of noise. The synthetic dataset, \( X \), is dense. Further, it has been established earlier in Sec. 4.3, that ALS - WR is a MAP estimator. Hence, these bounds can be verified for \( X \) corrupted by noise. For ALS - WR, as per [10],

\[
\gamma_h = \max \left\{ 0, \sigma - \frac{\sigma^2}{c_a c_b} \right\}
\]

(19)

where, \( \gamma_h \) is the \( h \)-th largest singular value obtained using MAP estimator; \( \gamma_h \) is the \( h \)-th largest singular value of the noisy dataset; \( \sigma \) is the standard deviation of the Gaussian noise with zero mean and varying \( \sigma \) for different values of \( r \).

![Fig. 6: Plot of the RMSE against Gaussian noise with zero mean and varying \( \sigma \) for \( r = 3 \).](image)

![Fig. 7: VBMF: Plot of the RMSE against Gaussian noise with zero mean and varying \( \sigma \) for different values of \( r \).](image)

![Fig. 8: ALS - WR: Plot of the RMSE against Gaussian noise with zero mean and varying \( \sigma \) for different values of \( r \).](image)

![Fig. 9: SVD: Plot of the RMSE against Gaussian noise with zero mean and varying \( \sigma \) for different values of \( r \).](image)

with which dataset is corrupted, \( c_a \) and \( c_b \) are prior variances of the \( h \)-th singular vector of \( U \) and \( V \) matrices respectively. For \( \sigma = 5 \), and \( r = 3 \); \( \gamma_1 = 946.48 \), \( \gamma_2 = 46.88 \), \( \gamma_3 = 46.53 \) for the corrupted \( X \) dataset. For ALS - WR, \( c_a = 100 \), i.e., number of products; and \( c_b = 1000 \), i.e., number of users. The estimated
values using Eq. 19 are \( \hat{\gamma}_1 \approx \gamma_1, \hat{\gamma}_2 \approx \gamma_2, \hat{\gamma}_3 \approx \gamma_3 \). The actual values, obtained by performing SVD on the predictive output obtained using ALS - WR is \( \hat{\gamma}_1 = 945.03, \hat{\gamma}_2 = 50.37, \) and \( \hat{\gamma}_3 = 49.38 \). These values are significantly close to the predicted theoretical values, further strengthening our claim of ALS - WR as a MAP estimator.

Similarly, for VBMF, as per \[10\],
\[
\max\{0, \frac{\sigma^2}{\gamma_h} - \frac{\sigma^2 \sqrt{\frac{M}{h}}}{\gamma_h} \} \leq \hat{\gamma}_h \leq \frac{\sigma^2}{\gamma_h} - \frac{\sigma^2 \sqrt{\frac{M}{h}}}{\gamma_h}.
\]

For VBMF, standard Gaussian priors are used, hence \( c_{\alpha} = c_{h_i} = 1 \), rest of the values pertaining to corrupted \( X \) remains the same. The estimated values using Eq. 20 are \( 841 \leq \hat{\gamma}_1 < 920.07, \hat{\gamma}_2 = 0, \) and \( \hat{\gamma}_3 = 0 \). The actual values, obtained by performing SVD on the predictive output obtained using VBMF is \( \gamma_1 = 946.57, \gamma_2 = 1.76 \times 10^4, \) and \( \gamma_3 = 1.44 \times 10^4 \). These values are close to the estimated values, predicted by [10].

An important point to note is that, although \( \hat{\gamma}_1 \) has high value for both ALS - WR and VBMF, \( \hat{\gamma}_2 \) and \( \hat{\gamma}_3 \) have been suppressed by VBMF estimator.

4.6 Real Dataset

Some of the experiments pertaining to varying noise and sparsity levels were conducted on two real world datasets, viz., MovieLens (ML) ratings 1M dataset and an E-commerce (Ecomm) client dataset. ML dataset has 6040 users and 3706 items, with 1000209 ratings between 1-5. Ecomm dataset has 472419 users and 239732 items, with 2720947 ratings between 1-3.

Fig. 10 and Fig. 11 show plot of RMSE against varying Gaussian noise and increasing sparsity respectively for ML ratings dataset. Variance of the derived datasets, with specific noise and sparsity, is plotted alongside. VBMF, in general outperforms the other two methods and in particular at higher noise and sparsity levels. It is observed that the shapes of VBMF and ALS - WR RMSE curves are similar to that of the variance curve. This effect is also seen in Ecomm dataset, as shown in Fig. 12, for VBMF. This suggests a strong correlation between the performance of these algorithms (VBMF and ALS - WR) and variance of the datasets. However, no such relationship was observed for SVD.

5. Conclusions

In this paper, the performance of two popular sparse matrix factorization algorithms VBMF and ALS - WR based on the characteristics of the datasets and the model parameters have been compared. Various experiments were conducted by varying noise and sparsity levels of the synthetic and the real datasets and model parameters (hidden features) to measure the performance of both the algorithms, using RMSE as the benchmark performance metric. As per the results obtained, Variational Bayesian based matrix factorization relatively outperforms the ALS - WR based method especially with increasing sparsity. We have demonstrated that ALS - WR is a MAP estimator. VBMF is more robust to over-fitting than MAP (ALS - WR) due to its underlying Bayesian averaging. Thus, our experimental results are in conformance with the theoretical behavior. We have also showcased instances, when VBMF underfits the data. Theoretical bounds on singular values of dense but noisy data were verified on our dataset. In future, we intend to establish theoretical bounds on singular values for matrix factorization of sparse and noisy data.
REFERENCES

About the Authors

Shamsuddin N. Ladha is presently working at TATA Research Development and Design Center, as a Research Associate, in Remote Sensing. Earlier, he was at Infosys Labs. His research interests are in the area of Image Processing, Computer Vision, and Machine Learning. He has a PhD in Computer Science and Engineering from IIT Bombay and Monash University. Prior to joining the PhD program, he has extensively worked in Software Research and Development in the USA and India.

Hari Manassery Koduvely is a Principal Engineer with Samsung R&D Institute, Bangalore, India. His research interests are in the area of Machine Learning and Statistical Models for Natural Language Understanding. Prior to Samsung he has worked in Infosys and Amazon. Dr. Hari has a PhD in Statistical Physics from Tata Institute of Fundamental Research in Mumbai and has done Postdoc at Weizmann Institute of Science, Israel and Georgia Tech, Atlanta.

Lokendra Shastri is a Consultant (VP level), Advanced Technology Labs at Samsung Research Institute, Bangalore. Earlier, he was Assoc. Vice-President and GM, Research at Infosys Limited. Before entering industry, Dr. Shastri was a Senior Research Scientist at the International Computer Science Institute, a research organization affiliated with the CS Division of the UC Berkeley, and on the faculty of the CIS Department at the Univ. of Pennsylvania. Dr. Shastri has made significant contributions to massively parallel models of inference, knowledge representation, semantic networks, episodic memory, and computational neuroscience. He has authored a book and over 80 invited chapters and peer-reviewed articles, with over $3100$ citations, and he also holds multiple patents. Dr. Schastri received his PhD in computer science from the University of Rochester, an MS in computer science from the Indian Institute of Technology, Madras, and a B.E. (honors) in Electronics (with distinction) from the Birla Institute of Technology and Science, Pilani.
Mitigation of Jamming Attack in Wireless Sensor Networks

Manju. V.C.*, Dr. Sasi Kumar**
* Research Student, Kerala University, Email: manju_tvm@yahoo.com
** Professor. Marian Engineering College, Email: drmsasikumar@yahoo.com

The categorization of jamming attacks is based on different network performance degradation phenomena like network congestion or channel fading. The development of multimodel detection technique can assist in detecting jamming attacks with lower false alarm rate and high precision. This paper proposes a fuzzy model based on RSSI, PDR and PSR for detection and classification of jamming attack precisely and examine the combination of different Jamming methods and Detection of service in Wireless Sensor Networks (WSN).

Keywords: Jamming Attacks, Wireless Sensor Networks, Multi-model detection technique

1. Introduction

A safe and sound sensor network is a difficult task due to the limited components connected with low-cost sensor hardware. An intruder intercepts a transmission and jams the transmission. An extensive set of wireless sensor applications can be built for monitoring hostile environments like wild forests, medical healthcare to promote health services, and industries to attain high quality management. In sensor networks, we may come across a type of node considered as faulty node and this node is measured to be faulty if it is providing the results which significantly deviate from the results of the neighbor nodes. The academics seems to believe that the jammer has limited resources, who may be an intruder or the own compromised nodes, fully or partially knowing the protocols, and attacking the network from a location well within the geographical extent of the WSN. The military believes that it is neither worth the effort to learn the WSN protocols nor essential to move into the WSN geographical area for jamming, since it is so easy to jam the nodes with brute RF power from a far-off safe distance, especially when the RF frequency is known. Therefore, there is a need to balance the two approaches in modelling the jamming attack to make our counterjamming efforts, like jamming detection and jammer localization, suitable for information warfare.

II. Literature Review

Accurate detection of radio jamming attacks is a challenge in mission critical scenarios. Many detection techniques have been proposed in the literature, but the precision component is always an issue and numerous scholars have suggested different mechanisms to detect jamming attacks. Each of these recommended methods are to be applied at the individual node level to crisply conclude whether the node is jammed or not. Their technique is either based on threshold values or to use digital signal processing techniques to differentiate between a legitimate and an illegitimate (jamming) signal and thus conclude about the presence or absence of the jammer. The existing models and concepts are given in the table below (Table 1 and Table 2)

<table>
<thead>
<tr>
<th>Table 1 : Existing suggested models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Suggested by</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Xu et al. [12]</td>
</tr>
<tr>
<td>Wood et al. [1]</td>
</tr>
<tr>
<td>Muralidharan et al. [10]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2 : Existing suggested concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suggested by</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>Xu et al. [12]</td>
</tr>
<tr>
<td>Muralidharan et al. [10]</td>
</tr>
<tr>
<td>Cakiroglu et al. [8]</td>
</tr>
<tr>
<td>Reese et al.[6]</td>
</tr>
<tr>
<td>Strasser et al. [9]</td>
</tr>
</tbody>
</table>
Various techniques are used for assessment of the node settings with those of its neighbours to fine tune their results and we examine the existing results since proposed by various scholars in the following sub sections.

The review of the mentioned prototypes reveals that while the military prototypes targets to attack the network at the physical layer with RF power being their main weapon. The educational prototypes target the data-link layer with RF power levels at par with the existing average transmitted power of a WSN node.

The military believes that it is neither worth the effort to learn the WSN protocols nor essential to move into the WSN geological region for jamming. As an observation, the fact that it is easy to jam the nodes with brute RF power from the safe distance.

III. Proposed Methodology

The proposed model is that the RF jamming detects the occurrence of the jammers and to classify them and the model is inherited on multimodal approach. It integrates PDR and signal strength as the detection constraints and the PDR is calculated over the given illustration window of time. Signal strength variation (S) and PW are the model-specific parameters.

Signal strength variation (S) is the change in signal strength taken in dB,

\[ S = SS_{\text{observed}} - SS_{\text{network}} \]

where SS network is the signal strength achieved during training session without jamming and SS observed is the signal strength observed when the network is suspected to be under attack. PW is the measure of time for which (S) is greater than the threshold value. It is engaged in microseconds.

