Bericht zum 2. Forschungsseminar „Scheduling mit Answer Set Programming“

Durchgeführt von: **Tom Marinovic**

Betreut durch: Professor Dr. Westfeld

31. August 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Informationssicherheit ist ein Thema, das alle Aspekte der Informatik durchzieht, jedoch im umgangssprachlichen Kontext oft nur mit expliziten Anwendungen in Verbindung gebracht wird. Bei der Erwähnung von Antivirusprogrammen, Firewalls und SIEM (Security Information and Event Management)-Lösungen wird oft übersehen, dass all diese Produkte Sicherheit auf bestehenden Infrastrukturen aufsetzen. Es gibt jedoch Szenarien und Sicherheitsanforderungen, die nicht von außen gewährleistet werden können, sondern bereits bei der Entwicklung einer Anwendung bedacht werden müssen. Diese beziehen sich auf die Absicherung gegenüber Programmierfehlern, bei denen ein Angreifer versucht, Verarbeitungsmuster einer Anwendung auszunutzen, um Informationen zu gewinnen und die Ausführung zu beeinflussen. Diese Notwendigkeit besteht für die meisten produktiv eingesetzten Anwendungen und steigt zunehmend für solche im KRITIS-Umfeld. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) definiert KRITIS wie folgt: „Kritische Infrastrukturen (KRITIS) sind Organisationen und Einrichtungen mit wichtiger Bedeutung für das staatliche Gemeinwesen, bei deren Ausfall oder Beeinträchtigung nachhaltig wirkende Versorgungsengpässe, erhebliche Störungen der öffentlichen Sicherheit oder andere dramatische Folgen eintreten würden.“ [1]
Verschiedene Angriffsmethoden nutzen unterschiedliche, häufig vorkommende Fehlermuster aus. Entsprechend müssen zur Vorbeugung verschiedene Methoden gewählt werden. Auf die theoretischen Hintergründe wird im Punkt 2.1 „Sicherheitsrelevante Konzepte“ genauer eingegangen. Im Rahmen des Forschungsseminars wurde eine solche Technik, das statische Planen (englisch „scheduling“) von Prozessschritten, anhand eines praktischen Anwendungsbeispiels näher untersucht. Eine Besonderheit hierbei war, dass das Untersuchungsobjekt primär in Answer Set Programming geschrieben ist, was eine zusätzliche Herausforderung darstellte.

1.2 Untersuchungsobjekt

Für diese Arbeit wurde ein Programm untersucht, das für eine Menge an Prozessen mit bestimmten Eigenschaften statische Pläne generiert. Statisches Planen in diesem Kontext bedeutet, dass für eine Menge an Prozessschritten ein Plan zur Abarbeitung gefunden wird, der sich auf einem gegebenen Prozessor ausführen lässt. Dabei muss nicht nur die Reihenfolge der Schritte, sondern bei modernen Multiprozessorsystemen auch die Verteilung über die einzelnen Kerne berücksichtigt werden. Das statische Planen einer Prozessmenge (im Folgenden auch Taskmenge) ist ein NP-Hartes Problem. Um dieses in einem wirtschaftlich sinnvollen Zeitraum bearbeiten zu können, wurde das Programm in „Answer Set Programming“ (im Folgenden ASP) entwickelt. Zusätzlich wurden Programme zum Generieren und Durchführen von Testplänen für das ASP-Scheduling-Programm in Python erstellt. ASP bietet einige Vorteile zur Lösung des Scheduling-Problems aufgrund der grundlegenden Vorgehensweise, die sich stark von objektorientierten Programmiersprachen wie Python unterscheidet. Grundsätzlich handelt es sich bei ASP um eine deklarative Programmiersprache, die besonders effizient in der Lösung komplexer kombinatorischer Probleme ist, wie eben das statische Planen von Aufgabenmengen. Probleme werden in verschiedenen Datenstrukturen, allen voran Tabellen, abgebildet, die zusätzlich durch Regeln und Constraints beschrieben werden. Anschließend wird mittels eines Solvers eine korrekte Belegung für diese Beschreibung gesucht. Für dieses Projekt wurde „Clingo“ als Grounder und Solver gewählt. Die folgende Grafik zeigt den Programmablauf zu Beginn des Projekts.