The jamming pulse acts as highpower Gaussian noise which can appear quite a few period over the channel and to calculate the N samples of channels established energy s(t) are composed and the illustrations, as a result figure a larger window of samples s(k), s(k - 1),...,s(k - N + 1), taken at consecutive smaller sampling time windows.

The detection is done using Equation 1.

\[ T(k) = \sum_{j=k}^{k-N+1} (s(j)2)^N \]  

Where T(k) is the average jamming pulse observed for casement of N illustrations and to make a decision the occurrence of a jammer T(k) is evaluated with a quantity of threshold and the threshold is cautiously calculated to avoid false detection. The following are the relevant parameters collected by the detector in a given sample window of time to detect the jamming attack and its type: (1) PDR, (2) \( \Delta S \), and (3) pulse width subject to \( \Delta S > 0 \).

PDR is the ratio of the total number of packets correctly received to the total number of packets received. For an environment with noise and interference, the PDR is measured at the receiver side as the ratio of number of packets received that pass cyclic redundancy check (CRC) to the total number of packets received.

The PDR of the known sampling window can be calculated as follows:

\[ \text{PDR} = (1-P_j)(1-P_c), \] (2)

Where \( P_j \) is the jamming probability computed for the different jammers in the subsequent sections and \( P_c \) is the collision probability of the packets when there are many transmitting nodes at the same time and in our experimentation of single transmitter and receiver is concerned, \( P_c \) is always zero. However, it comes into consideration when the number of contending stations for the channel is more than one.

For which \( \Delta S > 0 \) and \( y \) is the whole sampling window time. Subsequently, it is written as follows:

\[ R_j = \frac{x}{y}, \] (3)

The jamming rate is the rate the jammer squash the channel and if \( x \) is the time Where \( R_j \) is the jamming velocity. Then For example, the jamming pulse preceding for 1 \( \mu \)s in a total window of 1,000 \( \mu \)s, \( R_j \) is said to be 1/1,000 and for a constant jammer the continuous transmission of jamming pulse, the rate is 1. Jamming rate \( R_j \) for the time \( T \) can be derived through the following equation:

\[ R_j = \sum_{i=1}^{N-1} (P_{Wi} + 1 - P_{Wi}) T \] (4)

Where \( P_W \) is the jammer pulse time and \( (P_{Wi}+1-P_{Wi}) \) is the subwindow time during which \( \Delta S > 0 \). The total sample window time is \( T \).

III.a. Fuzzy Logic Controllers

The determining effort by Zadeh [7] on fuzzy algorithms introduced the idea of formulating the control algorithm by logical rules.

An FLC consists of a set of rules of the form: IF (a set of conditions are satisfied)

THEN (a set of consequences can be inferred).

In view of the fact that the antecedents and the consequents of these IF-THEN rules are associated with fuzzy concepts and they are frequently called fuzzy conditional statements Then in FLC terminology, a fuzzy control rule is a fuzzy conditional statement in which the antecedent is a condition in its application domain and the consequent is a control action for the system under control. The inputs of fuzzy rule-based systems should be given by fuzzy sets. Therefore, we have to fuzzify the crisp inputs, furthermore the output of a fuzzy system is always a fuzzy set. As a result to get crisp value we have to defuzzify it. Fuzzy logic control systems usually consist of four major parts: Fuzzification interface, Fuzzy rule base, Fuzzy inference engine and Defuzzification interface, as is presented in the figure 1 and the functions of fuzzification interface and defuzzification interface is shown in Fig. 2 and Fig. 3.
The fuzzy deduction mechanism is the kernel of a FLC and it has the capability of simulating human decision-making based on fuzzy concepts and of inferring fuzzy control actions employing fuzzy implication and the rules of inference in fuzzy logic.

### Step 1: Fuzzification

The first step is to take the crisp inputs, \( x_0 \) and \( y_0 \), and determine the degree to which these inputs belong to each of the appropriate fuzzy sets

\[
\mu_{A_1}(x_0) = 0.5, \quad \mu_{A_2}(x_0) = 0.2, \quad \mu_{B_1}(y_0) = 0.1, \quad \mu_{B_2}(y_0) = 0.7
\]  

### Step 2: Rules evaluation

The fuzzified inputs are applied to the antecedents of the fuzzy rules and if a known fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation and to assess the disjunction of the

In other words, the fuzzy control action \( C \) inferred from the fuzzy control system is transformed into a crisp control action. Where, \( \text{defuzzifier} \) stands for a defuzzification operator and the most used defuzzification operators. For a discrete fuzzy set \( C \) having the universe of discourse \( V \).

Their work was inspired by an equally influential publication by Zadeh [7]. Interest in fuzzy control has continued ever since, and the literature on the subject has grown rapidly. A survey of the field with fairly extensive references may be found in [6] or, more recently, in [8]. In Mamdani’s model the fuzzy implication is modelled by Mamdani’s minimum operator, the conjunction operator is min, the t-norm from compositional rule is min and for the aggregation of the rules the max operator is used.

In order to explain the working with this model of FLC will be considered the example from [7] where a simple two-input one-output problem that includes three rules is examined:

- Rule 1: IF \( x \) is \( A_3 \) OR \( y \) is \( B_1 \) THEN \( z \) is \( C_1 \)
- Rule 2: IF \( x \) is \( A_2 \) AND \( y \) is \( B_2 \) THEN \( z \) is \( C_2 \)
- Rule 3: IF \( x \) is \( A_1 \) THEN \( z \) is \( C_3 \)

#### Step 2: Rules evaluation

The fuzzified inputs are applied to the antecedents of the fuzzy rules and if a known fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation and to assess the disjunction of the

In other words, the fuzzy control action \( C \) inferred from the fuzzy control system is transformed into a crisp control action. Where, \( \text{defuzzifier} \) stands for a defuzzification operator and the most used defuzzification operators. For a discrete fuzzy set \( C \) having the universe of discourse \( V \).

Their work was inspired by an equally influential publication by Zadeh [7]. Interest in fuzzy control has continued ever since, and the literature on the subject has grown rapidly. A survey of the field with fairly extensive references may be found in [6] or, more recently, in [8]. In Mamdani’s model the fuzzy implication is modelled by Mamdani’s minimum operator, the conjunction operator is min, the t-norm from compositional rule is min and for the aggregation of the rules the max operator is used.

In order to explain the working with this model of FLC will be considered the example from [7] where a simple two-input one-output problem that includes three rules is examined:

- Rule 1: IF \( x \) is \( A_3 \) OR \( y \) is \( B_1 \) THEN \( z \) is \( C_1 \)
- Rule 2: IF \( x \) is \( A_2 \) AND \( y \) is \( B_2 \) THEN \( z \) is \( C_2 \)
- Rule 3: IF \( x \) is \( A_1 \) THEN \( z \) is \( C_3 \)

#### Step 1: Fuzzification

The first step is to take the crisp inputs, \( x_0 \) and \( y_0 \), and determine the degree to which these inputs belong to each of the appropriate fuzzy sets

\[
\mu_{A_1}(x_0) = 0.5, \quad \mu_{A_2}(x_0) = 0.2, \quad \mu_{B_1}(y_0) = 0.1, \quad \mu_{B_2}(y_0) = 0.7
\]  

#### Step 2: Rules evaluation

The fuzzified inputs are applied to the antecedents of the fuzzy rules and if a known fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation and to assess the disjunction of the
rule antecedents, one uses the OR fuzzy operation and characteristically, the classical fuzzy operation union is used:

\[ \mu_{A \cup B}(x) = \max \{\mu_A(x), \mu_B(x)\} \]  

(8)

Similarly, in order to evaluate the conjunction of the rule antecedents, the AND fuzzy operation intersection is applied:

\[ \mu_{A \cap B}(x) = \min \{\mu_A(x), \mu_B(x)\} \]  

(9)

Now the result of the antecedent evaluation can be applied to the membership function of the consequent. The most common method is called clipping. Because top of the membership function is sliced and the brief fuzzy set loses a few information. On the other hand clipping is preferred because it involves fewer complexes and generates an aggregated output surface that is easier to defuzzify. Another method, named scaling. It offers an improved approach for preserving the original shape of the fuzzy set: the original membership function of the rule consequent is adjusted by multiplying all its membership degrees by the value of the rule antecedent. The set of fuzzy rules used in this model is given in table3.

<table>
<thead>
<tr>
<th>Rule</th>
<th>PDR</th>
<th>PSR</th>
<th>RSSI</th>
<th>Jamming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Deceptive</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>Constant</td>
</tr>
<tr>
<td>4</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Reactive</td>
</tr>
<tr>
<td>6</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>Constant</td>
</tr>
<tr>
<td>8</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Random</td>
</tr>
<tr>
<td>10</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>11</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>Reactive</td>
</tr>
<tr>
<td>12</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>13</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>14</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>16</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>17</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>18</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>No</td>
</tr>
</tbody>
</table>

Step 3: Aggregation of the rule outputs

The membership functions of every rule consequents beforehand clipped or scaled are combined into a single fuzzy set.

Step 4: Defuzzification

The most popular defuzzification method is the centroid technique. It finds a point representing the centre of gravity (COG) of the aggregated fuzzy set A, on the interval [a, b].

A rational estimate can be found by calculating it over a sample of points and obtain a value of

\[ \text{COG}=0.64 \]

Through NS2 simulations, multiple set of three sets of crisp inputs, PDR, PSR and RSS are first mapped into fuzzy membership function. We have chosen trapezoid shape basically as it has a capability to manipulate easily to be an unsymmetrical function and can be mathematically to resemble natural function such as Bell function and the Gaussian function.

The following graphical representation present the trapezoid functions in respect to PDR, PSR and RSS.(figure 5,6,7)

IV. Performance Evaluation

The study also evaluated the performance of a reactive jammer that jams all packets. We further studied the robustness of our approach under the condition that the jammer does not succeed to jam all packets, but is still able to destroy 90% of the packet. Figure 8 shows the impact of these forms of reactive jamming on the correlation between the PDR, and RSS. Green colour indicates no jamming, while blue is reactive, followed by red, represents random, while black indicates deceptive and magenta is a constant.
The RSS distributions are further sorted based on their data-rates; for purposes of clarity we only show data-rates of -30 and -60 dbm. For example, in this experiment, 0% of packets in error due to signal have an PDR of about -60 dBm or less, while only 40% of no jamming due to signal have PDR of -83 dBm or less. Thus, by using a ‘cut-off’ value of -60 dBm, it would possible to achieve 0% collision loss. It would be possible to capture about 90% of collision cases while incurring a false-positive rate of 2%. Thus, RSS can act as a good metric for inferring the cause of packet loss.

**Fig. 8 : Input and Output surface corresponding to the membership values of input and output**
We used metrics such as PDR, PSR and RSSI to evaluate the performance of our approaches. Packet delivery ratio (PDR) is used to measure the effectiveness of jamming attacks, and consequently to describe the quality of a link. To detect jamming attack, we choose PDR and SS as jamming attack metrics for our system. In a normal scenario with no interference, high SS corresponds to a high PDR. Though, if the SS is low, the PDR will also be low. Parallel to this, a low PDR does not essentially indicate low SS: it may be that all of a neighbour nodes have died (from consuming battery resources or devices faults) otherwise the node is jammed and the key observation here is that in the jammed case. The SS have to be high and the PDR is low and the mixture of PDR, PSR and RSSI is capable of detecting any form of jamming attack and as discussed in the earlier sections.