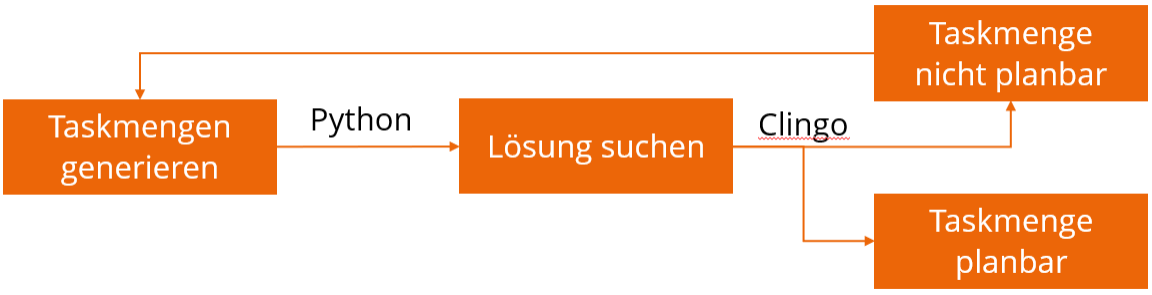

Abbildung 1: IST Zustand Programmablauf

2 Theoretische Vorbereitung

2.1 Sicherheitsrelevante Konzepte

Wie in der Einleitung bereits angedeutet, soll mit dem Erstellen statischer Abarbeitungspläne für CPU-Tasks ein bestimmtes Angriffsmuster unterbunden werden. Dabei handelt es sich um eine Art sogenannter Seitenkanalangriffe (engl. Side-Channel Attack), bei dem ein Angreifer versucht, Informationen über einen Prozess aufgrund seiner Ausführungseigenschaften zu erlangen. Dieses Szenario wird besonders relevant, wenn auf einem Prozessor sicherheitskritische und reguläre Prozesse ausgeführt werden. Gelingt es einem Angreifer, einen regulären Prozess zu übernehmen, kann er die Ausführung des sicherheitskritischen Prozesses beobachten. Dies erlaubt es ihm, Informationen über die Art und den Inhalt des geschützten Prozesses zu gewinnen, indem er dessen Verarbeitungsmuster analysiert. Beispielsweise kann das Verhalten bei korrekten und inkorrekten Authentifizierungsversuchen beobachtet werden, um Rückschlüsse auf das Anmeldeverfahren oder sogar das verwendete Passwort zu ziehen [2] [3]. Die folgende Grafik zeigt ein Beispiel für einen Seitenkanalangriff, bei dem ein Angreifer den Stromverbrauch eines Systems unter verschiedenen Bedingungen beobachtet hat.

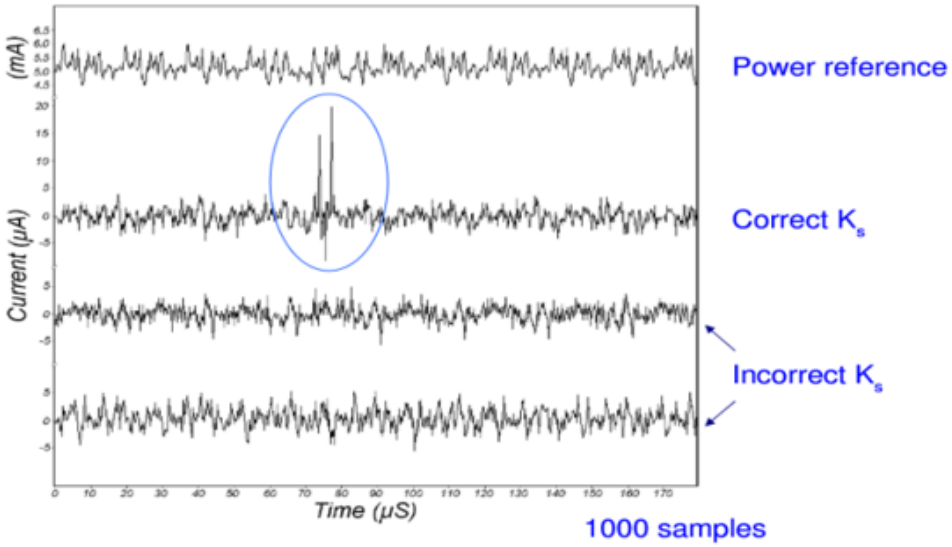

Abbildung 2: Side-Channel Angriff Schaubild

Dieser mögliche Informationsabfluss lässt sich jedoch bei dynamisch geplanten Aufgaben prozessseitig nicht unterbinden, da nicht der Prozess selbst sondern die ausführende Instanz beobachtet wird. Jedoch lässt sich dieses Problem umgehen wenn der Prozess statisch geplant wird, somit kann eine gleiche Ausführungsdauer und verhalten unabhängig der Prozessparameter sichergestellt werden.

Konkret sollen durch statisches Planen, dass Problem der Vorhersagbarkeit (engl. predictability) gelöst werden, indem sich jede Ausführung eines Prozesses gleich zu allen anderen Verhält. Man spricht in diesem Fall auch von „Temporal isolation“ [3]. Zusätzlich kann garantiert werden das sich kritische Tasks und nicht kritische keinen Cache teilen, was sonst zu Informationsverlust führen könnte. Abschließend erlaubt das deterministische Verhalten eines statische geplanten Prozesses ein bessere und genauere Tests, da deutlich weniger Variablefaktoren dies Ausführung beeinflussen. Auch bietet die Optimierung der Ausführungspläne, welche in folgenden Abschnitten weiter besprochen wird, gute Möglichkeiten zur Effizienten Auslastung der gegebene Hardware.
Aber nicht nur gegenüber Seitenkanalangriffen entstehen durchs statische Planen Vorteile, da andere Angriffsmethoden, wie das Ausnutzen von „Race Conditions“, ebenfalls auf Fehlern in der dynamischen Prozessplanung beruhen. Ein großes Problem beim statischen Planen ist jedoch der immense Mehraufwand und die schwierige Planbarkeit, sodass es sich nur für spezielle Anwendungsfälle eignet, bei denen der Mehraufwand beispielsweise aufgrund hoher Sicherheitsanforderungen gerechtfertigt ist. Auch muss die Tatsache berücksichtigt werden, das statische Pläne Hardware gebunden sind und sich nicht auf verschiedene Maschinen unterschiedlicher Bauart übertragen lassen.

2.2 Grundlagen Answer Set Programming

Wie zuvor bereits eingeführt, ist Answer Set Programming (ASP) eine logikorientierte, deklarative Programmiersprache, die sich besonders gut für komplexe Probleme wie das statische Planen eignet. Bei der Entwicklung mit ASP gibt es einige Besonderheiten zu beachten. Zum einen gibt es keinen Compiler wie in marktüblichen Programmiersprachen, sondern einen Grounder und einen Solver. Der Grounder soll die Anweisungen des ASP-Programms „aufspannen“; oft werden Tabellen beschrieben, und das Konstruieren und Füllen dieser Tabellen zur Laufzeit ist die Aufgabe des Grounders. Der Solver wiederum sucht dann nach einer Belegung für dieses System, die die Anforderungen unter den gegebenen Einschränkungen (engl. constraints) erfüllt. Für dieses Projekt wurden die weit verbreiteten Tools „gringo“ als Grounder und „clasp“ als Solver eingesetzt, die zusammen im Programm „clingo“ kombiniert sind [4]. Als deklarative Programmiersprache verfolgt ASP auch einen anderen Ansatz bei der Problemlösung. Es wird nicht die Lösung eines Problems in einer Reihe von Schritten beschrieben, die vom Computer abgearbeitet werden, sondern der Solver sucht selbstständig nach der bestmöglichen Lösung. Dafür ist es notwendig, das Problem sehr genau zu beschreiben, damit der Solver eine Belegung finden kann, die das gegebene Problem löst. Dabei werden sogenannte Constraints eingesetzt, um detaillierte Anforderungen zu formulieren. Allgemein gesprochen ist ein solcher Constraint beim statischen Planen beispielsweise, dass zu einem gegebenen Zeitpunkt nur ein Prozessschritt auf einer CPU stattfinden kann oder dass die Reihenfolge der Prozess-

schritte eines Tasks zwingend eingehalten werden muss. Ein Constraint dient also dazu Modelle zu eliminieren, welche nicht den Vorgaben entsprechen, die von Ihnen definiert werden [5].

3 Entwicklung

3.1 Analyse IST-Zustand

Zu Beginn des Projektes waren mehrere Programme vorhanden. Diese waren wie folgt:
Ein ASP-Programm, das die Anforderungen für einen statischen Plan definiert und für eine gegebene Taskmenge einen solchen Plan sucht (base.asp). Die Taskmenge wird ebenfalls in Form eines ASP-Programms (Testplan.asp) übergeben.
Zusätzlich existieren zwei Python-Programme: eins zur Generierung passender Taskmengen (automatedTestParameter.py), das Testplan.asp-Dateien erzeugt, und eins zur Steuerung der Ausführung, das die generierten Testpläne an das ASP-Programm zur Überprüfung übergibt (run.py).
Der Programmablauf beginnt mit dem Aufruf von run.py, welches in einer Endlosschleife zunächst einen Testplan mittels eines Aufrufs von automatedTestParameter.py erzeugt und diesen anschließend an base.asp übergibt. Das base.asp-Programm wird mit dem generierten Testplan (Testplan.asp) als Parameter aufgerufen und liefert entweder einen Ausführungsplan zurück oder die Aussage, dass die gegebene Taskmenge unter den bestehenden Constraints nicht planbar ist. Die Programmausführung mit der Abfolge der einzelnen Aufrufe lässt sich auch durch das folgende Sequenzdiagramm visualisieren.