**Simulation evaluation**

By using MATLAB, our own stimulator was implemented. A wireless network environment in a 300-by-300 feet field is simulated within which network nodes were distributed uniformly. Each node's transmission range was set to 30 feet. The performance of three algorithms, CL, VFIL-Tr and, VFIL-NoTr was assessed by us in several network conditions, including variable jammer's NLB radius and network node densities. We placed the jammer at the centre of simulation area in order to study the impact of those network parameters on algorithms, so that the jammer was surrounded by multiple.

The effectiveness of our virtual force iterative localization approach under both the region based model (RBM) and the SNR-based model (SBM) is assessed through simulation.

<table>
<thead>
<tr>
<th>Parameters used in simulations</th>
<th>Parameter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{elec}$</td>
<td>5 nJ/bit</td>
</tr>
<tr>
<td>$E_a$</td>
<td>10 pJ/bit/m²</td>
</tr>
<tr>
<td>$E_{ma}$</td>
<td>0.0013 pJ/bit/m⁴</td>
</tr>
<tr>
<td>$E_o$</td>
<td>0.5 J</td>
</tr>
<tr>
<td>EDA</td>
<td>5 nJ/bit/message</td>
</tr>
<tr>
<td>$d_o$</td>
<td>70 m</td>
</tr>
<tr>
<td>Message size</td>
<td>4000 bits</td>
</tr>
<tr>
<td>$P_{opt}$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**V. Result**

We first studied the effects of various network node densities on the localization performance. To adjust the network node densities, we varied the total number of nodes, N, deployed in the simulation. In particular, we chose N to be 200, 300, and 400, respectively.

Table 5: Simulation Deployment with 50 nodes

<table>
<thead>
<tr>
<th>Node IDs</th>
<th>Xpos (in meters)</th>
<th>Ypos (in meters)</th>
<th>PDR (%)</th>
<th>PSR (%)</th>
<th>RSSI (dBm)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.1214</td>
<td>72.0548</td>
<td>34</td>
<td>57</td>
<td>-115.1749</td>
<td>Reactive'</td>
</tr>
<tr>
<td>2</td>
<td>65.4512</td>
<td>65.2501</td>
<td>70</td>
<td>30</td>
<td>-125.7777</td>
<td>'No'</td>
</tr>
<tr>
<td>3</td>
<td>8.5464</td>
<td>63.3436</td>
<td>26</td>
<td>90</td>
<td>-109.7472</td>
<td>'No'</td>
</tr>
<tr>
<td>4</td>
<td>30.0269</td>
<td>71.5057</td>
<td>33</td>
<td>36</td>
<td>-117.2348</td>
<td>'Random'</td>
</tr>
<tr>
<td>5</td>
<td>59.2348</td>
<td>90.5185</td>
<td>26</td>
<td>95</td>
<td>-128.5093</td>
<td>'Reactive'</td>
</tr>
<tr>
<td>6</td>
<td>73.8729</td>
<td>0.4461</td>
<td>85</td>
<td>70</td>
<td>-129.6269</td>
<td>'No'</td>
</tr>
<tr>
<td>7</td>
<td>56.2432</td>
<td>33.7716</td>
<td>33</td>
<td>7</td>
<td>-120.9359</td>
<td>'Constant'</td>
</tr>
<tr>
<td>8</td>
<td>45.8301</td>
<td>65.7190</td>
<td>57</td>
<td>40</td>
<td>-119.9987</td>
<td>'Random'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>50</td>
<td>84.4873</td>
<td>75.3107</td>
<td>71</td>
<td>24</td>
<td>-131.1934</td>
<td>'No'</td>
</tr>
</tbody>
</table>

We originated that the outcome of different type of jammers is different on the WSN and conforms to the expected pattern and in which their effect and in order of decreasing force,
We mention here that 712 simulations (table 4) like (238 out of 240 for 25-node and 237 out of 240 for 50-node and 237 out of 240 for 100-node configurations) out of 720 simulations passed the chisquare test. It shows the values of PDR for 100 nodes configuration for different types of jammers, JIs and jnr.

![Graph showing PDR values for various nodes](image)

**Fig. 10:** Packets send ratio of various nodes

It shows the values of PDR for 100 nodes configuration for different types of jammers like JIs and jnr.

![Graph showing PDR values at each node](image)

**Fig. 11:** Packet delivery ratio

It shows the values of PDR for 100 nodes configuration for different types of jammers, JIs and jnr.

Value of packets send ratio observed at each node is defined in figure 10. Packet delivery ratio and received signal strength of various nodes are depicted in Fig. 11 and 12. The percentage of jamming attack present in the network model is defined in Fig. 9.

VI. Conclusions

The course of action of diagnosing jamming attacks has been investigated in this paper. The focus was kept on localizing the jammer after a jamming attack is identified specifically. The development of multimodal detection technique can help in detecting jamming attacks with lower false alarm rate and high precision to mitigate jamming effects and false data, we are formulated. To mitigate the jamming effects and false data we are formulated two jamming models, signal to noise ratio SNR based and location based and to efficiently tackle the Jamming attacks in multiple radios WSN; we develop a novel mitigation for identifying nodes. This paper is to investigate the scalability and stability of this method to various WSNs. The outstanding performances showed to strengthen the capability and a routing protocol to control trigger nodes to receivers so as to remain jammers idle. We study the idealized case of perfect knowledge by both the jammer and the network. The network about the strategy of each other and the case where the jammer and the network lack this knowledge and demonstrated the probability of jamming attacks by performing fuzzy logic controllers.

References


About the Author

**Manju V. C.** received BE Degree from M.S university, India and M.E degree from M.K University. Currently she is doing PhD under University of Kerala, India. Her research interest includes wireless networks ,wireless sensor networks and link layer protocols.

**M. Sasikumar** received his BSc (Engg) degree from College of Engineering, Trivandrum in 1968, ME degree from IISc Bangalore in 1976 and PhD degree from IIT Madras in Electronic Circuits and Signal Processing in 1984. He had worked in Govt. Colleges of Kerala. He worked as the Principal of College of Engineering, Trivandrum in 1998. He was the Senior-Joint Director in Directorate of Technical Education, Trivandrum during 1998-2000. He had published many articles in national/international conferences and journals. He had published a book entitled Power supplies. He had coordinated and guided many research projects in the field of signal processing and image processing. He is a senior member of Journal of Computer Society of India.
An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS

Shyamli Rao* & Amit Joshi**

* Department of Computer Engineering and Information Technology College of Engineering Pune, Maharashtra, India
Email: shyamli.rao27@gmail.com

** Department of Computer Engineering and Information Technology College of Engineering Pune, Maharashtra, India
Email: adj.comp@coe.ac.in

Hadoop is a distributed framework which uses a simple programming model for the processing of huge datasets over the network of computers. Hadoop is used across multiple machines to store very large files, which are normally in the range of gigabytes to terabytes. High throughput access is acquired using HDFS for applications with huge datasets. In Hadoop Distributed File System (HDFS), a small file is the one which is smaller than 64MB which is the default block size of HDFS. Hadoop performance is better with a small number of large files, as opposed to a huge number of small files. Many organizations like financial firms need to handle a large number of small files daily. Low performance and high resource consumption are the bottlenecks of traditional method. To reduce the processing time and memory required to handle a large set of small files, an efficient solution is needed which will make HDFS work better for large data of small files. This solution should combine many small files into a large file and treat these large files as an individual file. It should also be able to store these large files into HDFS and retrieve any small file when needed.

Keywords: hadoop distributed file system, small file

1. Introduction

Hadoop is an open source distributed framework which stores and processes large data sets and is developed by Apache Software Foundation. It is built on clusters of commodity hardware. Each single machine server stores large data and provides local computation which can be extended to thousands of machines. It is derived from Google’s file system and MapReduce. It is also suitable to detect and handle failures. It can be used by the application which processes large amount of data with the help of large number of independent computers in the cluster. In Hadoop distributed architecture, both data and processing are distributed across multiple computers. Hadoop consists of the Hadoop Common package, HDFS and MapReduce engine. A Hadoop cluster consists of a NameNode and DataNodes. Function of NameNode is to manage the metadata of file system and the actual data is stored at the DataNodes.

It is obligatory to divide the data across different DataNodes when large amount of data is stored on a single machine. Distributed file systems are the ones which are responsible for the management of data storage over a network. The distributed file system used by Hadoop is called HDFS and it is a storage system. HDFS has been considered as a highly reliable file system. HDFS is a scalable, distributed, high throughput and portable file system programmed in Java for the distributed framework of Hadoop. HDFS has read-many-write-once model that allows high throughput access, simplifies data consistency and eases concurrency control requirements. HDFS helps to connect nodes which are personal computers present in a cluster in which data is distributed. Then, the data files can be accessed and stored as an one seamless file system. HDFS work is done efficiently by distributing data and logic to on nodes for parallel processing. It is well grounded as it retains multiple copies of data and assigns processing logic in the event of failures on its own.

Hadoop also comes with the MapReduce engine. For writing applications, Hadoop MapReduce is used to process large data parallelly on large clusters. A MapReduce job normally divides the input data into separate chunks. These chunks are then processed parallelly by the map tasks. The framework classifies the results of the maps. These are fed as input to the reduce tasks. File system stores both the input and the output of the job. The framework manages scheduling of tasks, supervising them and also the unsuccessful tasks are reexecuted. It includes JobTracker and TaskTracker. Client applications sends requests of the MapReduce jobs to JobTracker and the JobTracker assigns these jobs to available task tracker nodes in the cluster. The JobTracker tries to keep the data and its processing in close proximity to

CSI Journal of Computing  |  Vol. 2 • No. 4, 2015
each other. MapReduce processing need not be done in Java unlike Hadoop which needs Java base.

II. Background

A. Hadoop Distributed File System

The Hadoop Distributed File System provides several services like NameNode, DataNode, SecondaryNameNode, JobTracker, TaskTracker, etc.

The NameNode is the main part of an HDFS. It keeps the metadata information which is the directory tree of all files present in the file system. It also tracks where the file data is kept across the cluster. However, the data of these files is not stored, but the metadata is stored. There is a single NameNode running in any DFS deployment. Namenode is the master of HDFS. It manages the slave DataNode daemons to perform input output tasks at the low-level. It also manages and keeps the information about on which nodes the data blocks of file are actually stored, on what basis the files are split into file blocks, and the functioning of the distributed files system. In the HDFS Cluster, the NameNode is a single point of failure. Whenever client applications need to locate a file or say they want to add or copy or move or delete a file, they always talk to the NameNode. Then the NameNode responds to the valid and successful requests by returning a list of DataNode where the data actually resides.

In HDFS, DataNode stores data. Generally there are many DataNodes, with data replicated across them to recover the data in case of data failure. When the cluster is started, DataNode connects to Namenode and waits till the acknowledgement comes from the Namenode. For file system operations, the Datanode responds to requests from the NameNode. Once the NameNode has given the location where the data is stored, client applications can instantly talk to a DataNode. Whenever MapReduce tasks are submitted to TaskTracker which is near a DataNode, they immediately contact to the DataNode for getting the files. TaskTracker operations should generally be performed on the same machine where Datanode resides. This helps for the MapReduce operations to be executed in close proximity to the data. To duplicate the data blocks, DataNode communicate with each other and thus redundancy is increased. The backup store of the blocks is provided by the Datanodes. To keep the metadata updated, Datanodes constantly report to the NameNode. If any one DataNode crashes or becomes unreachable over the network, backup store of the blocks assures that, file reading operations can still be performed.