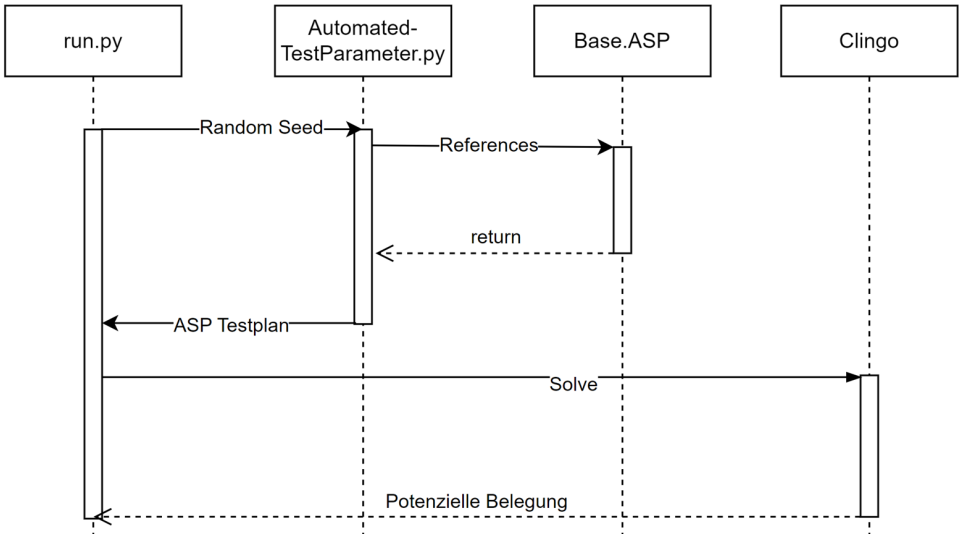

Abbildung 3: UML Sequenzdiagramm der Programmaufrufe

Sollte dieser Prozess länger als 10 Sekunden dauern, wird der Ausführungsprozess abgebrochen und der Plan als Timeout markiert. Ein Timeout bedeutet nicht, dass die Taskmenge nicht planbar ist, sondern dass in der gegebenen Zeit kein Plan gefunden werden konnte, der die Anforderungen von base.asp erfüllt.

Einer der ersten Projektschritte bestand darin, dieses Verhalten zu analysieren und ein Verständnis dafür zu entwickeln, warum manche Taskmengen länger in der Bearbeitung benötigen als andere. Des Weiteren sollte untersucht werden, was eine gute Taskmenge ausmacht und inwieweit die Optimierungsfunktionen von Clingo Einfluss auf das Ergebnis haben. Optimierung bedeutet in diesem Fall, dass Clingo nicht nur eine Belegung sucht, die die gegebene Taskmenge gemäß base.asp erfüllt, sondern auch nachweist, dass diese Belegung optimal ist. Somit kann es passieren, dass für eine Taskmenge mehrere valide Ergebnisse existieren, von denen jedoch nur eines optimal ist.

3.2 Programmanalyse und Test

Zuerst wurde das Verhalten des Programms beim Ändern der Taskmengen-Parameter untersucht. Dabei galt es, die Parameter für CPU-Anzahl, Anzahl der zu generierenden Tasks und maximale Ausführungszeit festzulegen. Die Referenzgrößen hierfür waren 4 CPUs, die 10 Tasks bearbeiten, für die insgesamt 64 Zeitschritte verfügbar sind. Dabei ist zu beachten, dass die Länge der Tasks selbst auch zufällig gewählt wird, aber immer 8, 16, 32 oder 64 beträgt. Erreicht wird dies über die Funktion:

```
planlength_Exponent = random.randint(3,6)
planlength = 2 ** planlength_Exponent
```

Zuerst wird eine zufällige Zahl zwischen 3 und 6 gezogen (3 und 6 eingeschlossen). Anschließend wird 2 mit der generierten Zahl potenziert, um die Länge eines Tasks festzulegen. Dieses Vorgehen wird für jeden Task einzeln durchgeführt. Es wird angenommen, dass jeder Taskschritt stets nur eine Zeiteinheit lang ist und alle Tasks sich trennen lassen, solange die Abarbeitungsreihenfolge eingehalten wird. Mit diesen Parametern wird anschließend über das Python-Programm „automatedTestparameter.py“ eine Menge an Tasks erstellt, die geplant werden sollen. Dabei fällt auf, dass die zufällige Länge der Tasks zu starken Schwankungen in der Komplexität führen kann. Ein Task, der aus 64 Schritten besteht, wird bei einer Bearbeitungszeit von 64 Zeiteinheiten durchgehend eine CPU blockieren. Sollten 4 solche Tasks generiert werden, dann ist die Taskmenge nicht planbar. Dies sollte sich darin widerspiegeln, dass mindestens 22% der generierten Taskmengen nicht planbar sind.
Damit das Verhältnis von planbaren zu nicht planbaren Taskmengen beobachtet werden konnte, wurden ca. 20.000 Pläne generiert und deren Ergebnis als SAT (planbar) oder UNSAT (nicht planbar) klassifiziert. Die folgende Grafik zeigt die Entwicklung von SAT- und UNSAT-Plänen über die Zeit, wobei nach 10 Sekunden ein Timeout für die Lösungssuche entsteht.