In HDFS, the Secondary NameNode acts as an assistant node to supervise the state of the cluster. In each cluster, there is one SecondaryNameNode. It resides on each machine in the cluster. The SecondaryNameNode differs from the NameNode in the context that, any real-time changes to HDFS are not received or recorded by this process. After definite time intervals defined by the cluster configuration, SecondaryNameNode communicates with the NameNode to get the instances of the HDFS metadata. It is a daemon that periodically wakes up, triggers a periodic checkpoint and then goes back to sleep. If the NameNode goes down, the SecondaryNameNode helps to lower the outage duration and data loss. If a NameNode fails and to use the SecondaryNameNode as the primary NameNode, we need to manually reconfigure the cluster.

The function of the JobTracker is to distribute the MapReduce tasks between particular nodes containing the data in the cluster. These nodes might be present in the same rack. Job tracker acquires jobs from the client applications. Once the code is submitted to the cluster, the JobTracker determines the enactment strategy by finding out which files to process. It then allocates nodes to different tasks and superintends all running tasks. To determine the location of the data, the JobTracker consults the NameNode. The JobTracker discovers TaskTracker nodes that available for processing the data. The work is then given to the located TaskTracker nodes. In case of task failure, that task is automatically instigated by the JobTracker on some other node. The number of retries has already been defined. The only JobTracker in Hadoop cluster is run on the master node.

The TaskTracker in the cluster is the one that receives tasks such as Map, Reduce and Shuffle operations from a JobTracker. The number of tasks a TaskTracker can accept is configured with a set of slots. To ensure that process failure does not bring down the task tracker, it manages a separate Java Virtual Machine (JVM) processes to do the work. It supervises these processes and captures the output and exit codes. It notifies the JobTracker whether the process completed successfully or not. After every fixed interval of time, the TaskTrackers also send out pulse messages to the JobTracker, to conform that the JobTracker is still alive.

B. Small File Problem

Normal approach for storing large number of small files includes directly storing the files in HDFS without any preprocessing done, this has many disadvantages, following are few amongst them:

1) In HDFS, if data in files is significantly smaller than the default block size, then the performance reduces drastically.

2) If small files are stored normally in HDFS, then it wastes a lot of space in storing metadata of all files. When small files are stored in HDFS, there will be lots of seeks and jumps from one datanode to other to get a file, which is ineffective data access method.

III. System Design

We have tried to eliminate the drawbacks of storing large number of small files in HDFS, so that the time required to read and write the files would be much less and also the metadata storage decreases significantly.

While writing the files into HDFS, first we have sorted the files and then stored them in order to get better prefetching. Our program stores the similar files(similarity of files is on the basis of their extension) zipped together into HDFS. Once the sorting has been done, we have zipped the small files till
the size of zipped file i.e combined file is equal to 64MB or till the extension of small files change. This approach helps, as it leads to the concept of locality of reference i.e. user will refer similar types of files for later use. Storing files into HDFS requires less time since we are zipping the files. Therefore, for example, instead of writing thousands of small files into HDFS, we will write only few zip files into HDFS, each zip containing hundreds of small files.

For the first read, a map gets created for each extension, then this map is searched for the given small file name and returns the combined file name. This combined file is copied to local machine. Since whole combined file is being copied to local machine the similar files present in the zipped file are being prefetched. For the next reads, time required to get combined filename from small file name would be less since map is already created and search time in a map is O(1) i.e constant. If a similar small file is read then same combined file name is returned hence it takes further less time since the combined file is already present in local machine.

![Fig. 1: Write Operation](image1)

A. File Merging Algorithm

1) Start the program
2) Select the Write button to write the files into HDFS
3) Enter the small file name
4) Our program reads the file, sorts it on the basis of extension and stores it in output.txt file
5) Initialisation of variable: index = 100 (some arbitrary number)
6) Reads each line from output.txt
7) Creates a zip file archive+index.zip
8) while(line != null)
9) if (sum(size of read files) is less than 64MB and extension doesn't change)
10) then
11) a) add files into zip
12) else
13) close zip
14) copy the archive from HDFS to local machine
15) index++
16) create a new zip file archive+index.zip
17) end if else
18) read next line of output.txt
19) end while
20) close zip file
21) copy zip file from local to HDFS

B. File Reading Algorithm

1) Start the program
2) Select the Read button to read from HDFS
3) Enter the small file name
4) For the first read it creates a map by looking at the hashtable.txt created by merger program
5) For later reads it checks for the match in map which requires O(1) time
6) Gets the archive index number
7) if (archive+index.zip is not in local machine)
8) copy the archive from HDFS to local machine
9) end if
10) read the small file from the zip present in local machine

B. File Sorting Algorithm

1) Read the file contents from the filename (given by user)
2) Read a line
3) while (line != null)
4) split on . (split on extension)
5) interchange extension and name
6) add this to list
7) read next line
8) end while
9) use Collection.sort (in build java function) to sort list
10) for (each list item)
11) split on extension
12) interchange extension and name
13) write into output.txt
14) end for

/subsection File Searching Algorithm

1) find the smallest key which is greater than or equal to given target (small file name)
IV. Evaluation And Results

A. Experimental Environment

The multinode cluster had 4 nodes, one master and 3 slave nodes. Each of these machines had following configurations:
1) Processor: Intel Core i7-2600 CPU @ 3.10GHz
2) RAM: 4 GB
3) HDD: 500 GB
4) Operating System: Linux
5) Version: Ubuntu 12.04 LTS
6) Java Version: 1.6.0 24
7) Hadoop: 1.2.1 version
8) Eclipse IDE: Helios
9) Network Protocol: Secure Shell All the machines were connected through ethernet.

The time taken for read and write operations was measured for both the original HDFS and the proposed solution for single node as well as for multinode cluster. A set of 1000, 2000, 4000, 6000, 8000 and 10000 files. Here we have considered a mix of text and pdf files. These files were first copied into HDFS and then read back to the local machine. The time taken to complete these operations was noted down and similar readings were obtained for the proposed solution. This was repeated three times and an average of the results was calculated and used for analysis.

B. Single-Node Cluster

1) Read Operation: The results obtained for read operation are summarized below:

```
if (smallfilename lies within the range of
  { smallfilename1 } to { smallfilename2 g })
3) return the index
4) else
5) print file doesn’t exist
6) exit program
7) end if else
```

2) Write Operation: The results obtained for write operation in single node cluster are summarized in a similar manner which is shown below:

![Fig. 3: Time taken for read operation in single node cluster](image)

![Fig. 4: Time taken for write operation in single node cluster](image)

The data obtained clearly indicates that the proposed solution is faster than the default original HDFS for read as well as write operation in single node cluster setup.

C. Multi-Node Cluster

1) Read Operation: In the following graph, we have compared the read time required in HDFS and in proposed solution for .txt files:

![Fig. 5: Time taken for reading .txt files in Multi-Node Cluster](image)

2) Write Operation: In the following graph, we have compared the read time required in HDFS and in proposed solution for .pdf files:

![Fig. 6: Time taken for reading .pdf files in Multi-Node Cluster](image)
2) **Write Operation**: The results obtained for write operation in a multinode cluster are summarized in a similar manner which is shown below:

![Comparison for Writing Files](image_url)

*Fig. 7: Time taken for Write operation in Multi-Node Cluster*

### D. READ Analysis

The first read will take more time since the map is created but the next reads take significantly lesser time as depicted in the figure below:

![Time taken for different reads in the proposed solution](image_url)

### V. Related Work

Bo Dong1, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li and Ying Li in [1] designed a way of effective storage and access pattern for large number of small files in HDFS. For storing and accessing small files, correlations between the files and locality of reference remaining among small files in the context of PPT courseware are taken into account. In the first step, they tried to merge all the correlated small files of a PPT courseware into a single big file that can effectively reduce the metadata load on the NameNode. In the second step, they introduced a concept of two-level prefetching mechanism. Applying this mechanism, the efficiency of accessing small files is improved. The experimental results finally indicate that their presented design efficiently reduces burden on the NameNode. It also improves the efficiency of storing and accessing huge number of small files on HDFS. However, it does not take into account other types of files such as .txt, .pdf, .png, .odt, etc.

Chandrasekar S, Dakshinamurthy R, Seshakumar P G, Prabavathy B and Chitra Babu in [2] proposed a solution based on the works of Dong et al., where a set of correlated files is combined, as identified by the client, into a single large file to reduce the file count. An indexing mechanism has been built to access the individual files from the corresponding combined file. Efficient management of metadata for small files helps for greater utilization of HDFS resources. However, it does not sort the files on the basis of their extension which makes it difficult while reading similar types of files.

Yang Zhang and Dan Liu [3] proposed an approach for small files processing strategy and proposes files efficient merger, which builds the file index and uses boundary file block filling mechanism. It successfully accomplishes its goal of effective files separation and files retrieval. Their experimental results clearly show that their proposed design has improved the storing and processing of huge number of small files efficiently in HDFS. However, it does not take into account file correlation mechanism which can reduce access time in HDFS. This solution can be enhanced further by effectively implementing file correlation mechanism.

### VI. Conclusions

HDFS was originally developed for storing large files. When a large number of small files are stored in it, efficiency is reduced. The approach designed in this solution could improve small files storage and accessing efficiency significantly. Files are sorted on the basis of their extensions and then merged into zip files whose size does not exceed the HDFS block size. This helps to locate any small file easily. A file read cache has been established, so that the program can read a small file quickly. The experimental results show that the approach can effectively reduce the load of HDFS and improve the efficiency of storing and accessing small files. For 10,000 small files, the writing time is reduced by 80% and the reading time is reduced by 92% on a single-node cluster while for the multi-node cluster, the percentage decrease for writing is 77% and for reading text files is 89% and for .pdf files is 15%.

### VII. Future Work

As for future work, this solution can be enhanced further to provide a more advanced file correlation framework. This framework should provide a mechanism to combine files of similar domain. Append operation can also be provided to add similar files into the existing combined files. It can also be extended for other types of file format.

### References

[1] Bo Dong1, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li, Ying Li "Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files". In proceedings of Services Computing (SCC), 2010 IEEE International Conference, Miami, FL, July 2010, pp. 65-72.


[3] Yang Zhang, Dan Liu "Improving the Efficiency of Storing for Small Files in HDFS". In proceedings of Computer Science and Service System (CSSS), 2012 International Conference,
An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS


About the Authors

**Shyamli Rao** received her B. Tech. in Computer Engineering from College of Engineering Pune and currently working in a private company for a year. She will be going to pursue M. Tech. from IIT Bombay. Her areas of interest are Cloud Computing, Parallel Computing and Data Mining.

**Amit Joshi** is an Assistant Professor of Computer Engineering and Information Technology department at College of Engineering Pune. He received M. Tech. from Dr. Babasaheb Ambedkar Technological University Lonere, Raigad. He is selected for Ph.D. Program at NIT, Trichy. His research interests include Parallel Computer Architectures, Parallel Computing using Multicore Architectures.
Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers

Abinaya Mahendiran, Srivatsan Sridharan, Sushanth Bhat and Shrisha Rao

IIIT Bangalore, 26/C Electronics City, Hosur Road, Bangalore 560 100,
Emails: abinaya.m@iiitb.org, srivatsan.s@iiitb.org, sushanth.bhat@iiitb.org, shrao@ieee.org

All the data used and generated for this work, as well as the complete source code and outputs, are available for download from https://github.com/abinayam/OSProject9C2.