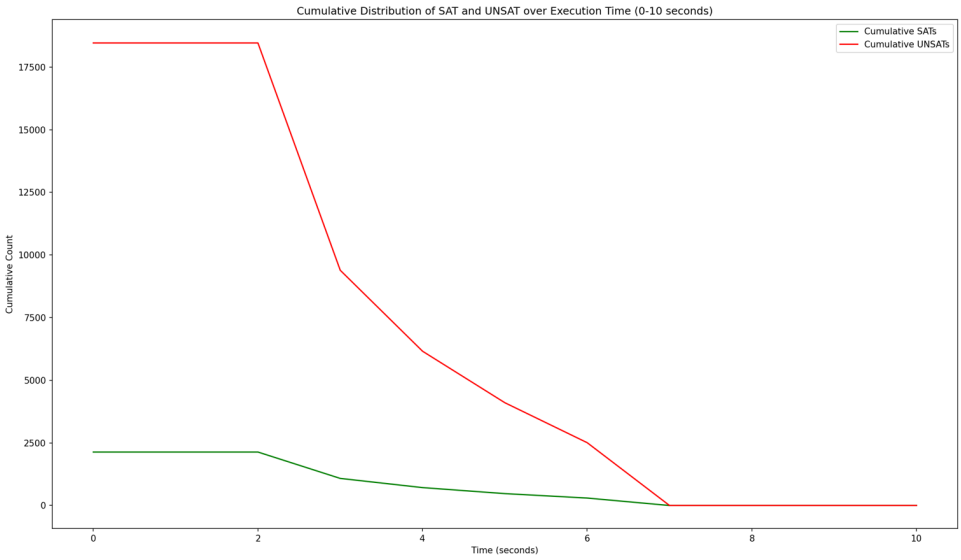

Abbildung 4: Verlauf von SAT und UNSAT über die Zeit

Wie auf der Grafik zu erkennen ist, wird bei einem Großteil der Pläne innerhalb von 2 Sekunden entschieden, ob diese UNSAT oder SAT sind. Der genaue Zeitpunkt variiert je nach verwendeter Hardware und Softwareversion. Die grundlegende Erkenntnis ist jedoch, dass die meisten Pläne innerhalb eines sehr kurzen Zeitraums entschieden werden. Auch das Verhältnis von SAT zu UNSAT lässt sich erkennen, wobei etwa 8,5% der Pläne planbar (SAT), etwa 79,5% nicht planbar (UNSAT) sind und etwa 12% einen Timeout bei der Lösungsfindung verursachen. Eine hohe Anzahl an UNSAT-Plänen war, wie zuvor beschrieben, zu erwarten, jedoch war eine Rate von fast 80% überraschend. Um das Verhalten des Programms weiter zu analysieren, wurden mehr Pläne mit verschiedenen Parametern generiert. Die folgende Tabelle zeigt die Ergebnisse, wobei TL für die maximale Anzahl an Zeitschritten (Tasklength) steht:

	4 CPUs, 10 Parts, 64 TL	4 CPUs, 10 Parts, 128 TL	2 CPUs, 8 Parts, 64 TL	6 CPUs, Parts 12, 64 TL
Anteile UNSAT	79,61%	79,95%	89,00%	78,66%
Anteile SAT	8,42%	8,95%	4,16%	3,82%
Anteile Timeout	11,97%	11,10%	6,84%	17,52%

Abbildung 5: Vergleich verschiedener Testplan Parameter

Die gewählten Parameter wurden explorativ und ohne gezieltes Vorgehen ausgewählt. Interessant ist, dass eine Erhöhung der CPU-Anzahl zu mehr Timeouts führt. Alle Pläne wurden unter der Einstellung geprüft, dass ein Optimum gefunden werden soll. Der Unterschied zwischen einem optimierten und einem nicht optimierten Plan besteht darin, dass bei einer Optimierung versucht wird, die Ressourcen so effizient wie möglich auszulasten, anstatt nur den ersten validen Plan als SAT zu markieren.

3.3 Visualisierung der Pläne

Wie zuvor erwähnt, wurden zum Testen und besseren Verständnis des Programmablaufs mehrere Eigenentwicklungen erstellt. Diese wurden in Python geschrieben und dienen dazu, mehr Details über die erstellten Testpläne zu erhalten. Die Anwendung, die das meiste Verständnis brachte, war eine Visualisierung der Testpläne. Die Hoffnung war, den Unterschied zwischen optimierten, nicht optimierten und nicht planbaren Taskmengen visuell besser analysieren zu können. Betrachtet man nun einen optimierten Plan lassen sich einige Erkenntnisse daraus ableiten. Der Plan wurde unter den Standardparametern 4 CPUs, 10 Tasks und 64 Zeitschritte generiert:

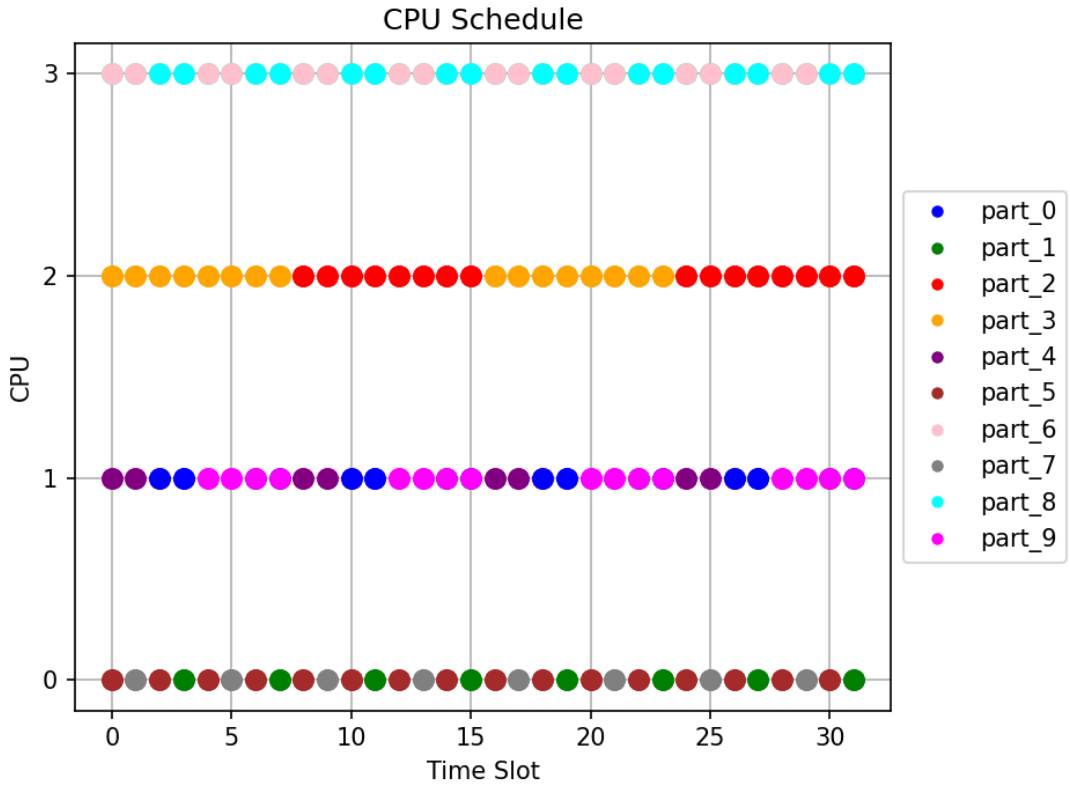

Abbildung 6: Optimal geplante Taskmenge

Die Optimierung lässt sich gut erkennen, da ein Prozess zu jedem Zeitschritt auf jeder CPU ausgeführt wird. Auch wird die maximale Anzahl von 64 Zeitschritten deutlich unterschritten, da alle Task in einem Zeitfenster von 32 Schritten geplant werden konnten. Betrachtet man hingehen eine Plan welcher mit den selben Parametern generiert wurde aber keine Suche nach einer optimalen Belegung seitens Clingo geschehen ist, fallen einige Unterschiede auf:

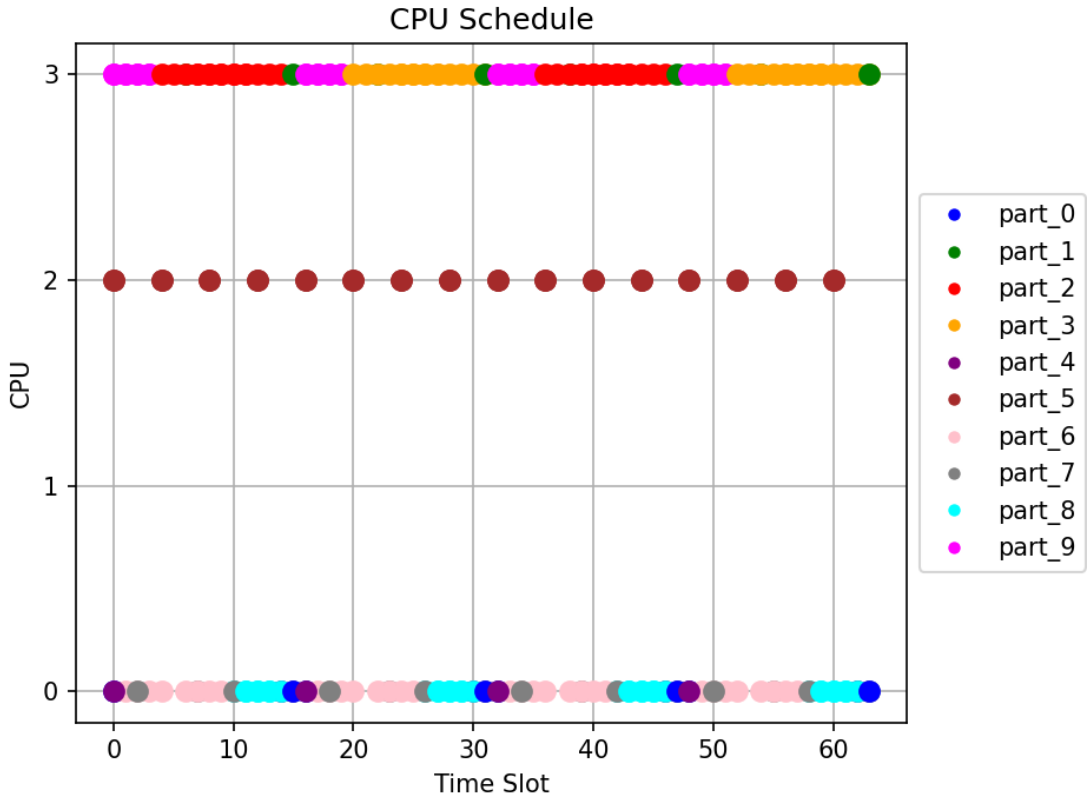

Abbildung 7: Nicht optimal geplante Taskmenge

Die Grafik zeigt das die gegebenen Tasks zwar geplant werden konnten, aber eine Effizienz Steigerung durchaus möglich ist. Zum einen wird CPU 1 gar nicht verwendet, zum anderen wird der Hauptteil der Tasks auf CPU 0 und 3 bearbeitet. Auch die Länge des Plans entspricht dem Maximum von 64 Zeitschritten. Durch die Visualisierung wird nochmals zunehmend erkennbar warum die Erhöhung der Taskanzahl in den Parametern, in deutlich weniger SAT-Plänen resultierte.

4 Abschluss

4.1 Einschränkung

Wie in jeder Arbeit gab es auch in dieser einige Einschränkungen, die erwähnt werden sollten. Eine der größten Herausforderungen war das Einarbeiten und Kennenlernen der Programmiersprache ASP. Aufgrund der grundlegend anderen Herangehensweise im Vergleich zu weiter verbreiteten Sprachen und der fehlenden Erfahrung musste viel Zeit investiert werden, um ein Basisverständnis aufzubauen. Die komplexe Codebasis, die über mehrere Programme verteilt ist, hat dies zusätzlich erschwert. Des Weiteren musste die Funktionsweise der Python-Programme ebenfalls erarbeitet werden. All dies führte dazu, dass keine zielführenden Änderungen am base.asp-Programm vorgenommen werden konnten und sich Anpassungen und Erweiterungen auf die unterstützenden Python-Programme beschränken mussten. Aufgrund dessen war es schwierig, Empfehlungen aus den Analyseergebnissen abzuleiten. Fehlende Vorerfahrung beschränkt auch die Aussagekraft der getroffenen Empfehlungen, da diese theoretischer Natur sind und nicht praktisch überprüft wurden. Zudem wurde die Komplexität des Themas vom Durchführenden unterschätzt, was zu Zeitverlust und schlechter Planung führte. Ein besser strukturiertes Vorgehen und eine klare Zielausrichtung wären von Vorteil gewesen. Der Aufwand für explorative Analysen überstieg den erwarteten Anteil. Auch wenn diese Umstände zur Einschränkung der Projektergebnisse führte, stellten sie doch wichtige Lektionen und Erkenntnisse da, welche in Folgeprojekten beachtet werden können.
Zur Korrektur von Rechtschreibung und Grammatik wurde ein generatives KI-Modell eingesetzt, wobei sämtliche Texte zunächst in Eigenarbeit erstellt und nachträglich überprüft wurden. Das eingesetzte Modell war Chat-GPT 4o entwickelt, trainiert und bereitgestellt von OpenAI.

4.2 Fazit und Weiterentwicklung

Abschließend sollen die gewonnenen Erkenntnisse nochmals zusammengefasst werden. Grundlegend führte das Projekt zu einem signifikanten Wissenszuwachs im Bereich der Logikprogrammierung mit Answer Set Programming sowie beim Vorgehen bei der Analyse bestehender Software. Zudem konnte theoretisches Wissen über Seitenkanal-Angriffe aufgebaut und mit praktischen Beispielen untermauert werden. Durch die Erweiterung um die Möglichkeit der Visualisierung der SAT Pläne konnte der Programmablauf im Vergleich zum Projektbeginn erweitert werden, er lässt sich nun wie folgt darstellen:

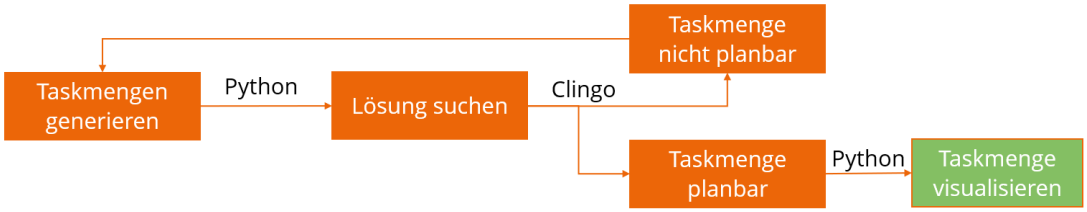

Abbildung 8: Verlauf von SAT und UNSAT über die Zeit

Durch die Visualisierung der generierten Pläne konnte ein deutlich besseres Verständnis für die Arbeitsweise und eigentliche Funktion der Programme erlangt werden. Auch wird der Unterschied zwischen optimierten und nicht optimierten Plänen deutlich erkennbar. Dennoch bietet das Programm noch einige Möglichkeiten zur Weiterentwicklung und Verbesserung. Momentan kann nur ein fertiger Plan übergeben und dargestellt werden, eine Erweiterung gezielt Pläne aus einer Menge von generierter Tastmengen darzustellen wäre denkbar. Auch könnte die Aussagekraft der Grafiken erweitert werden indem mehr Details zu den einzelnen Prozessschritten angezeigt. Auch eine Filteroption für kritische Tasks, solche die die Gesamtlaufzeit des Plans bestimmen wäre möglich. Die erstellen Python Programme zur Analyse der Software sind auf GitHub unter ASPScheduling verfügbar. Die erstellten Programme können frei angepasst und erweitert werden.

4.3 Ausblick

Wie zuvor bereits erwähnt, bietet dieses Thema und ASP allgemein viele Möglichkeiten für Folgeprojekte oder Weiterentwicklungen. In solchen Projekten sollte die steile Lernkurve von ASP eingeplant werden. Das Fehlen von Vorkenntnissen und das Unterschätzen der Komplexität führten in diesem Projekt zu erheblichem Zeitverlust und zu Einschränkungen. Diese Herausforderung könnte durch entsprechende Planung und frühzeitige Einarbeitung in die Programmierung mit ASP verringert werden. Das Projekt hat gezeigt, dass die Implementierung und Optimierung von statisch geplanten Taskmengen mit Answer Set Programming (ASP) ein vielversprechendes Feld ist.

Besonders wertvoll waren die Erkenntnisse zur Visualisierung und Analyse von Testplänen, die halfen, die Unterschiede zwischen optimierten und nicht optimierten Taskmengen besser zu verstehen. Die Analyse der Auswirkungen von verschiedenen Parametern auf die Planbarkeit und die Entdeckung der hohen Rate an nicht planbaren Taskmengen bieten wertvolle Anhaltspunkte für eine tiefgreifendere Untersuchung. Darüber hinaus wurde deutlich, dass die Optimierungsfunktionen von Clingo entscheidend zur Effizienzsteigerung beitragen können, indem sie nicht nur eine gültige Belegung suchen, sondern auch deren Optimalität nachweisen. Die Herausforderungen, die sich dabei aus der Komplexität und der hohen Rate an Timeouts ergibt muss jedoch beachtet werden. Nichtsdestotrotz kann das Projekt als Teilerfolg gewertet werden. Auch wenn die Ergebnisse unter den initialen Erwartungen lagen, entstand doch ein nicht zu vernachlässigender Wissenszuwachs aufgrund der Durchführung. Die Erkenntnisse über die Funktionsweise von ASP und die Herausforderungen bei der Planung und Analyse von Aufgabenmengen bieten eine solide Grundlage für weitere Arbeiten und Entwicklungen in diesem Bereich.

Literatur

[1] *Was sind Kritische Infrastrukturen?* de. URL: https://www.bsi.bund.de/DE/Themen/Regulierte-Wirtschaft/Kritische-Infrastrukturen/Allgemeine-Infos-zu-KRITIS/allgemeine-infos-zu-kritis_node.html.

[2] Mark Randolph und William Diehl. „Power Side-Channel Attack Analysis: A review of 20 years of study for the Layman“. In: *Cryptography* 4.2 (Mai 2020), S. 15. DOI: 10.3390/cryptography4020015. URL: https://doi.org/10.3390/cryptography4020015.

[3] François-Xavier Standaert. *Introduction to Side-Channel attacks*. Dez. 2009, S. 27–42. DOI: 10.1007/978-0-387-71829-3_2. URL: https://doi.org/10.1007/978-0-387-71829-3_2.

[4] Martin Gebser u.a. „A User's Guide to gringo, clasp, clingo, and iclingo“. In: (Okt. 2020). URL: http://wp.doc.ic.ac.uk/arusso/wp-content/uploads/sites/47/2015/01/clingo_guide.pdf.

[5] Vladimir Lifschitz. „Answer Set Programming“. In: (). URL: https://doi.org/10.1007/978-3-030-24658-7.