We propose a classification scheme for genetic markers, to associate them with a single type or multiple types of cancer. Given multiple unknown datasets with genetic markers that are associated with particular types of cancer, we classify these genetic markers into two different classes. One class comprises the single genetic markers associated with a particular type of cancer, but with the potential to cause other cancers. The other class is comprised of multiple genetic markers each associated with a particular type of cancer, which in combination also have the potential to be associated with a different type of cancer. Use of this classification scheme would be helpful in predicting a possibility of new cancers from available data about genetic markers, and would thus have diagnostic or therapeutic value in oncology. A graphics processing unit (GPU) is used to accelerate the process of classification by making the entire classification scheme faster compared to the same process running on a CPU. A feature subset selection algorithm is used to remove irrelevant features by comparison of all the features from the dataset with the threshold file which contains the appropriate feature that relate them with the possibility of causing cancer. This work enables the classification of all given genetic markers into two categories, of which one category in particular consists of markers whose influence in causing multiple types of cancer is otherwise unknown.

Index Terms: Cancer, classification, CUDA, genetics, GPU, oncology, machine learning, prediction algorithms, supervised learning.

1. Introduction

Cancer incidence and deaths are rising worldwide as a result of the growth and aging of the human population [1]. There are certain genetic markers associated with every type of cancer. It is commonly accepted that the detailed study of these genetic markers would help unlock the mysteries of how cancers develop in human beings. With large datasets of clinical or laboratory results about genetic markers becoming available [2], they can be analyzed in order to understand the similarity and differences between different types of cancers and their correlates.

The main result of this work is to establish a method to detect otherwise unknown relationships between the occurrences of cancers with that of the genetic markers. It also aims at cross-referencing cancer data for multinomial classification of the genetic markers. This implies the classification of genetic markers causing cancer into two sets of overlapping classes: multiple genetic markers that are associated with a single type of cancer, and single genetic markers that are associated with multiple types of cancer.

One class suggests the possibility of single genetic markers associated with particular types of cancer, as having the potential to cross the threshold of causing new types of cancer in a manner that is not directly known from a bare scrutiny of the data as such. Thus, this leads to a better understanding of the association of single genetic markers with different types of cancer.

The other class indicates the possibility of multiple genetic markers that are individually associated with different types of cancer, but which in combination have the potential to cross a threshold for causing a new cancer. This class suggests that a specific set of multiple genetic markers each causing cancer, can in combination give rise to the possibility of some other type of cancer. So another benefit of this classification scheme is to predict the possibility of a new cancer in a patient if the multiple genetic markers in combination have the potential to cross the threshold value of a new type of cancer.

The use of the parallel computing power of the graphics
Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers

processing unit (GPU) makes the entire process execute in near-linear time even for large datasets.

One of the best ways to analyze such large datasets is to use a machine learning algorithm to identify multiple genetic markers that are associated with the same type of cancer, as well as identifying single genetic markers that are associated with multiple types of cancers [3]. This is the premise underlying our work.

The main thrust of our approach is in the process of classification of the genetic markers that are associated with cancers into two overlapping classes. This comprises extracting those genetic markers that have been identified in prior literature as potentially causing certain types of cancers, followed by their classification into two types of overlapping classes: one that consists of all the genetic markers that are associated with a single type of cancer; and a second that consists of single genetic markers that are associated with multiple types of cancer. By doing so, we also try to establish the correlations between the different types of cancers and their genetic markers.

In order to accomplish this task, a machine learning algorithm based on k-nearest neighbors (kNN) is used. The available raw sets of data are subjected to preprocessing before analysis. Data cleansing is carried out in order to make sure that the dataset has relevant features, i.e., common attributes among the different types of cancer, following which the data are divided into training and testing datasets.

The process of cleansing is carried out by a feature subset algorithm where attributes are analyzed for data type as well as the value each genetic marker holds. The division of the given dataset into training and testing datasets is done using a centroid algorithm where the centroid of all the given dataset points is calculated and an intercept is generated. Based on the position of each dataset point on the hyperplane, it is classified as belonging to either the training dataset or the test dataset. Approximately sixty percent of the dataset points are grouped into the training dataset and the remaining forty percent are grouped into the test dataset, as is standard [4]. The training dataset has the genetic markers along with the types of cancer they can cause. This dataset is used in the training phase. The testing dataset contains genetic markers without indication of the types of cancer they can cause. The kNN algorithm is able to predict the correlations of the types of cancers with the genetic markers given in the test dataset.

The entire work is implemented using CUDA parallel programming on a GPU system, with innovative use of supervised machine learning algorithms to classify the genetic markers. CUDA is a parallel computing platform and a programming model that enables dramatic increases in computing performance by harnessing the power of the GPU. The CUDA architecture consists of several components such as parallel compute engines, Nvidia GPUs, OS kernel-level support for hardware initialization and configuration, and a user mode driver which provides a device level API for developers. The GPU can execute large numbers of threads in parallel and operates as a co-processor with the conventional central processing unit (CPU).

Besides its many well-known applications in computer graphics, GPU computing has been applied recently in several other fields, such as biomedical engineering [5], distributed storage systems [6], medical imaging [7], pattern matching [8], processing of streaming data [9], and others.

A GPU is hierarchically split into grids, blocks and threads, providing a high degree of parallelism [10]. Each of the grids, blocks and threads has the capacity and resources to execute different code in parallel. This parallelism increases the performance of the chosen machine learning algorithm kNN. The algorithm generates an output of the classification scheme that classifies the genetic markers associated with the cancer into two different type of classes, as noted previously.

It is evident from the output that the percentage of single genetic markers causing multiple cancers is high. There are some genetic markers from the given dataset that do not fall in either of the two classes. These are regarded as outliers that do not correspond to any type of cancer. Prior work [11]–[13] has only portrayed the occurrence of the cancer given a set of genetic markers, but does not mention the correlation of the genetic markers to cancers. Such works also do not harness the parallel computing capability of the GPU. Designing the architecture over a parallelized environment has increased the performance of the kNN algorithm [14] to a significant extent.

Traditional CPU implementations of kNN take exponential time to run on huge datasets, whereas the time taken by the GPU implementation of kNN takes almost-linear time for execution. The accuracy of the system can further be improved by performing regression analysis by making the data to work as both the training and the test dataset. Regression analysis is done in order to minimize the error in the predictor model chosen. The testing dataset is further divided into subsets, of which a few sets can be used for training purposes and the remaining can be used for testing purposes in the consecutive iterations. Ultimately, all the elements of the entire dataset would have been used both as training and testing dataset. This would result in reduced prediction errors. The classification method uses fast-kNN approach for classification and cross-referencing of the genetic markers into two overlapping sets of classes. The feature subset selection algorithm is used to extract only the relevant feature by comparing the input genetic marker dataset with the threshold file. The threshold file contains only those parameters that are relevant for this classification process. The feature subset selection algorithm compares the input genetic marker dataset with the parameters of the threshold file, and removes all unnecessary features or irrelevant parameters from the classification process.

To summarize: this work proposes a method for classification of genetic markers into two sets of overlapping classes. The current approaches [13] [11] [12] [15] predict only the occurrence of a cancer from the given genetic marker dataset, but we propose a classification method to correlate different sets of genetic markers and associate them with either a single or multiple types of cancer. This method will benefit end users (patients and oncological diagnosticians) in
2.2 Our System

- By predicting the possibility of a new type of cancer from the patient’s genetic marker dataset. This happens by correlating different genetic markers that are known to cause cancers in patients to predict the possibility of the same in combination to cross a threshold value of causing a different type of cancer. This has obvious diagnostic implications.
- By predicting the possibility of a new cancer from a single genetic marker in a patient. This happens by exploring the possibility of the association of a single genetic marker that could cause cancer in a patient’s body, with a different type of cancer, so that a patient with that marker may also carry a risk for another cancer that was not previously known.
- The power of parallel computing using the GPU is harnessed to make the execution of the classification method much faster in time even for large datasets. The classification method uses a fast-kNN approach for rapid classification and association of the genetic markers into two overlapping sets of classes. The relevant features are extracted from the genetic marker dataset using a feature subset selection algorithm where the relevant feature is extracted by comparison with the parameters in the threshold file.

2. CLASSIFICATION OF GENETIC MARKERS

2.1 Existing Systems

GenePattern [16] [17] is a powerful genomic analysis platform that provides access to more than 230 tools for gene expression analysis, proteomics, SNP analysis, flow cytometry, RNA-seq analysis, and common data processing tasks. A web-based interface provides easy access to these tools, and allows the creation of multi-step analysis pipelines that enable reproducible in silico research. The features of the GenePattern tool are, in brief, the following: Genes that are distinct in different phenotypes can be chosen; supervised and unsupervised learning algorithms can be used to predict the class to which a gene sequence belongs; and there is a provision to convert data from one file format to another. The tool provides differential analysis and gene marker selection. The supervised learning algorithms used by the tool can predict the correct classes. Using unsupervised learning, a biologically relevant unknown taxonomy is identified by a gene expression signature.

Other existing systems, especially those that are used for breast cancer [18]–[20] recognition, mainly deal with the question of detecting whether some given genetic markers would cause a particular type of (breast) cancer or not. There is a need to develop a system that could identify particular genetic markers as having the potential to cause multiple types of cancer rather than just one, and would also correlate different genetic markers to jointly infer a risk of cancer.

2.2 Our System

With large sets of raw data being available, it is worthwhile to come up with a machine-learning algorithm that would identify multiple genetic markers that are associated with the same type of cancer, as well as single genetic markers that are associated with multiple types of cancers. From a given set of genetic markers, features are extracted by applying a feature extraction algorithm. Of all the obtained features, only the relevant features are taken into consideration by a feature subset selection algorithm. Once this is done, the given data are partitioned into training and test data using a data partitioning algorithm. A supervised learning model using the kNN algorithm is implemented to classify the given genetic markers into two classes: a class that associates a single genetic marker with multiple cancers, and a class that associates multiple genetic markers with a single cancer. The system designed gives the correlation between the cancers and the genetic markers.

There are a few prediction algorithms based on decision tree induction and bayesian classifiers [21], that help us in identifying and predicting the occurrence of cancer from the given dataset of genetic markers.

2.2.1 Parallelism Using GPU

When the dataset is huge and complex, solving the machine learning problem becomes computationally expensive. Specifically, the time to run optimization algorithms increases drastically with increasing dataset sizes. So, parallelism is a good way to speed up computations, and GPU computing is a good way to achieve the same. A GPU has a large number of Arithmetic and Logic Units (ALUs) as compared to the traditional CPU. As a result, it can do large amounts of complex computations at a faster rate than CPUs. GPUs are effective at performing rapid calculations due to their pipeline, which can perform multiple computations at the same time. A CPU can also do some parallel computation, but it is limited by the number of cores available. In our system, the Nvidia [10] [22] Tesla GPU has been used, in which there are 128 cores divided into 8 clusters consisting of 16 cores each. Each core is capable of handling multiple threads. A number of threads can be assigned according to the records that must be processed in parallel. Multiple threads can be executed in a single block, and each thread is identified by the block number and the thread index. The collection of blocks is represented by a single grid. This architecture gives the ability to process massive amounts of data in parallel, thus reducing the computational time to a great extent. Parallelization through GPU can be done in the following two ways, learning and prediction.

2.2.2 Learning

The training dataset is further divided into several parts based on the number of cores in the GPU. For example, if there are 4 GPUs with 512 cores, then the training dataset is divided into 512 parts, and is fed into the cores simultaneously during the training phase in order to speed up the computation. Each core acts on the dataset, processes it and produces the results. The individual results are then summed up and taken as the final output of the training.
phase of the algorithm. The phase of learning takes as the input the preprocessed data about the genetic markers that cause cancers, and produces two different subsets—the test and the training datasets. These training and test datasets are generated by the process of cross-validation in order to obtain a better data partition.

2.2.3 Prediction

In the prediction phase, the test dataset is also divided into subsets based on the number of cores in the GPU. Then the individual subsets are processed by the cores simultaneously and the results of the prediction algorithm are obtained. The final outcome of the prediction phase classifies the given test data into one of the two classes, either the class of multiple genetic markers that are associated with a single type of cancer, or the class of genetic markers that are associated with multiple types of cancer. The prediction phase includes weighted voting to determine the correlation between the training samples and the testing samples. Based on the results, the test samples are classified into either of the two classes. These classified training samples are then used by the algorithm for learning, which in turn helps improve the accuracy of the supervised prediction model.

Supervised learning algorithms take a set of input data along with the associated class labels, and build a predictor model based on the selected features, to produce relevant responses by assigning appropriate class labels to new sets of data when input to the model. As a part of the data preprocessing phase, there are some techniques that are used to handle specifically the missing data that would be most appropriate for the machine learning algorithm. One of the techniques is the method of ignoring instances with unknown feature values. This method ignores instances which have at least one unknown feature value. The value of the feature that occurs most often is selected to be the value for all the unknown features.

The architecture of the our system is as shown in Fig. 1. Present systems only predict the occurrence of specific cancers given genetic markers, but our classification method deals with the correlation of the genetic markers with single or multiple types of cancers.

Table 1 depicts the differences between the existing approaches and our genetic marker classification system.

3. Data Preprocessing and Extraction

3.1 Data Sources

The datasets used as inputs for this classification process contain parameters related only to the genetic markers, and do not feature any personally identifiable or other sensitive information. Some parameters present in those input datasets may also not be useful, and some parameters might be missing values. So to remove redundant parameters and to preprocess the entire input dataset, the use of a threshold file becomes necessary.

The threshold file contains parameters that are relevant for classification. It contains genetic markers causing cancer, the respective threshold values that when crossed by a genetic marker are known to cause cancer in humans, and the respective value associated with each genetic markers. For instance, considering the colon cancer dataset, it contains parameters labeled “Tumor n,” where n represents any number. These labels note particular tumors that are associated with the genetic markers causing cancers, and some normal (not cancer-causing) genetic markers too. The
threshold file contains only the genetic markers (labelled “Tumor”) that cause cancer.

1) Colon Cancer Dataset from The University of Edinburgh-School of informatics [23]: This dataset contains expression levels of 2000 genetic markers taken in 62 different samples. For each sample it is indicated whether it came from a tumor biopsy or not. This is used in our work by treating the genetic markers as records with 62 attributes embodied. The size of the dataset holding the genetic markers is 1.9 MB data with 529 KB names and 207 bytes labels.

2) Lung Cancer Dataset - Broad Institute [24]: The lung cancer dataset contains expression levels of 7129 genetic markers taken over 72 different samples. The size of the dataset holding the genetic markers is 3.8 MB, with 72 samples.

3) Breast Cancer Dataset - Broad Institute [25]: The breast cancer dataset contains expression levels of 1212 genetic markers taken over 55 different samples. Each sample data, there is an associated class label which indicates whether the dataset is associated with cancer or not. The threshold file contains the class labels of those genetic markers that are associated with breast cancer. The size of the dataset holding the genetic markers is 1.5 MB, with 55 different samples. The threshold file is 0.7 MB holding only those genetic markers that are associated with the cancer. This threshold file is used to extract the relevant parameters from 1212 genes taken over 55 samples.

4) Prostate Cancer Dataset - Broad Institute [25]: The prostate cancer dataset contains expression levels of 5896 genetic markers taken over 78 different samples. The size of the dataset holding the genetic marker is 2.68 MB with 78 samples.

   The entire collection of obtained cancer data is merged into a single compound dataset. This compound dataset is what we use for our work.

3.2 Pre-processing Module

The data pre-processing module deals with the following:

1) Data format [2] [26]: The raw data that are available are in different file formats. To implement any data mining technique, the dataset has to be of a single format. The file format chosen for the compound dataset is Attribute-Relation File Format (ARFF) [27], which is an ASCII text file format.

2) Data cleansing [28]–[30]: If the dataset has any features that holds irrelevant values, i.e., null or non-numeric values, those features are discarded before it is further processed by the feature subset selection algorithm.

3.3 Feature Subset Selection Algorithm

Feature subset selection is the process of identifying and removing irrelevant and redundant features [31], [32]. This reduces the dimensionality of the data and enables learning algorithms to operate faster and more effectively. Generally, features are characterized as relevant or irrelevant. The former are features that have an influence on the output, and whose roles cannot be subsumed by the rest. Irrelevant features are those with no influence on the output, and whose values are generated at random for each example.

Feature subset selection can also be thought of as the process of identifying and removing possible irrelevant and redundant features [33], for the purpose of reducing the dimensionality of the data and enabling learning algorithms to operate faster and more effectively.

In our case, the compound dataset which is in a common format contains genetic markers for different types of cancer. Each type of cancer may have a different set of features associated with it. In order to make sure that all the classes of cancer have only common features, the relevant features are selected by the feature subset selection algorithm [34]–[36]. The actual data are read, and the data types of the chosen features are checked. If there are any mismatches in the data types even when the features referred to are the same, then those features are discarded. After this second filtering step, the relevant features are selected and written to a file along with the associated output labels. For each feature, the feature names and data types are compared from the data file and the threshold file. If they are the same, then it is added to the feature set. After this filtering step, the relevant features are selected and written to a file along with the associated output labels.

The main aspects of the feature subset selection algorithm are as follows.

3.4 Redundant Features

A redundancy exists whenever a feature can take the role of another. Feature selection algorithms in general have two components [37], a selection algorithm that generates proposed subsets of features and attempts to find an optimal subset, and an evaluation algorithm that determines how good a proposed feature subset is, returning some measure of goodness to the selection algorithm. However, without a suitable stopping criterion, the feature selection process may run exhaustively or forever through the space of subsets. If

![Table 1: Comparison of the proposed classification system with the state of the art](https://example.com/table1.png)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation among Cancer Types</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Processing</td>
<td>Serial</td>
<td>Parallel</td>
</tr>
<tr>
<td>Processor</td>
<td>CPU</td>
<td>GPU</td>
</tr>
<tr>
<td>Prediction Time</td>
<td>Exponential</td>
<td>Linear</td>
</tr>
</tbody>
</table>
the addition or deletion of any feature does not produce a better subset, then it can be considered as a stopping criterion. Ideally, feature selection methods search through the subsets of features and try to find the best one among the possible candidate subsets according to some evaluation function.

<table>
<thead>
<tr>
<th>Algorithm 1: FEATURE SUBSET SELECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data: dataFileFeatures[], thresholdFileFeatures[]</td>
</tr>
<tr>
<td>Result: Common feature set features[]</td>
</tr>
<tr>
<td>1 $k$ = 0;</td>
</tr>
<tr>
<td>2 features[] = null;</td>
</tr>
<tr>
<td>3 for $i$ from 0 to dataFileFeatures.size()-1 do</td>
</tr>
<tr>
<td>4 for $j$ from 0 to thresholdFileFeatures.size()-1 do</td>
</tr>
<tr>
<td>5 if (dataFileFeatures[$i$] = thresholdFileFeatures[$j$]) and (getDataType(dataFileFeatures[$i$]) = getDataType(thresholdFileFeatures[$j$])) then</td>
</tr>
<tr>
<td>6 $features[k + 1] \leftarrow cols1[j]$;</td>
</tr>
<tr>
<td>7 break;</td>
</tr>
<tr>
<td>8 end</td>
</tr>
<tr>
<td>9 end</td>
</tr>
<tr>
<td>10 end</td>
</tr>
</tbody>
</table>

**Explanation of Algorithm 1**

(i) For each feature, the feature names and data types are compared from the data file and the threshold file (line 5).

(ii) If they are the same then it is added to the feature set (line 6).

(iii) After this filtering step, the relevant features are selected and written to a file along with the associated output labels.

However, this procedure is exhaustive as it tries to find only the best one. It may be too costly and practically prohibitive, even for a medium-sized feature set size. Other methods based on heuristics or random search methods attempt to reduce the computational complexity while compromising on performance. Filter methods are independent of the inductive algorithm, whereas wrapper methods use the inductive algorithm as the evaluation function. Stopping functions [38] [39] can be used, with criteria based on the following parameters: distance, information, dependence and consistency.

### 3.5 Partitioning of Dataset

The dataset in the ARFF format is then segregated into two subsets, the test dataset and the training dataset. The two are later used for the classification of the genetic markers into two different classes.

(i) **Training Dataset:**

The training dataset contains the input data together with the correct/expected output. This dataset is usually prepared by collecting some data in a semi-automated way. It is important that there be expected output labels associated with the dataset, because it is used for supervised learning. Approximately 60% [4] of the original dataset is to be used for the training dataset.

(ii) **Test Dataset:**

The remaining data are taken for the test dataset. These are the data that validate the underlying model. For the purpose of cross-validation, the test dataset is in turn divided into further subsets which can then be used to test the accuracy of the chosen model effectively.

### 3.6 Centroid Generation

Algorithm 2 calculates the centroid among all the distributed dataset points and stores the coordinates as (centroid\_x, centroid\_y) points in the space. Random coordinates are generated for each sample row data, and starting values are assigned to arrays max \_x[], min \_x[], max \_y[], min \_y[]. For each sample data, we assign maximum value of random coordinates to max \_x and max \_y and minimum values to min \_x and min \_y. We assign the average of min \_x, max \_x and min \_y, max \_y to centroid\_x and centroid\_y respectively. This part of the data preprocessing algorithm calculates the centroid among all the distributed dataset points and stores the coordinates as (centroid\_x, centroid\_y) points in the space.

![](Feature_Reduction.png)

**Fig. 2:** Working of the feature subset selection algorithm.

![](Percentage_of_red.png)

**Fig. 3:** Percentage of feature reduction achieved.


Algorithm 2: CENTROID GENERATION

Data: \( x_{\text{coordinate}}, y_{\text{coordinate}}, \max_x, \min_x, \max_y, \min_y \)
Result: centroid_x, centroid_y

\[
\begin{align*}
&\text{max}_x \leftarrow \text{rand}_x \_\text{coordinate}[0]; \\
&\text{max}_y \leftarrow \text{rand}_y \_\text{coordinate}[0]; \\
&\text{min}_x \leftarrow \text{rand}_x \_\text{coordinate}[0]; \\
&\text{min}_y \leftarrow \text{rand}_y \_\text{coordinate}[0]; \\
&\text{begin} \\
&\text{for } i = 1 \text{ to } \text{SampleDataRowCount} - 1 \text{ do} \\
&\quad \text{if } \text{rand}_x \_\text{coordinate}[i] \geq \text{max}_x \text{ and } \\
&\quad \quad \text{rand}_y \_\text{coordinate}[i] \geq \text{max}_y \text{ then} \\
&\quad \quad \quad \text{max}_x \leftarrow \text{rand}_x \_\text{coordinate}[i]; \\
&\quad \quad \quad \text{max}_y \leftarrow \text{rand}_y \_\text{coordinate}[i]; \\
&\quad \text{end} \\
&\quad \text{if } \text{rand}_x \_\text{coordinate}[i] < \text{min}_x \text{ and } \\
&\quad \quad \text{rand}_y \_\text{coordinate}[i] < \text{min}_y \text{ then} \\
&\quad \quad \quad \text{min}_x \leftarrow \text{rand}_x \_\text{coordinate}[i]; \\
&\quad \quad \quad \text{min}_y \leftarrow \text{rand}_y \_\text{coordinate}[i]; \\
&\quad \text{end} \\
&\text{end} \\
&\text{centroid}_x = (\text{max}_x + \text{min}_x) / 2; \\
&\text{centroid}_y = (\text{max}_y + \text{min}_y) / 2; \\
&\text{end}
\]

Explanation of Algorithm 2

(i) Random coordinates are generated for each sample row data, and starting values are assigned to arrays \( max_x, min_x, max_y, min_y \) (lines 1 to 4).
(ii) For each sample data, maximum values of random coordinates are assigned to \( max_x \) and \( max_y \) and minimum values to \( min_x \) and \( min_y \) (lines 6 to 15).
(iii) The average of \( min_x, max_x, min_y, max_y \) are assigned to centroid_x and centroid_y respectively (lines 16, 17).

3.7 Intercept Generation

Algorithm 3 generates a random point on either the x-axis or the y-axis, and correspondingly generates its intercept on the alternate axis using the principle of parallelogram, thereby dividing the dataset into training and testing subsets. After generating random coordinates \( \text{rand}_x \) and \( \text{rand}_y \), the greater of the two values is picked and assigned to the coordinate, and the flag is set accordingly. Based on the property of similar triangles, it finds the coordinate2 using the centroid and the known coordinate1.

Algorithm 3: INTERCEPT GENERATION

Data: \( \text{rand}_x, \text{rand}_y, \text{centroid}_x, \text{centroid}_y \)
Result: coordinate1, coordinate2

\[
\begin{align*}
&\text{begin} \\
&\quad \text{begin} \\
&\quad \quad \text{if } \text{rand}_x > \text{rand}_y \text{ then} \\
&\quad \quad \quad \text{coordinate1} = \text{rand}_x; \\
&\quad \quad \quad \text{flag0} = 1; \\
&\quad \quad \text{else} \\
&\quad \quad \quad \text{coordinate1} = \text{rand}_y; \\
&\quad \quad \quad \text{flag1} = 1; \\
&\quad \text{end} \\
&\quad \text{if } \text{flag0} == 1 \text{ then} \\
&\quad \quad \text{coordinate2} = ((\text{coordinate1} * \text{centroid}_y) / \text{centroid}_x) - \text{centroid}_y; \\
&\quad \text{end} \\
&\quad \text{if } \text{flag1} == 1 \text{ then} \\
&\quad \quad \text{coordinate2} = ((\text{coordinate1} * \text{centroid}_x) / \text{centroid}_y) - \text{centroid}_x; \\
&\quad \text{end} \\
&\text{end}
\]

Explanation of Algorithm 3

(i) Random coordinates \( \text{rand}_x \) and \( \text{rand}_y \) are generated and checked to see which is greater. The greater value is assigned to the coordinate and the flag set accordingly (lines 3 through 9).
(ii) Based on the property of similar triangles, coordinate2 is found using the centroid and known coordinate1.

3.8 Training and testing dataset division

Algorithm 4 counts the points of the training and test datasets on either side of the intercept. The line joining the intercepts coordinate1 and coordinate2 is the boundary for dividing the dataset into training and test subsets. If the count of the coordinates generated during the centroid generation algorithm above this boundary is high, we consider the rows corresponding to these coordinates as the training dataset. The rows corresponding to the rest of the coordinates below the line are the test dataset and vice-versa. The next step of Algorithm 4 is writing the test and training dataset values to the corresponding files. The set with most number of points is taken as the training dataset and the set with fewer points is taken as the test dataset. The corresponding row values of the indices are stored in training data[i] and test data[i], and written to training data[i].arff test data[i].arff files.

Table 2: Achieving feature reduction using feature subset algorithm

<table>
<thead>
<tr>
<th>Types of Cancer</th>
<th>Initial Number of Features</th>
<th>Final Number of Features</th>
<th>% of Feature Reduction Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>34</td>
<td>24</td>
<td>29.41</td>
</tr>
<tr>
<td>Prostate Cancer</td>
<td>48</td>
<td>26</td>
<td>45.83</td>
</tr>
<tr>
<td>Lung Cancer</td>
<td>35</td>
<td>21</td>
<td>40.00</td>
</tr>
<tr>
<td>Colon Cancer</td>
<td>33</td>
<td>27</td>
<td>18.18</td>
</tr>
</tbody>
</table>
Supervised learning algorithms take a set of input data along with the associated class labels, and build a predictor model [40] based on the selected features, and produce relevant responses by assigning appropriate class labels to new sets of data when input to the model. k-NN is an instance-based learning algorithm which classifies the objects based on the closest k training examples in the feature space. The predictor function is only approximated locally [41] and all computation is deferred until classification [42].

In the present work, each genetic marker is a vector in n dimensions that represents the features of a particular type of cancer. The training instances in n dimensions are mapped to a two-dimensional vector (which represents the output of the given predictor model), where one dimension represents the class of genetic markers that are associated with a single type of cancer, and the other dimension represents the genetic markers that are associated with multiple types of cancer.

When a test instance (which is also a vector in n dimensions) is given to the predictor model, based on the value of k, the euclidean distance between the test instance and the k-nearest training instances is calculated [14].

The euclidean distance \( D \) between any two points in \( n \) dimensions, say \( X_i \) and \( X_j \) where \( i \) and \( j \) are the \( i^{th} \) and the \( j^{th} \) point in the feature space, is given by:

\[
D = \sqrt{(X_i - X_j)^2}
\]

The choice of the parameter \( k \) is very critical as it determines the accuracy of the predictor model. The larger the value of \( k \) [14], the smaller is the effect of noise on classification. Here, “noise” refers to irrelevant features in the dataset. On the other hand, if the value of \( k \) is too large, the predictor model always assigns the dominant class label to the test instance. So, the value of \( k \) has to be chosen with utmost care.

For the problem of classifying the genetic markers into two different classes, the value of \( k \) must be odd. Once the \( k \)-nearest training instances are found, their class assignments are used to predict the class for test instances using weighted voting.

### Algorithm 5: Mean Calculation - Training Dataset

**Input:** trainingData[ ], feature_set, no_of_features, no_of_rows

**Result:** actual[ ] - set of means of all the features

```plaintext
sum=0;
actual[] = NULL;
for k = 0 to no_of_features-1 do
    sum = sum + trainingData[i].NumericAttributes[feature_set[k]]
end
actual[k] = sum / no_of_rows;
```

### Explanation of Algorithm 4

(i) The line joining the intercepts \( \text{coordinate1} \) and \( \text{coordinate2} \) is the boundary for dividing the dataset into training and test subsets.

(ii) If the count of the coordinates above this boundary generated during centroid generation is high, we consider the rows corresponding to these coordinates as the training subset.

(iii) The rows corresponding to the coordinates below the line are the test subset, and vice versa.
Explanation of Algorithm 5
(i) This algorithm finds the mean of each feature/attribute in the training dataset.
(ii) The sum of all numerical values of every feature is found (line 5).
(iii) Then the mean of the feature set is found and stored (line 7).

Algorithm 6: kNN Classification Algorithm
Data: This algorithm finds the k-nearest neighboring samples in the test dataset
Input: TestData[samples][attributes]
Result: Returns the mean of the k-nearest neighboring samples of testdata for a particular attribute

```plaintext
1. calculateEuclideanDistance(predict_attr);
2. findkNN();
```

Explanation of Algorithm 6
(i) The find kNN function calls the calculateEuclideanDistance function for each attribute of the test data (line 3).
(ii) The calculateEuclideanDistance function computes the euclidean distance of the test data from the mean of training data using the formula:

\[ D = \sqrt{(X_i - X_j)^2} \] (line 8).
(iii) Then the test data is sorted based on these distance values (line 5).
(iv) The mean of kNN which is the predicted output value of this algorithm is found (line 9).
(v) This algorithm has been implemented using CUDA-C, which uses parallel execution of threads.
(vi) Because calculation of euclidean distance is exclusive to one particular attribute, we can have number of the threads equal to the number of attributes, which run in parallel in order to find the euclidean distances.
(vii) This strategy improves the running time of the algorithm for large data.

Algorithm 7: Correctness of kNN
Input: TestData,k_value,actualMean
Result: Verifies the efficiency of the prediction of kNN algorithm and writes the data coordinates to an output file

```plaintext
1. kMean = kNN(TestData,k_value);
2. if (||actualMean - kMean||)/actual < 0.4 then
3.   writeToFile(outputfile, result, actual, cancer_type);
4.   /* Writes kMean, actualMean, cancer_type to the output file */
```

Explanation of the correctness of Algorithm 7
(i) This algorithm evaluates the efficacy of the kNN algorithm.
(ii) The mean evaluated by kNN algorithm is compared with the actual mean of the training dataset (line 2).
(iii) If there is less than 40% deviation from the actual mean then both means, along with cancer type, are written to a file (line 3).
(iv) These means represent coordinates for the x – y plot with x-axis representing actual mean of the training data and y-axis representing the calculated mean of the kNN algorithm.
(v) Also the cancer type is used to depict the different types of cancer in the plot.

4. Performance and Results
The model was tested with different cancer datasets in order to compare the performance while using GPU and CPU. The programs were made to run with varying cancer data files and threshold files. The results obtained are plotted to obtain the degree of correlation among the cancer dataset and time taken for processing them using both GPU and CPU. Figure 4 and Table 4 show the performance comparison of the developed model on both CPU and GPU processors, with varying number of cancer data records.

Function calculateEuclideanDistance(predict_attr)
```plaintext
1. dist=0;
2. for i = 0 to samples - 1 do
3.   dist=0; for j = 0 to attributes - 1 do
4.     if j == predict_attr then
5.       dist = dist + square(TestData[i][j] - TrainingMean[j]);
6.     end
7.   end
8.   TestData[i].distance = sqrt(dist);
9. end
```

Function findkNN
```plaintext
1. sum=0;
2. for every attribute in attributes do
3.   calculateEuclideanDistance(attribute);
4. end
5. sort(TestData);
6. for i = 1 to k do
7.   sum = sum + TestData[i].NumericAttributes[predict_attr];
8. end
9. mean = sum/k;
10. return mean;
```
The strategy improves the running time of the algorithm. Because calculation of Euclidean distance is exclusive order to compare the performance while using GPU and CPU, the number of records increases, the time taken (in seconds) also increases exponentially. So this is a significant finding, as the cancer dataset to be processed might easily be in the thousands.

The speed could be improved by using the GPU, which takes linear time to complete the process of classification for a large number of records. This is seen in Fig. 4 which shows the time taken by the CPU to process the records is almost exponentially growing with the increase in numbers of records, while the time taken by the GPU seems to almost remain constant as the number of records increases. (It actually rises slowly, linearly, but the rise is not visible in the graph.) The main point behind the comparison of the performance of CPU and GPU lies in the fact of the demonstration of power of parallel processing for the classification algorithm, i.e., it could help in reducing the time complexity involved in solving the problems involved.

Fig. 5 pertaining to the entire process of cancer dataset classification scheme is thus generated. The main objective is to demonstrate the percentage of genetic markers that are responsible for causing multiple cancer from the given dataset and the percentage of multiple genetic markers in combination that cause single types of cancer. The outliers may be as a result of wrong classification, but are significantly less which indicates that the percentage error as result of applying proposed system is quite low. The significance of the plot in Fig. 5 is as follows.

(i) The two tall bars in the graph represent two different types of cancer whereas the short bar represents the outlier – the wrong classification.

(ii) The two tall bars represent the genetic markers associated with each type of cancer.

The main intention of this system is not only to classify the genetic markers into two classes, but also to speed up the entire process of classification. This is seen to be achieved using the GPU, while in case of a CPU, if the number of records increases, the time taken (in seconds) also increases exponentially. So this is a significant finding, as the cancer dataset to be processed might easily be in the thousands.

Fig. 4: Performance Comparison – GPU vs CPU

It is clear that the performance of GPU is better than the performance of CPU as the number of records increases.

The result of the classification produced by kNN algorithm is plotted as a bar diagram. The output suggests the presence of two different classes. One class represents the genetic markers that can cause multiple types of cancer, and the other is a class that represents a possibility of multiple genetic markers from different cancer data sets with the potential, in combination, to cause a single type of cancer. The genetic markers that do not belong to either class are outliers, i.e., they are not significant enough, given the data available, to be correlated with any type of cancer. So the benefit of this classification scheme is to predict the possibility of a new cancer in a patient if the multiple genetic markers in combination have the potential to cross the threshold value of a new type of cancer.

Table 3 represents the different classes under which the genetic markers fall. Class label 0 shows that it is an outlier, 1 shows that the genetic markers are associated with a single type of cancer and 2 that it is associated with multiple types of cancer. Thus the correlations between the genetic markers and the cancers could be easily established in this process. The main point to note from Table 3 is that in almost all types of cancer data that have been tested by this system, all the data yield almost similar results. This means that the percentage of genetic markers associated with multiple cancers is higher compared to that of the multiple genetic markers causing single types of cancer, and the percentage of outliers. The percentage of outliers, which amounts to is quite low, thus indicating the efficacy of the system.

Table 4 illustrates the comparison of the performance of the proposed algorithm on GPU and CPU. In a GPU, there are many cores working in parallel and the power of parallel programming is thus achieved to a greater extent.
(iii) It is evident from the graph that the bar indicating markers associated with multiple cancers is the highest among all in the graph. So, for our given dataset that deal with four types of cancers, there are more genetic markers with the ability to cause multiple cancers.

(iv) The genetic markers that are wrongly classified that lie outside the class boundaries are outliers in the data, i.e., they are not significant enough to be correlated with any type of cancer.

(v) The rest are the genetic markers, that cause just one type of cancer. This single type of cancer mostly comes from the type of cancer dataset that the given genetic markers belongs to.

5. Conclusion

Cross-referencing genetic markers for cancer using the kNN has been developed on an Nvidia GeForce GPU. The power of parallel programming has been exploited by the use of a GPU, where the performance of the predictor model can be improved considerably by simultaneous execution of multiple threads as compared to the much slower CPU execution. This current model works only on textual information, but can be further enhanced to process information from image data or normalized micro-array data.

There are possible future preventive, diagnostic, or therapeutic applications of this work, and such correlations among genetic markers can also of course be attempted for diseases other than cancers.

More immediately, it surely would be helpful to identify those genetic markers that could potentially have higher chances of causing multiple types of cancers if they are present in the human body, and correlate the same with other risk factors; likewise, a person with multiple genetic markers associated with a specific cancer may well be at higher risk than someone with just one or none of those markers. Such and other uses of the analytical and computational tools we have suggested could provide a new weapon in the fight against dangerous diseases that take millions of lives yearly.

Acknowledgments

The authors would like to thank Bhavani Kakarla and Sana Javed for their help in gathering and analyzing data from various sources.

References


Table 3 : Association of genetic markers with single or multiple cancers

<table>
<thead>
<tr>
<th>Type of Cancer</th>
<th>Genetic Markers associated with Single Cancer</th>
<th>Genetic Markers associated with Multiple Cancer</th>
<th>% of Outliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>33.33</td>
<td>59.45</td>
<td>7.22</td>
</tr>
<tr>
<td>Prostate Cancer</td>
<td>45.35</td>
<td>49.23</td>
<td>5.42</td>
</tr>
<tr>
<td>Lung Cancer</td>
<td>38.48</td>
<td>53.33</td>
<td>8.19</td>
</tr>
<tr>
<td>Colon Cancer</td>
<td>43.27</td>
<td>49.68</td>
<td>7.05</td>
</tr>
</tbody>
</table>

Table 4 : Performance Comparison Between CPU and GPU

<table>
<thead>
<tr>
<th>S.No</th>
<th>Number of Records</th>
<th>Time Taken in CPU (in secs)</th>
<th>Time Taken in GPU (in secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>400</td>
<td>5.852</td>
<td>1.515</td>
</tr>
<tr>
<td>2.</td>
<td>600</td>
<td>14.166</td>
<td>1.522</td>
</tr>
<tr>
<td>3.</td>
<td>800</td>
<td>26.445</td>
<td>1.54</td>
</tr>
<tr>
<td>4.</td>
<td>1000</td>
<td>38.962</td>
<td>1.562</td>
</tr>
<tr>
<td>5.</td>
<td>1200</td>
<td>52.775</td>
<td>1.568</td>
</tr>
<tr>
<td>6.</td>
<td>1400</td>
<td>78</td>
<td>1.579</td>
</tr>
</tbody>
</table>


About the Authors

Abinaya Mahendiran received her Bachelor of Technology degree in information technology from Amrita School of Engineering, India, and her Master of Technology degree in information technology from IIIT-Bangalore. She has been with Cognizant Technology Solutions and is presently with Cisco Systems. Her research interest is in machine learning, with a particular focus on deep learning.

Srivatsan Sridharan received his Bachelor of Engineering degree in computer science and engineering from Sri SaiRam Engineering College, affiliated with Anna University, Chennai, India and his Master of Technology degree in information technology from the International Institute of Information Technology - Bangalore (IIIT-Bangalore), a graduate school of information technology in Bangalore, India. He is presently with Cisco Systems.

Sushanth Bhat received his Master of Technology degree in information technology from IIIT-Bangalore, and his Bachelor of Engineering degree in computer science from NMAM Institute of Technology, Nitte, India. He has been with Cisco Systems, and is now with e-commerce major Flipkart.com.

Shrisha Rao received his Ph.D. in computer science from the University of Iowa, and before that his M.S. in logic and computation from Carnegie Mellon University. He is an associate professor at IIIT-Bangalore. His research interests are in distributed computing, specifically algorithms and approaches for concurrent and distributed systems, and include solar energy and microgrids, cloud computing, energy-aware computing (“green IT”), and demand side resource management. Dr. Rao is a senior member of the IEEE, and a member of the ACM, the American Mathematical Society, and the Computer Society of India.
Letter to the Editor

CORRECTIONS

My paper “Bidirectional Heuristic Search based on Error Estimate” (Article 7) was published in your CSI Journal of Computing Vol. 2, No. 1-2, 2013. I observe that the assumption “(iv) $h(s) = h(t)$” may be removed from the Section 2. Removing this assumption will not materially change any examples, theorems, proofs, algorithm, results and conclusion of the paper. In the original paper, I assumed a constant $h_0 = h(s) (= h(t))$. Now, I have taken two constants $h_0$ and $h_\infty$ such that $h_0 = h(s)$ and $h_\infty = h(t)$. This change will enhance the algorithm and the paper. I am requesting you to accommodate such changes if possible. It will have great value to me.

The following changes are needed in the original paper.

In Section 2:

(Page S1:59 left column) in Definition 2.1: Replace “$h_0$” with “$h_1(s)$” in two places.

(Page S1:59 right column) in Definition 2.2: Replace “$h_0$” with “$h_1(t)$” in two places.

(Page S1:59 right column) under we assume the following: Remove “(iv) $h_0(s) = h_1(t)$”.

(Page S1:60 left column) in Table 1: Replace “$h_0$” with “$h_0(t)$” in two places.

(Page S1:60 left column) in Theorem 2.1:

Replace “$FE_1(t) = BE_1(t) = FE_2(s) = BE_2(s)$” by two lines as follows

“$FE_1(t) = BE_2(s)$”

“$FE_2(s) = BE_1(t)$”

Replace “$2[c(P)-h_0]$” with “$2[c(P)-h_0(s)+h_0(t)]$”.

(Page S1:60 left column) in Corollary 2.1: Replace “$h_0$” with “$\frac{1}{2}(h_0(s)+h_0(t))$”.

(Page S1:60 right column) below Fig. 3: Replace “$h_0$” with “$h_1(s)$” in three places in three separate lines.

In Section 3

(page S1:61 left column) in third line in the algorithm BAE*:

Replace “$h_0 \leftarrow h_1(s) (= h_1(t))$” between Set and comma with “$h_0 \leftarrow h_1(s), h_\infty \leftarrow h_1(t)$”.

In Step 3 of the algorithm, in last but one line: Replace “$h_0$” with “$(h_0 + h_\infty)/2$”.

In Step 5 of the algorithm, in last but one line: Replace “$h_0$” with “$h_\infty$”.

In Step 6 of the algorithm, in last line: Replace “$h_0$” with “$h_\infty$”.

(page S1:61 right column) in two lines above the last line: Replace “$h_0$” with “$h_\infty$” in two places.

(page S1:62 right column) in Definition 3.3: Replace “$2g_1(n) + h_1(n)-h_2(n) \leq 2h_1(s)-h_1(s)$” with “$TE_1(n) \leq TE_1^*(t)$”.

(page S1:62 right column) in Definition 3.4: Replace “$2g_2(n) + h_1(n)-h_2(n) \leq 2h_2(s)-h_2(t)$” with “$TE_2(n) \leq TE_2^*(s)$”.

(page S1:62 right column) in last but two lines: Replace “$h_0$” with “$h_\infty$”.

In Grossary of Notation

(page S1:64 right column) In the Grossary of Notation last but eight lines:

Replace the line “$h_0 = h_1(s) (= h_1(t))$” by two lines as follows

“$h_\infty = h_1(s)$”

“$h_\infty = h_1(t)$”

Samir K Sadhukhan
Indian Institute of Management Calcutta
Kolkata 700104, India, samir@iimcal.ac.in.
19/8/2014
The best paper in each volume of the CSI Journal of Computing will be given an award in one of the CSI sponsored conferences.

The details will follow.
Contents

Article R6 8 Pages  Impact, Team Size, Collaboration, and Epochs Stories from Four Decades of Software Engineering Research  
Manju. V. C., Dr. Sasi Kumar

Article R7 6 Pages  An Efficient Approach for Storing and Accessing Huge Number of Small Files in HDFS  
Shyamli Rao & Amit Joshi

Article R8 13 Pages  Cross-Referencing Cancer Data Using GPU for Multinomial Classification of Genetic Markers  
Srivatsan Sridharan, Abinaya Mahendiran, Sushanth Bhat and Shrisha Rao

Letter to the Editor